1513 Assignment 3

Xinyuan Liang

December 2023

1

The assumption of zero-mean Gaussian noise is not very accurate, but it is still reasonable to use it since the mean values are close enough to 0, and the mode error will not effect too much of the algorithm during calculations. The distributions of certain errors, such as v_l and v_r errors for stereo camera, are very inaccurate for Gaussian distribution assumptions since they do not show a symmetric bell curves. The zero-mean assumption is also not that accurate for stereo cameras since the mean value is small but not very close to 0.

To compensate this, the only way is to offset or interpolate certain data with large error. For example, for any v_l data with error larger than 30, interpolate these data and their variance, this will compress all the errors that are larger than 30 down to a reasonable degree, so that the distribution will become a perfect bell curve with zero-mean. Another way is to create a model that multiplies a scalar weight to the data point, so when the error is larger, the weight will decrease, therefore the model will be less dependent on data with large errors and the phase shift will likely make the mean approach zero.

The variance matrices are simply arrays of the square of every standard deviations in those error plots. Therefore,

$$\mathbf{Q}_k = \begin{bmatrix} \delta\mathbf{d}_k \\ \delta\psi_k \end{bmatrix} = \begin{bmatrix} \sigma_{v_x}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{v_y}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{v_z}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{\omega_1}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{\omega_2}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{\omega_3}^2 \end{bmatrix} = \begin{bmatrix} 0.0026 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0021 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0008 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.009 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.017 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1747 \end{bmatrix}. \quad (1)$$

$$\mathbf{R}_k^j = \delta\mathbf{n}_k^j = \begin{bmatrix} \sigma_{u_l}^2 & 0 & 0 & 0 \\ 0 & \sigma_{v_l}^2 & 0 & 0 \\ 0 & 0 & \sigma_{u_r}^2 & 0 \\ 0 & 0 & 0 & \sigma_{v_r}^2 \end{bmatrix} = \begin{bmatrix} 38.005 & 0 & 0 & 0 \\ 0 & 129.854 & 0 & 0 \\ 0 & 0 & 41.963 & 0 \\ 0 & 0 & 0 & 132.508 \end{bmatrix} \tag{2}$$

Note that the size of $\mathbf{R}_k$ will change depends on how many landmarks are observed at each timestep since it's the stack of multiple $\mathbf{R}_k^j$. For example, if two landmarks are observed, simply stack $\mathbf{R}_k^j$ twice.

For Q_k, there are some small error which is due to the different sampling period of the stereo camera. The actual sampling period is around 15.385Hz which is used in our calculations. The sampling period for this camera without any error is 15Hz, 0.385Hz difference will get the variance to be smaller if we don't consider it and use the ideal sampling period, this will likely cause extra error in the later estimations. This affects the IMU estimation since the sampling period is used in the motion model. The model is reconfigured to be dealing with different sampling period changes throughout the process.

2

Before the assemble of the cost function, the model must be rearranged to perfectly fit it.

2.1 Motion Model

To account for a variable sampling period,

$$T_k = t(k) - t(k-1) \tag{3}$$

The discrete-time motion model is still

$$\mathbf{r}_i^{v_k i} = \mathbf{r}_i^{v_{k-1} i} + \mathbf{C}_{v_{k-1} i}^T \mathbf{d}_{v_{k-1}}^{v_k v_{k-1}} \tag{4}$$

$$\mathbf{C}_{v_k i} = \mathbf{\Psi}_{v_k v_{k-1}} \mathbf{C}_{v_{k-1} i} \tag{5}$$

where k is the discrete-time index. The state of the system at timestep k is

$\mathbf{r}_i^{v_k i}$: Translation of vehicle frame with respect to inertial frame, expressed in inertial frame.

$\mathbf{C}_{v_k i}$: Rotation of vehicle frame with respect to inertial frame.

The quantities $\mathbf{d}_{v_{k-1}}^{v_k v_{k-1}}$ and $\mathbf{\Psi}_{v_k v_{k-1}}$ are given by

$$\mathbf{d}_{v_{k-1}}^{v_k v_{k-1}} := \mathbf{v}_{v_{k-1}}^{v_{k-1} i} T_k + \delta \mathbf{d}_k \tag{6}$$

$$\mathbf{\Psi}_{v_k v_{k-1}} := cos\psi_k \mathbf{1} + (1 - cos\psi_k) \left(\frac{\boldsymbol{\psi_k}}{\psi_k}\right) \left(\frac{\boldsymbol{\psi_k}}{\psi_k}\right)^T - sin\psi_k \left(\frac{\boldsymbol{\psi_k}}{\psi_k}\right)^\times \tag{7}$$

$$\boldsymbol{\psi_k} := \boldsymbol{\omega}_{v_{k-1}}^{v_{k-1} i} T_k + \delta\boldsymbol{\psi_k} \rightarrow \boldsymbol{\theta_k} := \boldsymbol{\omega}_{v_{k-1}}^{v_{k-1} i} T_k + \delta\boldsymbol{\theta_k} \tag{8}$$

$$\psi_k := |\boldsymbol{\psi_k}| \rightarrow \theta_k := |\boldsymbol{\theta_k}| \tag{9}$$

note that $\boldsymbol{\theta_k}$ is the k^{th} column of $\boldsymbol{\theta_{v_k i}}$ given in the Matlab data, θ term is only used for calculating initial guesses when groundtruth is needed.

The interoceptive measurements (and associated noise variables) come from the IMU in this case:

$\mathbf{v}_{v_k}^{v_k i}$: Translational velocity of vehicle frame with respect to inertial frame, expressed in vehicle frame.

$\boldsymbol{\omega}_{v_k}^{v_k i}$: Rotational velocity of vehicle frame with respect to inertial frame, expressed in vehicle frame.

$\delta\mathbf{d}_k$: Translational process noise.

$\delta\psi_k$: Rotational process noise.

The state that is going to be estimated is the whole trajectory of poses:

$$\mathbf{x} = \{\mathbf{T}_0, ..., \mathbf{T}_K\} \tag{10}$$

such that for a timestep k,

$$\mathbf{T}_k = \mathbf{T}_{v_k i} = \begin{bmatrix} \mathbf{C}_{v_k i} & -\mathbf{C}_{v_k i}\mathbf{r}_i^{v_k i} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{11}$$

The rotation and translation matrices can be extracted by the same logic.

The inputs are the combinations of translational and rotational velocity from the IMU

$$\mathbf{v} = \{\check{\mathbf{T}}_0, \boldsymbol{\varpi}_1, \boldsymbol{\varpi}_2, ..., \boldsymbol{\varpi}_K\} \tag{12}$$

where $\boldsymbol{\varpi}_k$ is the stack of $\mathbf{v}_{v_k}^{v_k i}$ and $\boldsymbol{\omega}_{v_k}^{v_k i}$ (or poses), i.e. $\begin{bmatrix} \mathbf{v}_{v_k}^{v_k i} \\ \boldsymbol{\omega}_{v_k}^{v_k i} \end{bmatrix}$ and $\check{\mathbf{T}}_0$ here doesn't necessarily mean the first $\mathbf{T}$, it only represents the first initial value of the input in this case.

Firstly, solve the motion model in continuous time frame,

$$\dot{\mathbf{T}} = \boldsymbol{\varpi}^\wedge \mathbf{T} \tag{13}$$

where $\wedge$ symbol is the skew-symmetric operator to transform it to the same dimension as $\mathbf{T}$ and lift it to the Lie algebra, i.e. $\begin{bmatrix} \boldsymbol{\omega}_{v_k}^{v_k i \wedge} & \mathbf{v}_{v_k}^{v_k i} \\ \mathbf{0}^T & 0 \end{bmatrix}$.

Use perturbation by process noise according to

$$\mathbf{T} = exp(\boldsymbol{\epsilon}^\wedge)\bar{\mathbf{T}} \tag{14}$$

$$\boldsymbol{\varpi} = \bar{\boldsymbol{\varpi}} + \delta\boldsymbol{\varpi} \tag{15}$$

to get and transform to the discrete time as following

$$\bar{\mathbf{T}}_k = exp(T_k\bar{\boldsymbol{\varpi}}_k^{\wedge})\bar{\mathbf{T}}_{k-1} = \boldsymbol{\Xi}_k\bar{\mathbf{T}}_{k-1} \tag{16}$$

$$\boldsymbol{\epsilon}_k = exp(T_k\bar{\boldsymbol{\varpi}}_k^{\curlywedge})\boldsymbol{\epsilon}_{k-1} + \mathbf{w}_k = Ad(\boldsymbol{\Xi}_k)\boldsymbol{\epsilon}_{k-1} + \mathbf{w}_k \tag{17}$$

where $\mathbf{w}_k$ is the process noise $\mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$.

2.2 Observation Model

The position of a point on the floor, P_j, expressed in the camera frame is given by

$$\mathbf{p}_{c_k}^{p_jc_k} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{C}_{cv}\left(\mathbf{C}_{v_ki}\left(\boldsymbol{\rho}_i^{p_ji} - \mathbf{r}_i^{v_ki}\right) - \boldsymbol{\rho}_v^{cv}\right) \tag{18}$$

where $\{\mathbf{r}_i^{v_ki}, \mathbf{C}_{v_ki}\}$ is the 'state' of the vehicle, $\{\boldsymbol{\rho}_v^{cv}, \mathbf{C}_{cv}\}$ is the known translation/rotation from the vehicle frame to the camera frame, and $\mathbf{p}_i^{p_ji}$ is the known position of P_j in the inertial frame.

Let $\mathbf{p}_j = \begin{bmatrix} \boldsymbol{\rho}_i^{p_ji} \\ 1 \end{bmatrix}$ and $\mathbf{A} = \begin{bmatrix} \mathbf{C}_{cv} & -\mathbf{C}_{cv}\boldsymbol{\rho}_v^{cv} \end{bmatrix}$, then

$$\mathbf{p}_{c_k}^{p_jc_k} = \mathbf{A}\mathbf{T}_k\mathbf{p}_j \tag{19}$$

The observation model, $\mathbf{g}(\cdot)$, projects $\mathbf{p}_{c_k}^{p_jc_k}$ into the rectified images of an axis-aligned stereo camera:

$$\mathbf{y}_k^j := \mathbf{g}(\mathbf{p}_{c_k}^{p_jc_k}) = \begin{bmatrix} u_l \\ v_l \\ u_r \\ v_r \end{bmatrix} = \frac{1}{z}\begin{bmatrix} f_ux \\ f_vy \\ f_u(x-b) \\ f_vy \end{bmatrix} + \begin{bmatrix} c_u \\ c_v \\ c_u \\ c_v \end{bmatrix} + \delta\mathbf{n}_k^j \tag{20}$$

where $\delta\mathbf{n}_k^j$ is the measurement noise.

This can be written as

$$\mathbf{y}_k^j = \begin{bmatrix} f_u & 0 & c_u & 0 \\ 0 & f_v & c_v & 0 \\ f_u & 0 & c_u & -f_ub \\ 0 & f_v & c_v & 0 \end{bmatrix} \frac{1}{z} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \delta\mathbf{n}_k^j \tag{21}$$

such that the 4×4 matrix is the $\mathbf{M}$ transformation matrix.

Therefore, the measurements from time k_1 to k_2 are

$$\mathbf{y} = \{\mathbf{y}_0^1, ..., \mathbf{y}_0^M, ..., \mathbf{y}_K^1, ..., \mathbf{y}_K^M\} \tag{22}$$

where M is the number of landmarks.

Using perturbations:

$$\mathbf{T}_k = exp(\boldsymbol{\epsilon}_k^{\wedge})\bar{\mathbf{T}}_k \simeq (1 + \boldsymbol{\epsilon}_k^{\wedge})\bar{\mathbf{T}}_k \tag{23}$$

$$\mathbf{p}_j = \bar{\mathbf{p}}_j + \mathbf{D}\boldsymbol{\zeta}_j \tag{24}$$

where

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{25}$$

is a dilation matrix so that our landmark perturbation, $\boldsymbol{\zeta}_j$, is 3×1, and the update term

$$\boldsymbol{\epsilon}_k = ln\left(\mathbf{T}_k\bar{\mathbf{T}}_k^{-1}\right)^{\vee} \tag{26}$$

Let input be

$$\bar{\mathbf{x}} = \{\bar{\mathbf{T}}_0, ..., \bar{\mathbf{T}}_K, \bar{\mathbf{p}}_1, ..., \bar{\mathbf{p}}_M\} \tag{27}$$

and

$$\delta\mathbf{x} = \left[\begin{array}{c} \boldsymbol{\epsilon}_0 \\ \vdots \\ \boldsymbol{\epsilon}_K \\ \hline \boldsymbol{\zeta}_1 \\ \vdots \\ \boldsymbol{\zeta}_M \end{array}\right] \tag{28}$$

In addition,

$$\bar{\mathbf{x}}_k^j = \{\bar{\mathbf{T}}_k, \bar{\mathbf{p}}_j\} \tag{29}$$

and

$$\delta\mathbf{x}_k^j = \begin{bmatrix} \boldsymbol{\epsilon}_k \\ \boldsymbol{\zeta}_j \end{bmatrix} \tag{30}$$

Therefore, with this input,

$$\mathbf{p}_{c_k}^{p_j c_k} = \mathbf{A}\mathbf{T}_k\mathbf{p}_j \simeq \mathbf{A}(1 + \boldsymbol{\epsilon}_k^{\wedge})\bar{\mathbf{T}}_k(\bar{\mathbf{p}}_j + \mathbf{D}\boldsymbol{\zeta}_j) \simeq \left(\bar{\mathbf{T}}_k\bar{\mathbf{p}}_j + \boldsymbol{\epsilon}_k^{\wedge}\bar{\mathbf{T}}_k\bar{\mathbf{p}}_j + \bar{\mathbf{T}}_k\mathbf{D}\boldsymbol{\zeta}_j\right) \tag{31}$$

This can be rearranged as

$$\mathbf{p}_{c_k}^{p_j c_k} = \mathbf{p}(\bar{\mathbf{x}}_k^j) + \mathbf{Z}_k^j\delta\mathbf{x}_k^j \tag{32}$$

where

$$\mathbf{p}(\bar{\mathbf{x}}_k^j) = \mathbf{A}\bar{\mathbf{T}}_k\bar{\mathbf{p}}_j \tag{33}$$

$$\mathbf{Z}_k^j = \left[\left(\mathbf{A}\bar{\mathbf{T}}_k\bar{\mathbf{p}}_j\right)^{\odot} \quad \mathbf{A}\bar{\mathbf{T}}_k\mathbf{D}\right] \tag{34}$$

$$\delta\mathbf{x}_k^j = \begin{bmatrix} \boldsymbol{\epsilon}_k \\ \boldsymbol{\zeta}_j \end{bmatrix} \tag{35}$$

Insert this into the second function

$$\mathbf{g}(\mathbf{x}_k^j) = \mathbf{s}(\mathbf{p}(\mathbf{x}_k^j)) \simeq \mathbf{s}\left(\mathbf{p}(\bar{\mathbf{x}}_k^j) + \mathbf{Z}_k^j \delta\mathbf{x}_k^j\right) \simeq \mathbf{s}(\mathbf{p}(\bar{\mathbf{x}}_k^j)) + \mathbf{S}_k^j \mathbf{Z}_k^j \delta\mathbf{x}_k^j \tag{36}$$

where $\mathbf{s}(\cdot)$ is the function to transform to camera frame, and

$$\mathbf{S}_k^j = \left.\frac{\partial \mathbf{s}}{\partial \mathbf{p}}\right|_{\mathbf{p}(\bar{\mathbf{x}}_k^j)} \tag{37}$$

According to equation (8.404) in the textbook, this Jacobin is actually

$$\frac{\partial s}{\partial \mathbf{p}} = \mathbf{M}\frac{1}{\mathbf{p}_3}\begin{bmatrix} 1 & 0 & -\frac{\mathbf{p}_1}{\mathbf{p}_2} & 0 \\ 0 & 1 & -\frac{\mathbf{p}_2}{\mathbf{p}_3} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{\mathbf{p}_4}{\mathbf{p}_3} & 1 \end{bmatrix} \tag{38}$$

Let

$$\mathbf{g}(\bar{\mathbf{x}}_k^j) = \mathbf{s}(\mathbf{z}(\bar{\mathbf{x}}_k^j)) \tag{39}$$

and

$$\mathbf{G}_k^j = \mathbf{S}_k^j \mathbf{Z}_k^j \tag{40}$$

Then

$$\mathbf{y}_k^j = \mathbf{g}(\mathbf{x}_k^j) + \delta\mathbf{n}_k^j \simeq \mathbf{g}(\bar{\mathbf{x}}_k^j) + \mathbf{G}_k^j \delta\mathbf{x}_j^k + \delta\mathbf{n}_k^j \tag{41}$$

2.3 Errors & Objective Function

For each input,

when $k = 0$:

$$\mathbf{e}_{v,0}(\mathbf{x}) = ln(\check{\mathbf{T}}_0 \mathbf{T}_0^{-1})^\vee = ln(\check{\mathbf{T}}_0 \bar{\mathbf{T}}_0^{-1} exp(-\boldsymbol{\epsilon}_0^\wedge))^\vee \simeq \mathbf{e}_{v,0}(\bar{\mathbf{x}}) - \mathcal{J}(-\mathbf{e}_{v,0}(\bar{\mathbf{x}}))^{-1}\boldsymbol{\epsilon}_0 \tag{42}$$

such that $\mathbf{e}_{v,0}(\bar{\mathbf{x}}) = ln\left(\check{\mathbf{T}}_0 \bar{\mathbf{T}}_0^{-1}\right)^\vee$.

Let $\mathcal{J}(-\mathbf{e}_{v,0}(\bar{\mathbf{x}}))^{-1} = \mathbf{E}_0$, this can be assumed to be the identity (or $\mathbf{1}$) for simplicity when writing the code.

For $k = 1...K$:

$$\mathbf{e}_{v,k}(\mathbf{x}) = ln(\boldsymbol{\Xi}_k \mathbf{T}_{k-1} \mathbf{T}_k^{-1})^\vee \tag{43}$$

where $\boldsymbol{\Xi}_k = exp(T_k \bar{\boldsymbol{\varpi}}_k^\wedge)$.

Use perturbation again,

$$\begin{aligned} \mathbf{e}_{v,k}(\mathbf{x}) = ln\left(\boldsymbol{\Xi}_k exp(\boldsymbol{\epsilon}_{k-1}^\wedge)\bar{\mathbf{T}}_{k-1}\bar{\mathbf{T}}_k^{-1} exp(-\boldsymbol{\epsilon}_k^\wedge)\right)^\vee = \\ ln\left(\boldsymbol{\Xi}_k \bar{\mathbf{T}}_{k-1}\bar{\mathbf{T}}_k^{-1} exp((Ad(\boldsymbol{\Xi}_k)\boldsymbol{\epsilon}_{k-1})^\wedge) exp(-\boldsymbol{\epsilon}_k^\wedge)\right)^\vee \\ \simeq \mathbf{e}_{v,k}(\bar{\mathbf{x}}) + \mathcal{J}(-\mathbf{e}_{v,k}(\bar{\mathbf{x}}))^{-1} Ad(\bar{\mathbf{T}}_k \bar{\mathbf{T}}_{k-1}^{-1})\boldsymbol{\epsilon}_{k-1} - \mathcal{J}(-\mathbf{e}_{v,k}(\bar{\mathbf{x}}))^{-1}\boldsymbol{\epsilon}_k \end{aligned} \tag{44}$$

such that $\mathbf{e}_{v,k}(\bar{\mathbf{x}}) = ln\left(\mathbf{\Xi}_k \bar{\mathbf{T}}_{k-1} \bar{\mathbf{T}}_k^{-1}\right)^\vee$.

Let $\mathbf{F}_{k-1} = \mathcal{J}(-\mathbf{e}_{v,k}(\bar{\mathbf{x}}))^{-1} Ad(\bar{\mathbf{T}}_k \bar{\mathbf{T}}_{k-1}^{-1})$, then

$$\mathbf{e}_{v,k}(\mathbf{x}) = \mathbf{e}_{v,k}(\bar{\mathbf{x}}) + \mathbf{F}_{k-1}\boldsymbol{\epsilon}_{k-1} - \mathbf{E}_k \boldsymbol{\epsilon}_k \tag{45}$$

Similarly, one can assume $\mathcal{J}(-\mathbf{e}_{v,k}(\bar{\mathbf{x}}))^{-1} = \mathbf{E}_k \simeq \mathbf{1}$ for simplicity.

Then the motion objective function for $k = 0$ is

$$J_{v,0}(\bar{\mathbf{x}}) = \frac{1}{2}\mathbf{e}_{v,0}(\bar{\mathbf{x}})^T \check{\mathbf{P}}_0^{-1} \mathbf{e}_{v,0}(\bar{\mathbf{x}}) \tag{46}$$

where $\check{\mathbf{P}}_0$ is the initial covariance matrix one will use at the beginning.

For $k = 1...K$

$$J_{v,k}(\bar{\mathbf{x}}) = \frac{1}{2}\mathbf{e}_{v,k}(\bar{\mathbf{x}})^T \mathbf{Q}_k^{-1} \mathbf{e}_{v,k}(\bar{\mathbf{x}}) \tag{47}$$

For each observation,

$$\mathbf{e}_{y,k}^j = \mathbf{y}_k^j - \mathbf{g}(\mathbf{x}_k^j) \simeq \mathbf{y}_k^j - \mathbf{g}(\bar{\mathbf{x}}_k^j) - \mathbf{G}_k^j \delta \mathbf{x}_k^j \tag{48}$$

such that $\mathbf{e}_{y,k}^j(\bar{\mathbf{x}}) = \mathbf{y}_k^j - \mathbf{g}(\bar{\mathbf{x}}_k^j)$.

Then the observation objective function is

$$J_{y,k}(\bar{\mathbf{x}}) = \frac{1}{2}\sum_{j,k} \mathbf{e}_{y,k}^j(\bar{\mathbf{x}})^T \mathbf{R}_k^{j\,-1} \mathbf{e}_{y,k}^j(\bar{\mathbf{x}}) \tag{49}$$

Therefore, the overall objective function is

$$J(\bar{\mathbf{x}}) = \sum_{k=0}^{K} (J_{v,k}(\bar{\mathbf{x}}) + J_{y,k}(\bar{\mathbf{x}})) \tag{50}$$

Note that the objective function for the whole trajectory is simply calculated with $\mathbf{e}(\mathbf{x})$ terms instead of $\mathbf{e}(\bar{\mathbf{x}})$, both notations' definitions are derived above.

3

Measurement errors can be stacked, therefore

$$\mathbf{e}_{y,k}(\mathbf{x}) = \begin{bmatrix} \mathbf{e}_{y,k}^1(\mathbf{x}) \\ \vdots \\ \mathbf{e}_{y,k}^M(\mathbf{x}) \end{bmatrix} \tag{51}$$

$$\mathbf{e}_{y,k}(\bar{\mathbf{x}}) = \begin{bmatrix} \mathbf{e}_{y,k}^1(\bar{\mathbf{x}}) \\ \vdots \\ \mathbf{e}_{y,k}^M(\bar{\mathbf{x}}) \end{bmatrix} \tag{52}$$

and

$$\mathbf{G}_k = \begin{bmatrix} \mathbf{G}_k^1 \\ \vdots \\ \mathbf{G}_k^M \end{bmatrix} \tag{53}$$

Let

$$\delta \mathbf{x} = \left[\begin{array}{c} \boldsymbol{\epsilon}_0 \\ \boldsymbol{\epsilon}_1 \\ \boldsymbol{\epsilon}_2 \\ \vdots \\ \boldsymbol{\epsilon}_K \\ \hline \boldsymbol{\zeta}_1 \\ \vdots \\ \boldsymbol{\zeta}_M \end{array}\right] \tag{54}$$

$$\mathbf{H} = \left[\begin{array}{cccccc} \mathbf{E}_0 & & & & & \\ -\mathbf{F}_0 & \mathbf{E}_1 & & & & \\ & -\mathbf{F}_1 & \ldots & & & \\ & & \ldots & \mathbf{E}_{K-1} & & \\ & & & -\mathbf{F}_{K-1} & \mathbf{E}_K & \\ \hline \mathbf{G}_0 & & & & & \\ & \mathbf{G}_1 & & & & \\ & & \mathbf{G}_2 & & & \\ & & & \ldots & & \\ & & & & \mathbf{G}_K & \end{array}\right] \tag{55}$$

$$\mathbf{e}(\bar{\mathbf{x}}) = \left[\begin{array}{c} \mathbf{e}_{v,0}(\bar{\mathbf{x}}) \\ \mathbf{e}_{v,1}(\bar{\mathbf{x}}) \\ \vdots \\ \mathbf{e}_{v,K}(\bar{\mathbf{x}}) \\ \hline \mathbf{e}_{y,0}(\bar{\mathbf{x}}) \\ \mathbf{e}_{y,1}(\bar{\mathbf{x}}) \\ \vdots \\ \mathbf{e}_{y,K}(\bar{\mathbf{x}}) \end{array}\right] \tag{56}$$

and

$$\mathbf{W} = diag(\check{\mathbf{P}}_0, \mathbf{Q}_1, ..., \mathbf{Q}_k, \mathbf{R}_0, \mathbf{R}_1, ..., \mathbf{R}_K) \tag{57}$$

Then the objective function can be written as

$$J(\mathbf{x}) \simeq J(\bar{\mathbf{x}}) - \mathbf{b}^T \delta \mathbf{x} + \frac{1}{2} \delta \mathbf{x}^T \mathbf{A} \delta \mathbf{x} \tag{58}$$

where

$$\mathbf{A} = \mathbf{H}^T \mathbf{W}^{-1} \mathbf{H} \tag{59}$$

and

$$\mathbf{b} = \mathbf{H}^T \mathbf{W}^{-1} \mathbf{e}(\bar{\mathbf{x}}) \tag{60}$$

Minimize $J(\mathbf{x})$ w.r.t $\delta\mathbf{x}$ by taking the derivative:

$$\left.\frac{\partial J(\mathbf{x})}{\partial \delta \mathbf{x}^T}\right|_{\bar{\mathbf{x}}^*} = -\mathbf{b} + \mathbf{A}\bar{\mathbf{x}}^* \rightarrow \bar{\mathbf{x}}^* = \mathbf{A}^{-1}\mathbf{b} \tag{61}$$

Iterating between solving for the optimal perturbation and updating the nominal quantities using

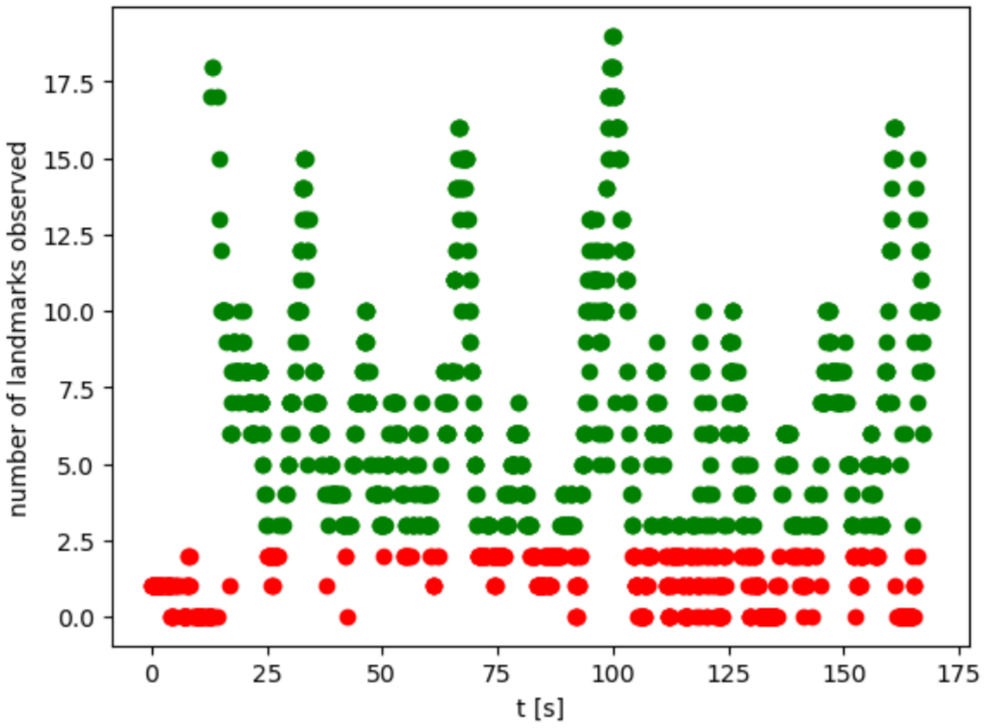
$$\bar{\mathbf{T}}_k \leftarrow exp(\boldsymbol{\epsilon}_k^{*\wedge})\bar{\mathbf{T}}_k \tag{62}$$

$$\bar{\mathbf{p}}_j \leftarrow \bar{\mathbf{p}}_j + \mathbf{D}\boldsymbol{\zeta}_j^* \tag{63}$$

where $\boldsymbol{\epsilon}_k^*$ and $\boldsymbol{\zeta}_j^*$ can be extracted from $\bar{\mathbf{x}}^*$.

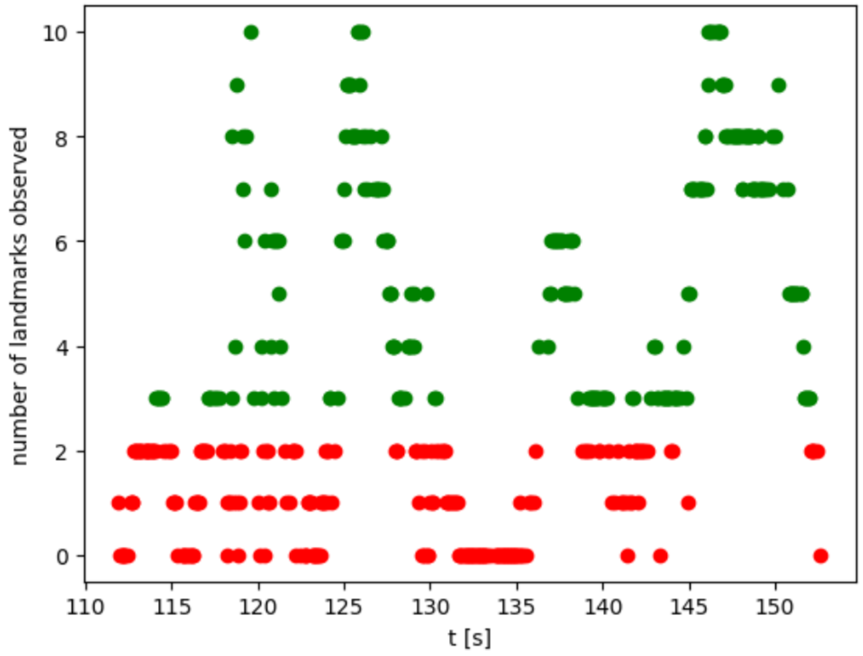
4

The plot of M_t vs. t_k is



where green dots are used for timesteps with at least three observed landmarks, and red dots are used otherwise. Python code is provided in the appendix.

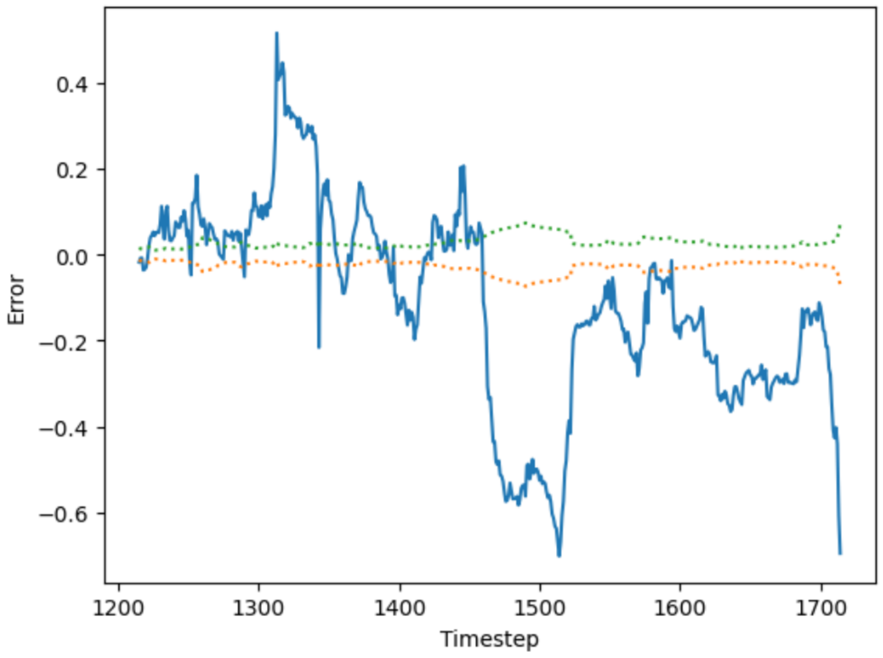
The portion of this graph from 1215 to 1714 is



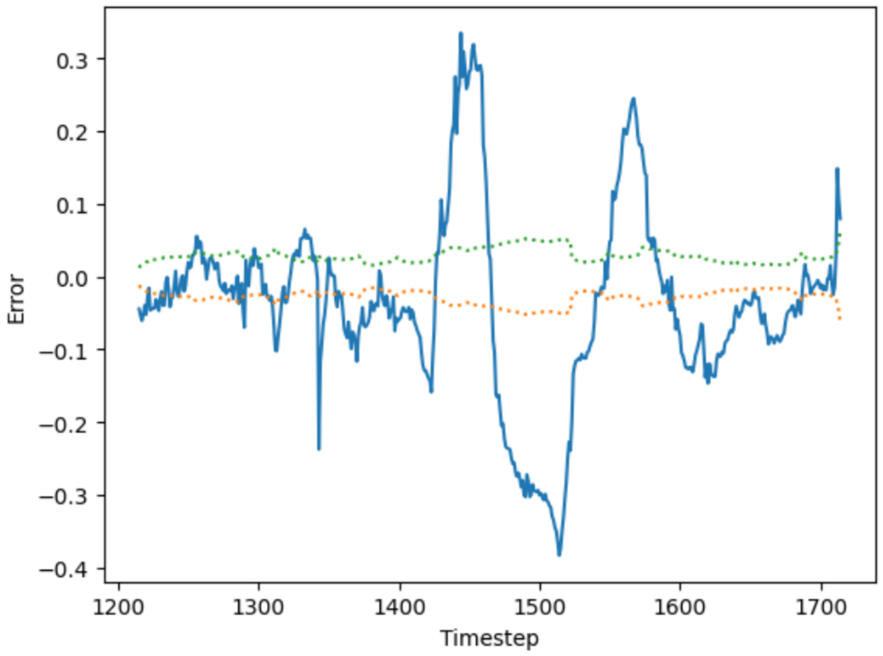
5

(a)

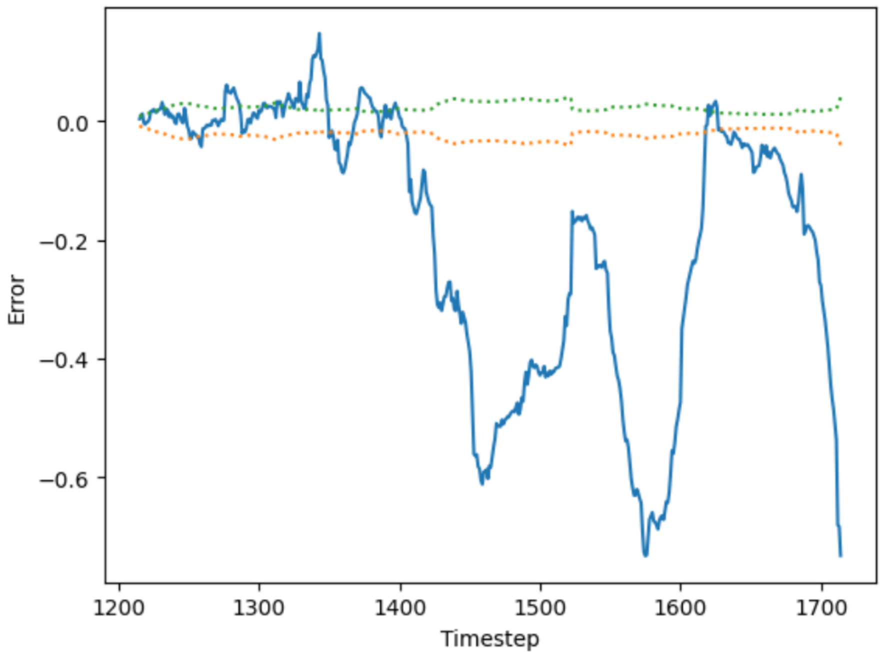
$\delta r_{x,k}$ vs. t_k is



$\delta r_{y,k}$ vs. t_k is

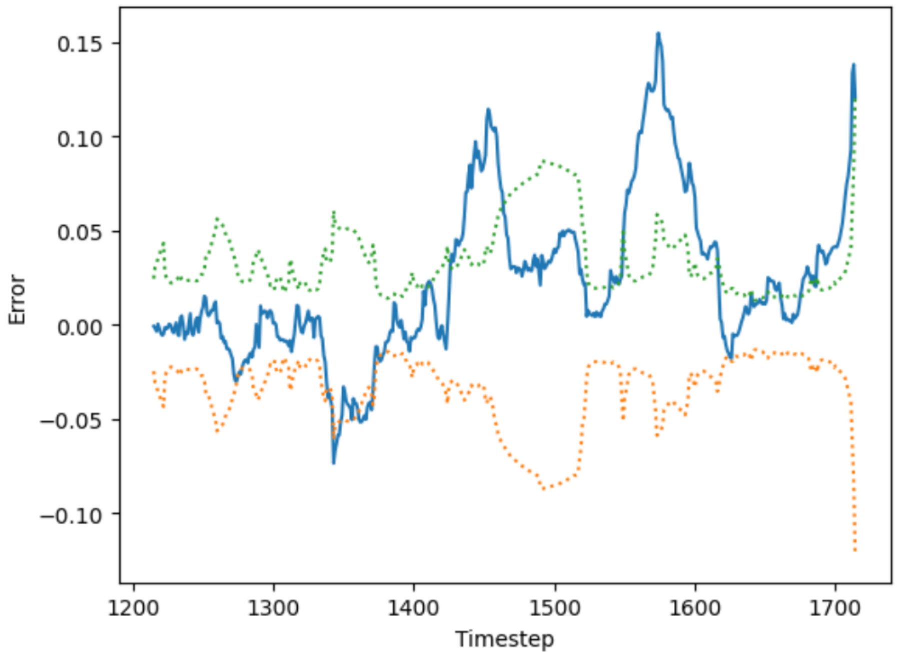


$\delta r_{z,k}$ vs. t_k is

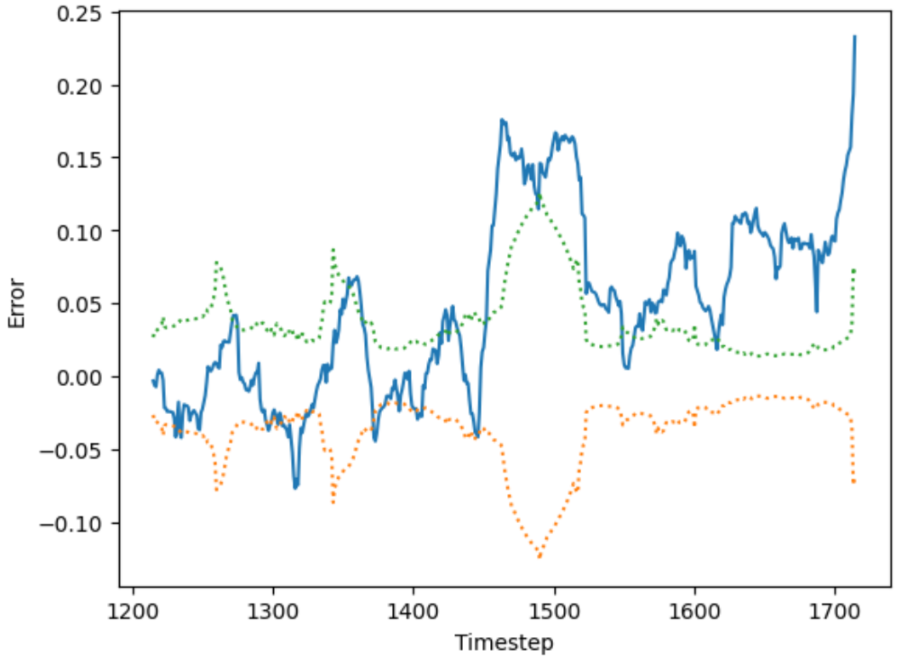


It is clear that for Batch case, the estimation for translation is over confident and the error is a bit large.

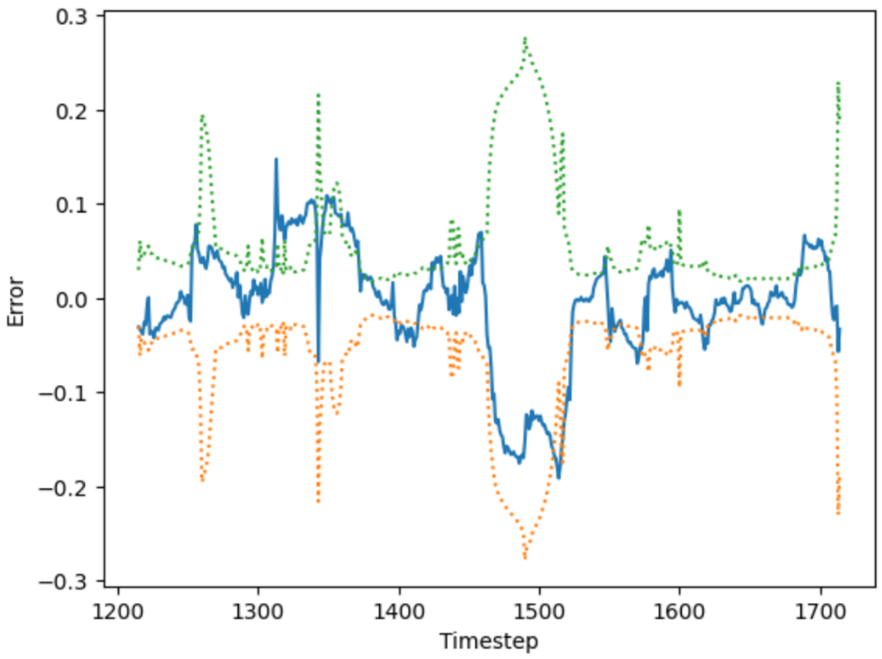
$\delta\theta_{x,k}$ vs. t_k is



$\delta\theta_{y,k}$ vs. t_k is



$\delta\theta_{z,k}$ vs. t_k is

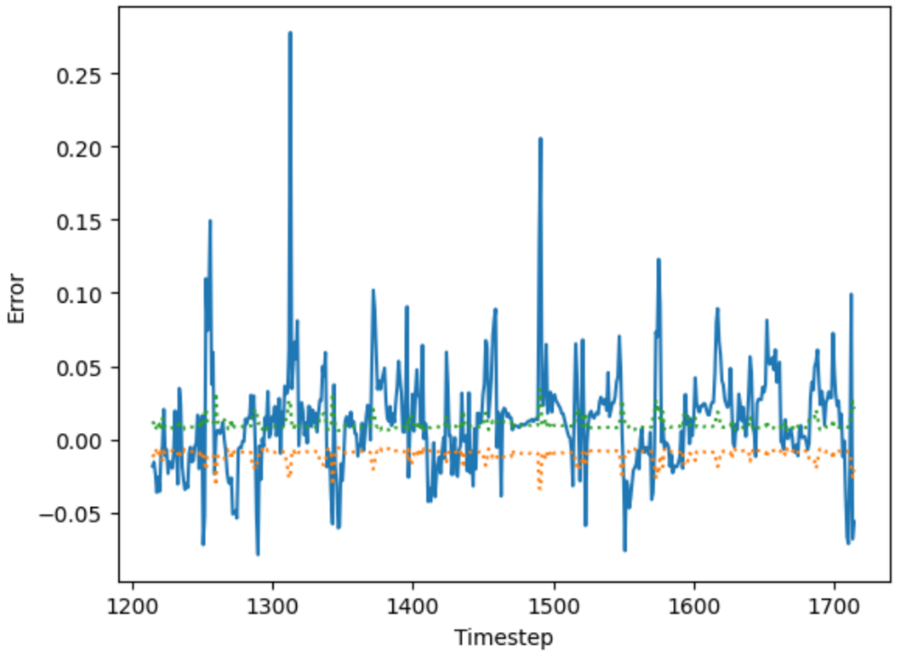


The rotational error is still a bit large, but it shows better accuracy than translational error and it has less over confidence problem.

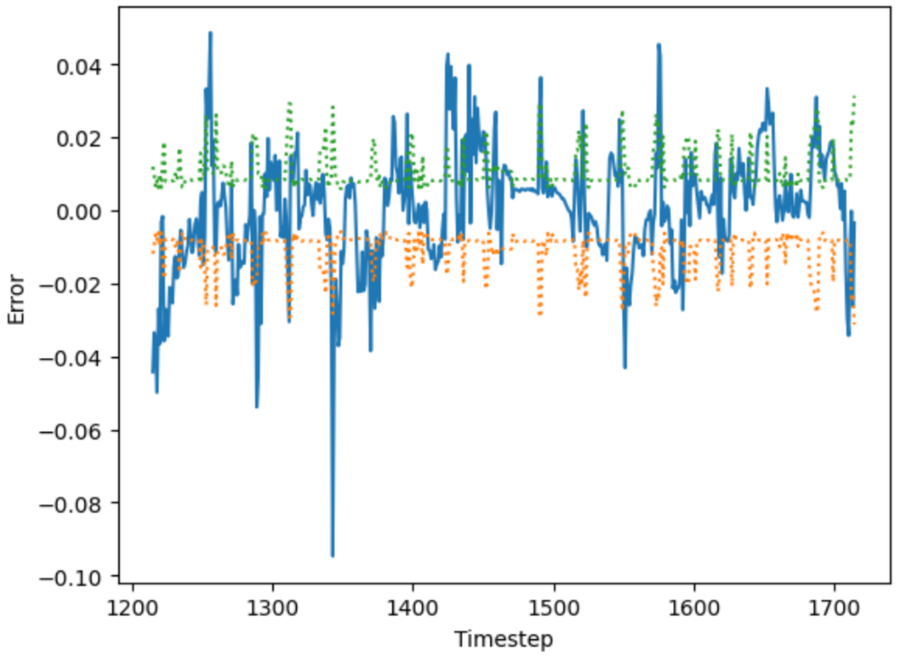
This process takes about 8 seconds.

(b)

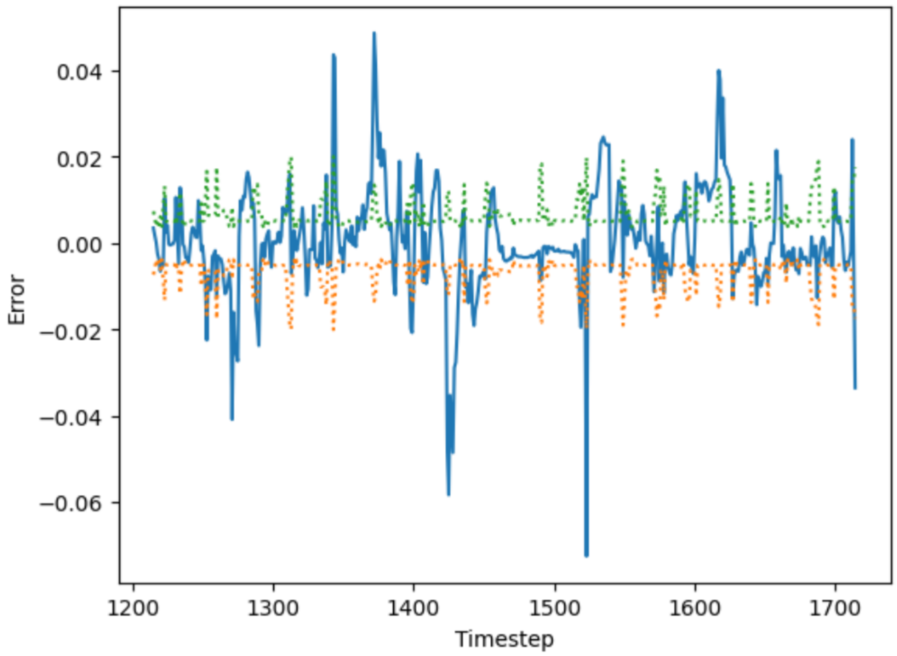
$\delta r_{x,k}$ vs. t_k is



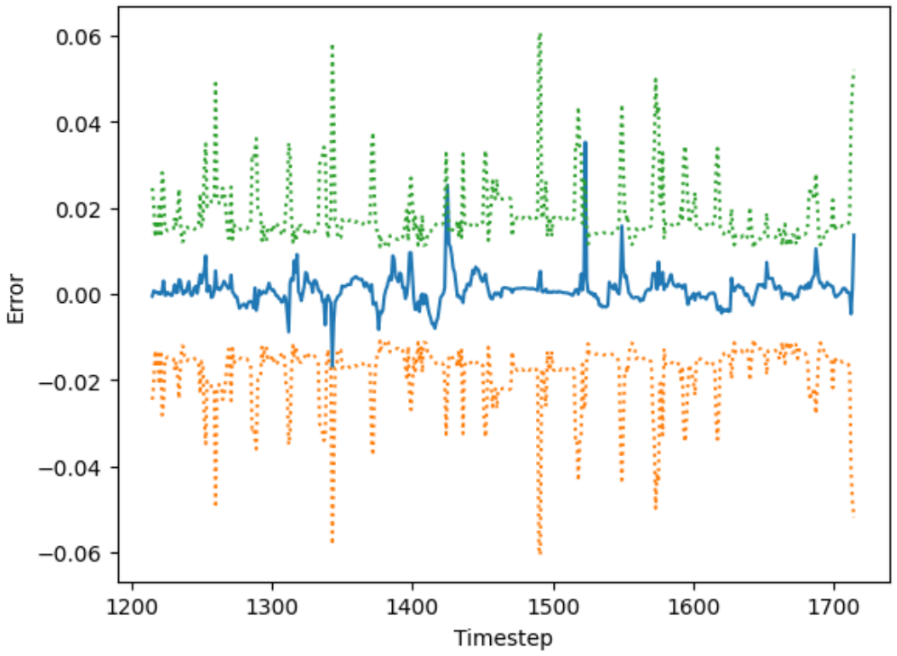
$\delta r_{y,k}$ vs. t_k is



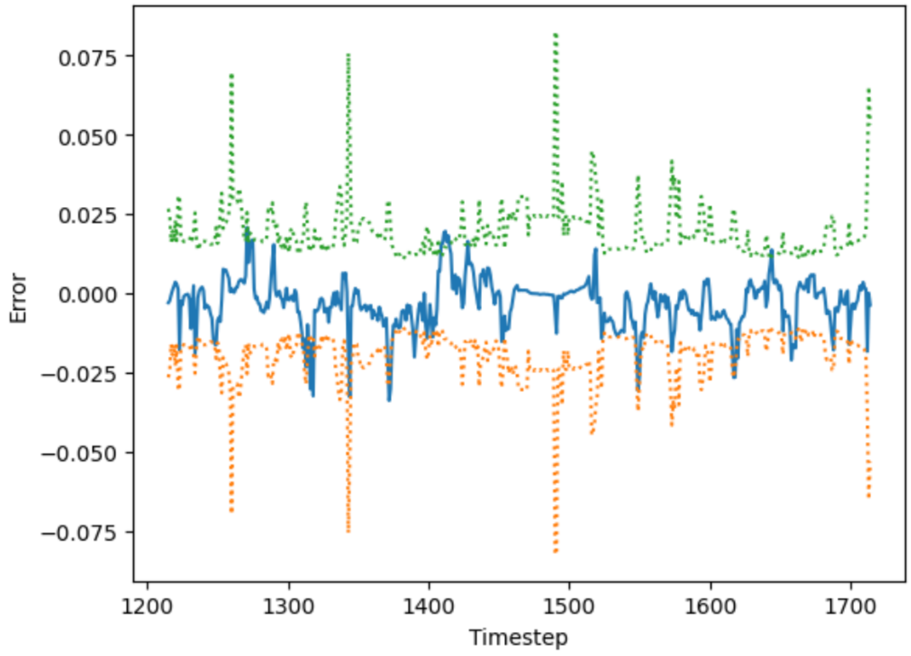
$\delta r_{z,k}$ vs. t_k is



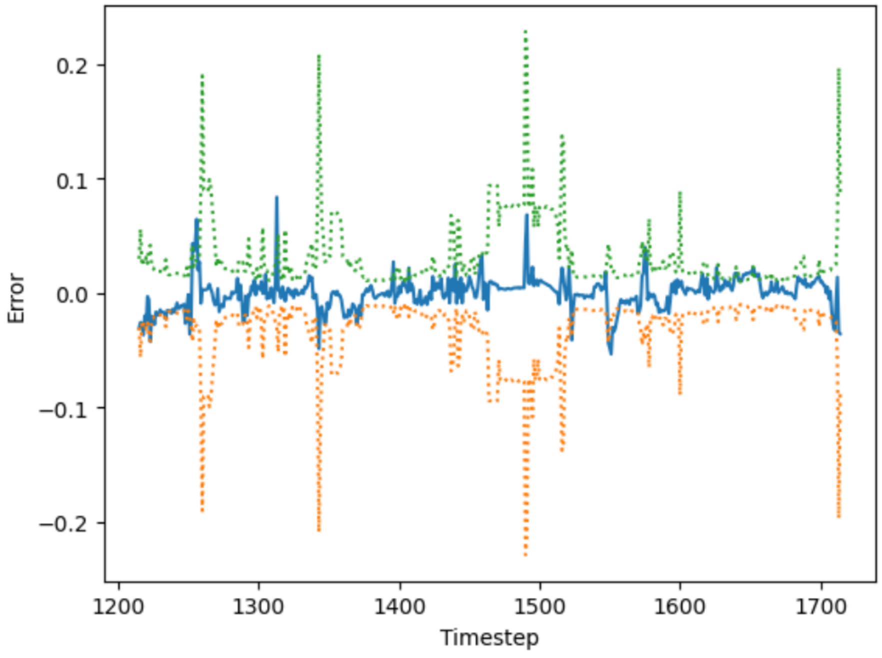
$\delta \theta_{x,k}$ vs. t_k is



$\delta\theta_{y,k}$ vs. t_k is



$\delta\theta_{z,k}$ vs. t_k is

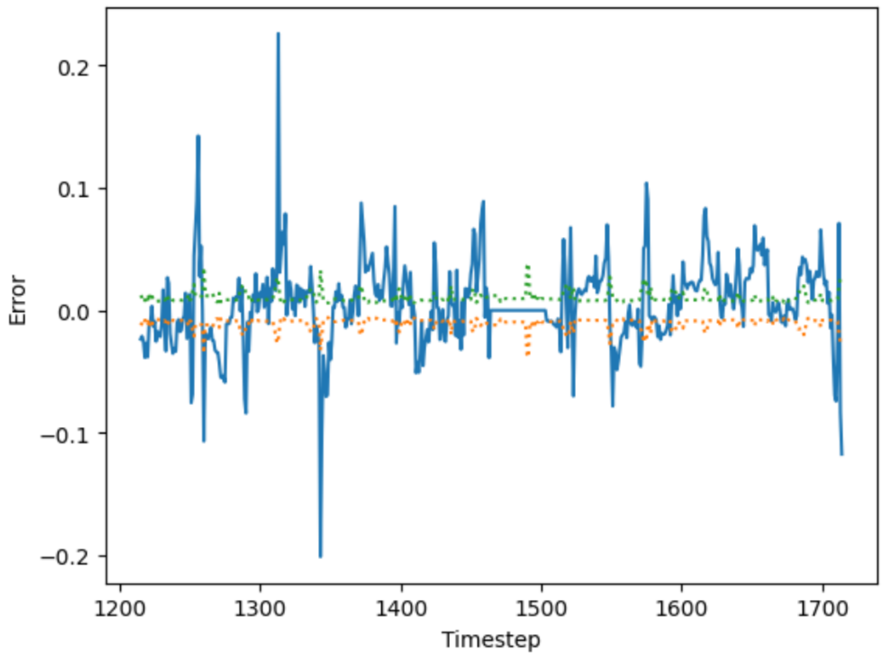


This is the sliding window case with $\kappa = 50$, it is clear that the error is much smaller for both rotation and translation, and the over confidence problem is basically solved at this moment.

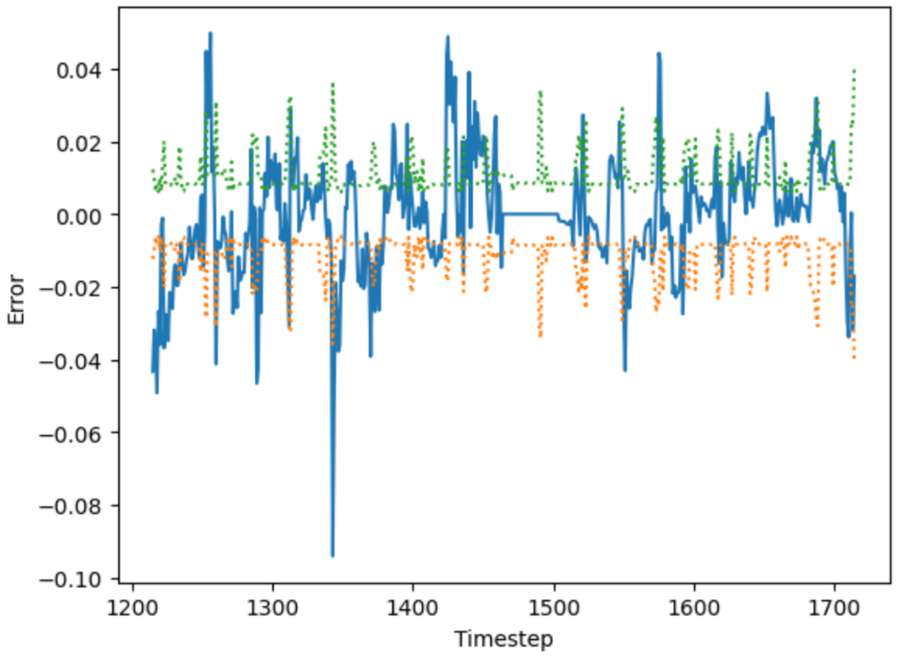
This process takes about 48 seconds.

(c)

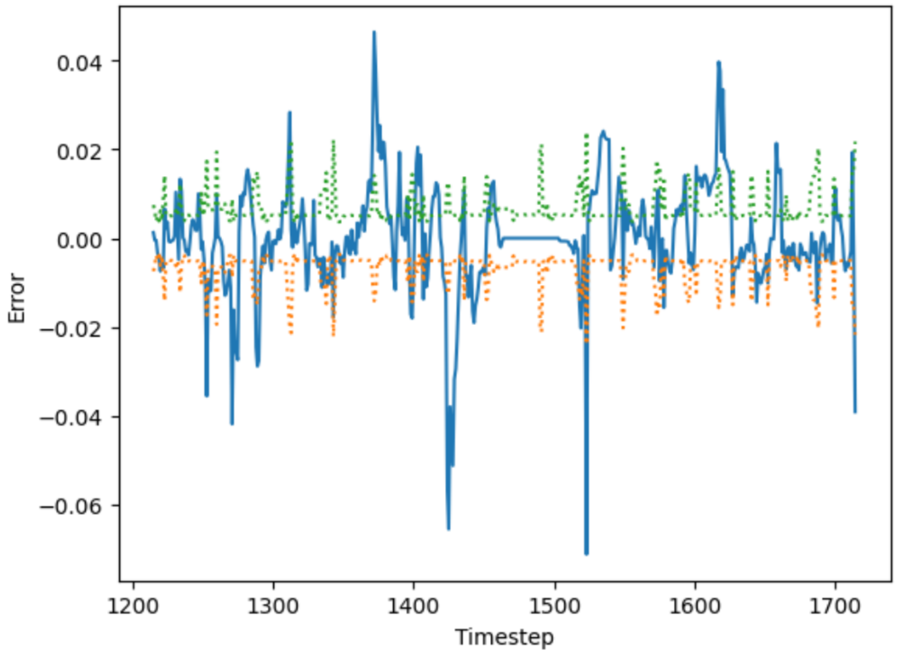
$\delta r_{x,k}$ vs. t_k is



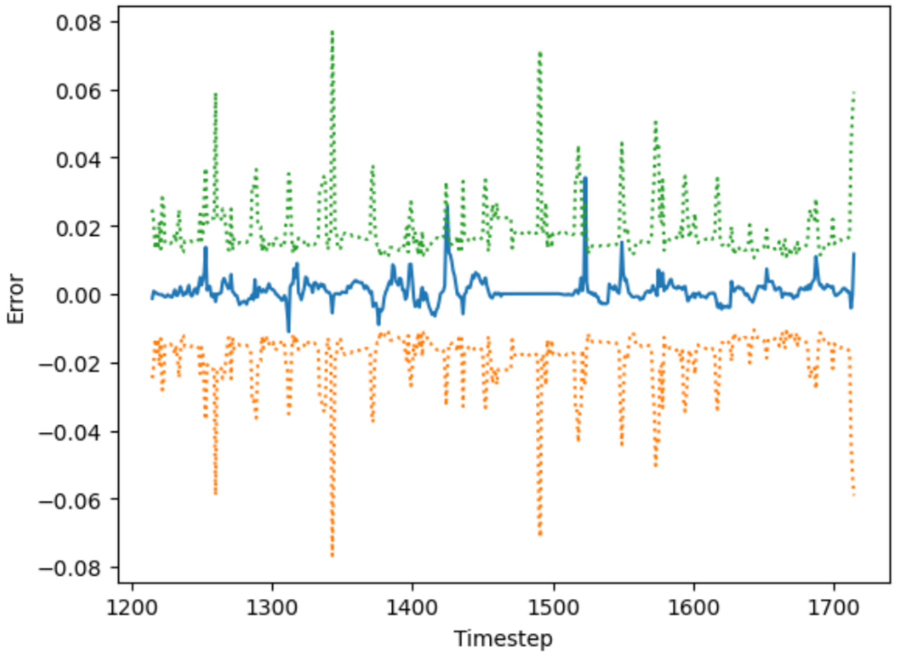
$\delta r_{y,k}$ vs. t_k is



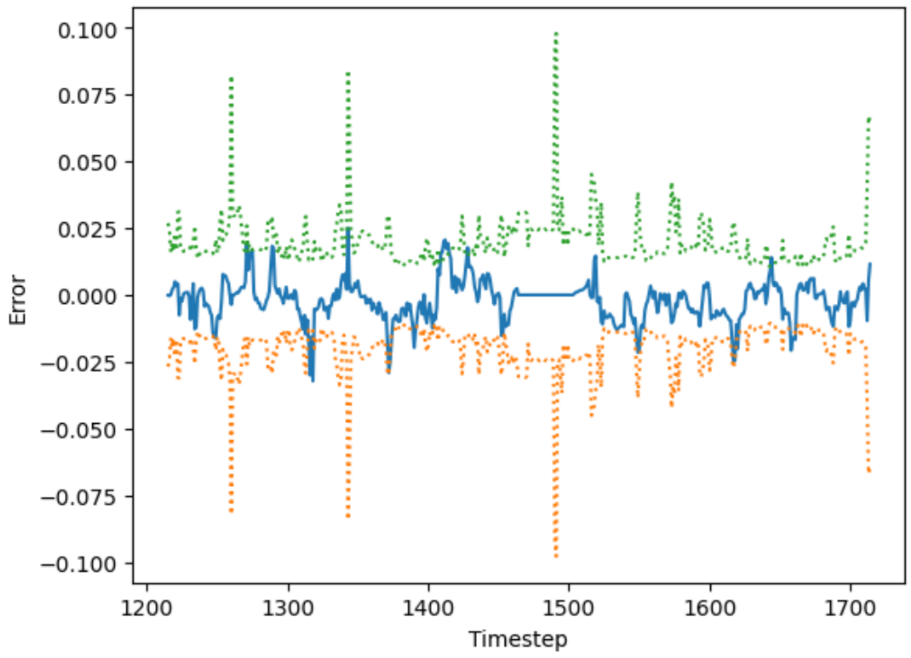
$\delta r_{z,k}$ vs. t_k is



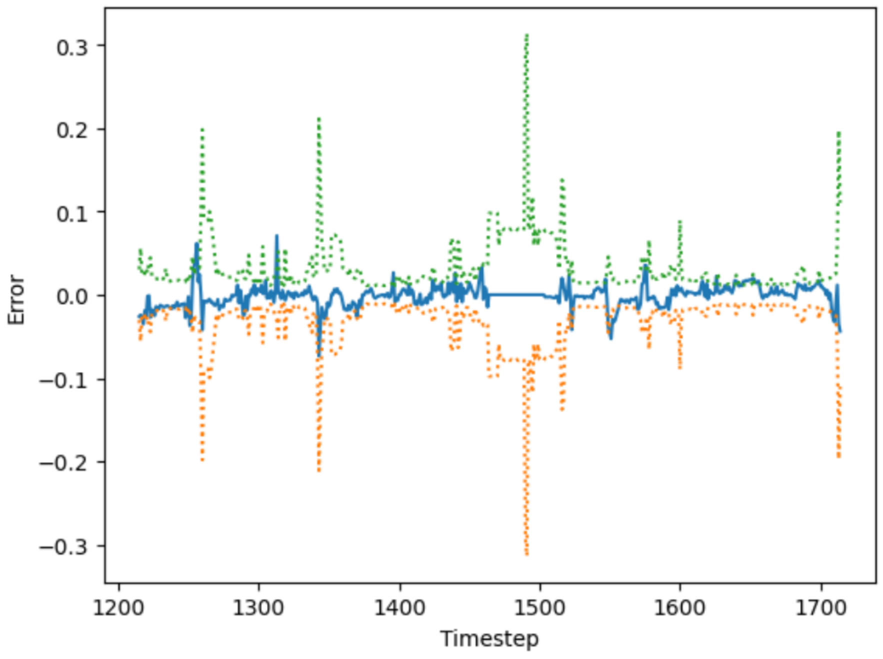
$\delta \theta_{x,k}$ vs. t_k is



$\delta\theta_{y,k}$ vs. t_k is



$\delta\theta_{z,k}$ vs. t_k is



This is the sliding window case with $\kappa = 10$. Although the error does not change much, but the confidence interval clearly increases during this process compared to $\kappa = 50$.

This process takes about 9.3 seconds.

Cooperation: It is clear that sliding window has much better accuracy than the Batch case, but it also requires more computational effort, the larger the κ is, the more computational time it takes.

Compare to Q4, at around the 135 data point of t, there are much fewer landmarks being observed, this contributes to the larger error in the following error plots, which at around timestep 1500, the error will grow rapidly, and the confidence interval will grow larger, this indicates that due to less information, the estimation for that certain period of time is less confident.

6 Appendix

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

dataset = scipy.io.loadmat('dataset3.mat')
theta_vk_i = dataset['theta_vk_i']  # a 3xK matrix where kth column is the
                                    # axis-angle representation of the
                                    # groundtruth value of C_{v_k,i}.
                                    # Use this vector as psi_k in (3.3c) to
                                    # recover the rotation matrix.
r_i_vk_i = dataset['r_i_vk_i']  # a 3xK matrix where the kth column is the
                                # groundtruth value of r_i^{v_k,i} [m]
t = dataset['t']  # a 1xK matrix of time values t(k) [s]
w_vk_vk_i = dataset['w_vk_vk_i']  # a 3xK matrix where kth column is
                                  # the measured rotational velocity,
                                  # w_{v_k}^{v_k,i} [rad/s]
w_var = dataset['w_var']  # a 3x1 matrix of the computed variances
                          # (based on groundtruth) of the
                          # rotational speeds [rad^2/s^2]
v_vk_vk_i = dataset['v_vk_vk_i']  # a 3xK matrix where the kth column
                                  # is the measured translational velocity,
                                  # v_{v_k}^{v_k,i} [m/s]
v_var = dataset['v_var']  # a 3x1 matrix of the computed variances
                          # (based on groundtruth) of the
                          # translational speeds [m^2/s^2]
rho_i_pj_i = dataset['rho_i_pj_i']  # a 3x20 matrix where jth column is
                                    # the position of feature j,
                                    # rho_i^{p_j,i} [m]
y_k_j = dataset['y_k_j']  # a 4xKx20 array of observations, y_k^j [pixels].
                          # All components of y_k_j[:,k,j] will be -1
                          # if the observation is invalid.
y_var = dataset['y_var']  # a 4x1 matrix of the computed variances
                          # (based on groundtruth) of the
                          # stereo measurements [pixels^2]
C_c_v = dataset['C_c_v']  # a 3x3 matrix giving the rotation from
                          # the vehicle frame to the camera frame, C_{c,v}
rho_v_c_v = dataset['rho_v_c_v']  # a 3x1 matrix giving the translation
                                  # from the vehicle frame to the
                                  # camera frame, rho_v^{c,v} [m]
fu = dataset['fu'].item()  # the stereo camera's horizontal focal length,
                           # f_u [pixels]
fv = dataset['fv'].item()  # the stereo camera's vertical focal length,
                           # f_v [pixels]
```

```
cu = dataset['cu'].item()  # the stereo camera's horizontal optical center,
                           # c_u [pixels]
cv = dataset['cv'].item()  # the stereo camrea's vertical optical center,
                           # c_v [pixels]
b = dataset['b'].item()  # the stereo camera baseline, b [m]


# Q4


r_count = 0
lcount = np.zeros(np.shape(t)[1])
for k in range(np.shape(t)[1]):
    for j in range(19):
        if y_k_j[1,k,j] == -1:
            continue
        else:
            r_count += 1
    lcount[k] = r_count
    r_count = 0
for i in range(np.shape(t)[1]):
    if lcount[i] >= 3:
        plt.scatter(t[0,i], lcount[i], color = 'green')
    else:
        plt.scatter(t[0,i], lcount[i], color = 'red')
plt.xlabel('t [s]')
plt.ylabel('number of landmarks observed')


# Q4 from 1215 to 1714
r_count = 0
lcount = np.zeros(500)
k1 = 1215
k2 = 1714
for k in range(k1, k2):
    for j in range(19):
        if y_k_j[1,k,j] == -1:
            continue
        else:
            r_count += 1
    lcount[k - 1215] = r_count
    r_count = 0
for i in range(k1, k2):
    if lcount[i - 1215] >= 3:
        plt.scatter(t[0,i], lcount[i-1215], color = 'green')
    else:
        plt.scatter(t[0,i], lcount[i-1215], color = 'red')
plt.xlabel('t [s]')
plt.ylabel('number of landmarks observed')


# Q5


# Functions

# Wrap to pi function
def omfil(ang):
    return (ang + np.pi) % (2*np.pi) - np.pi

# Convert theta_vk_i to the groundtruth of C_vk_i
def Psi_func(theta):
    theta = theta.reshape(3,1)
    mag = np.linalg.norm(theta)
    # Skew-symmetric operation
    theta_u = (theta/mag).reshape(3,1)
    theta_skews = np.array([[0, -theta_u[2][0], theta_u[1][0]],
                            [theta_u[2][0], 0, -theta_u[0][0]],
                            [-theta_u[1][0], theta_u[0][0], 0]])
    return (np.cos(mag) * np.eye(3)) + ((1 - np.cos(mag)) *
                (theta/mag) @ (theta/mag).T) - (np.sin(mag) * theta_skews)

# Combine translation and rotation to poses
def T_pose(C, r):
    r = r.reshape(3, 1)
    return np.vstack((np.hstack((C, r)), np.array([0, 0, 0, 1])))

# Transformation matrix for camera
A = T_pose(C_c_v, -C_c_v @ rho_v_c_v)

# \wedge operation that lift a vector to lie group
def se3(s):
    if s.shape == (3, 1):
        s_wed = np.zeros((3, 3))
        s_wed[0, 1] = -s[2]
        s_wed[0, 2] = s[1]
        s_wed[1, 0] = s[2]
        s_wed[1, 2] = -s[0]
        s_wed[2, 0] = -s[1]
        s_wed[2, 1] = s[0]
    elif s.shape == (6, 1):
        s_wed = np.zeros((4, 4))
        s_wed[0, 1] = -s[5]
        s_wed[0, 2] = s[4]
        s_wed[1, 0] = s[5]
        s_wed[1, 2] = -s[3]
        s_wed[2, 0] = -s[4]
        s_wed[2, 1] = s[3]
        s_wed[0, 3] = s[0]
        s_wed[1, 3] = s[1]
        s_wed[2, 3] = s[2]
    return s_wed

# \vee operation or inverse \wedge to get the vector form a skew-symmetric form
def se3_inv(S):
    if S.shape == (3, 3):
        s_vee = np.zeros((3, 1))
        s_vee[0] = S[2, 1]
        s_vee[1] = S[0, 2]
        s_vee[2] = S[1, 0]
    elif S.shape == (4, 4):
        s_vee = np.zeros((6, 1))
        s_vee[0] = S[0, 3]
        s_vee[1] = S[1, 3]
        s_vee[2] = S[2, 3]
        s_vee[3] = S[2, 1]
        s_vee[4] = S[0, 2]
        s_vee[5] = S[1, 0]
    return s_vee

# Extract C from the last T
def extC(T):
    return T[0:3, 0:3]

# Extract r from the last T
def extr(T):
    return T[0:3, 3]

# Initial guess
def initial_guess(k1, k2, T_ig=None):
    Tn = np.zeros((t.shape[1], 4, 4))
    if T_ig is None:
        C_0 = Psi_func(theta_vk_i[:,k1])
        r_0 = -C_0 @ r_i_vk_i[:,k1]
        T_ig = T_pose(C_0, r_0)
    Tn[k1] = T_ig
    for i in range(k1+1, k2+1):
        Tk = t[0][i] - t[0][i-1]
        C_kp = extC(Tn[i-1])
        theta_f = omfil(w_vk_vk_i[:, i-1] * Tk)
        Psi = Psi_func(theta_f)
        C_k = Psi @ C_kp
        r_kp = -C_kp.T @ extr(Tn[i-1])
        dr = C_kp.T @ (v_vk_vk_i[:, i-1] * Tk)
        r_k = -C_k @ (r_kp + dr)
        Tn[i] = T_pose(C_k, r_k)
    return Tn

# Camera matrix
M = np.array([[fu, 0, cu, 0], [0, fv, cv, 0], [fu, 0, cu, -fu*b], [0, fv, cv, 0]])

# GN algorithm
def GN(T_op, k1=1215, k2=1714): # Change this to the timestep from each question
    dk = k2 - k1

    Fn = []
    Gn = []
    e_v_kn = []
    Qn = []
    Rn = []
    e_y_kn = []

    # Calculate e_v0
    C_0 = Psi_func(theta_vk_i[:, k1])
    r_0 = -C_0 @ r_i_vk_i[:, k1]
    T_0 = T_pose(C_0, r_0)
    e_v_kn.append(se3_inv(scipy.linalg.logm(T_0 @ np.linalg.inv(T_op[k1]))))

    # Calculate e_v_k
    for i in range(k1+1, k2+1):
        Tk = t[0][i] - t[0][i-1]
        T_opk = T_op[i]
        T_opk_p = T_op[i-1]
        varpi_k = np.hstack((-v_vk_vk_i[:, i], -w_vk_vk_i[:, i])).reshape((6,1))
        # Could be negative of this depends on the reference frame
        Xi_k = scipy.linalg.expm(Tk * se3(varpi_k))
        e_v_k = se3_inv(scipy.linalg.logm(Xi_k @ T_opk_p @ np.linalg.inv(T_opk)))
        e_v_kn.append(e_v_k)
        # Assume E term to be 1
        F_kpad = T_opk @ np.linalg.inv(T_opk_p)
        # Adjoint of F_kpad
        F_kp = np.zeros((6, 6))
        C = extC(F_kpad)
        r = extr(F_kpad)
        F_kp[0:3, 0:3] = C
        F_kp[3:, 3:] = C
        F_kp[:3, 3:] = se3(r.reshape((3, 1))) @ C
        Fn.append(F_kp)

    # Calculate e_y_k
    for k in range(k1, k2+1):
        Tk = t[0][k] - t[0][k-1]
        T_opk = T_op[k]
        G_kn = []
        e_yn = []

        for j in range(20):
            y_j = y_k_j[:, k, j].reshape((4,1))
            if y_j[0] == -1: continue
            p_j = np.vstack((rho_i_pj_i[:, j].reshape((3,1)), np.eye(1)))
            p_j_c = A @ T_opk @ p_j

            # Calculate S_k_j Jacobian
            S_k_j_i = p_j_c
            ds = np.array([[1, 0, -S_k_j_i[0].item()/S_k_j_i[2].item(), 0],
                           [0, 1, -S_k_j_i[1].item()/S_k_j_i[2].item(), 0],
                           [0, 0, 0, 0],
                           [0, 0, -S_k_j_i[3].item()/S_k_j_i[2].item(), 1]])
            S_k_j = M @ ((1/S_k_j_i[2].item())*ds)

            # Circ operation to get Z_k_j matrix
            Z_k_j_circ = T_opk @ p_j
            rho = Z_k_j_circ[:-1]
            eta = Z_k_j_circ[3, 0]
            Z = np.zeros((4, 6))
            Z[:3, :3] = eta * np.eye(3)
            Z[:3, 3:] = -se3(rho)
            Z_k_j = A @ Z

            G_kn.append(S_k_j @ Z_k_j)
            e_yn.append(y_j - M @ (p_j_c/p_j_c[2].item()))
        # In case there's no observation
        if G_kn:
            G_k = np.vstack(G_kn)
            e_y_k = np.vstack(e_yn)
            Gn.append(G_k)
            e_y_kn.append(e_y_k)
        else:
            Gn.append(np.zeros((0,6)))
        # Motion covariance matrix
        Qn.append(Tk**2 * np.diag(np.vstack((v_var, w_var)).squeeze()))
        # Observation covariance matrix
        for z in range(len(G_kn)):
            Rn.append(np.diag(y_var.squeeze()))
    # Calculate H matrix which is assambled by F and G
    H_t = np.eye(dk*6 + 6)
    for i in range(len(Fn)):
        H_t[6*i+6:6*i+12, 6*i:6*i+6] = -Fn[i]
    H_b = scipy.linalg.block_diag(*Gn)
    H = np.vstack((H_t, H_b))

    e_t = np.vstack(e_v_kn)
    # Still in case for no observation
    if len(e_y_kn) == 0:
        e_bar = e_t
    else:
        e_b = np.vstack(e_y_kn)
        e_bar = np.vstack((e_t, e_b))
    # The whole covariance matrix
    W = scipy.linalg.block_diag(*(Qn + Rn))
    np.fill_diagonal(W, 1/W.diagonal())
    # B here is A matrix in the assignment answer since A is used earlier
    B = H.T @ W @ H
    b = H.T @ W @ e_bar
    # Optimal delta x after minimize the cost function derivative
    x_star = np.linalg.inv(B) @ b

    for i in range(dk+1):
        T_op[i+k1] = scipy.linalg.expm(se3(x_star[6*i:6*i+6])) @ T_op[i+k1]

    return T_op, B


# Q5(a)


# Get groundtruth pose
T_true = np.zeros((t.shape[1], 4, 4))
for i in range(1215, 1714+1): # Change this to the timestep from each question
    C_true = Psi_func(theta_vk_i[:, i])
    r_true = -C_true @ r_i_vk_i[:, i]
    T_n = T_pose(C_true, r_true)
    T_true[i] = T_n

# Get initial pose
T_initial = initial_guess(1215, 1714, T_ig=None)
# Change this to the timestep from each question
T_op, B = GN(T_initial, 1215, 1714)
# Change this to the timestep from each question
e_th = []
e_r = []

for k in range(1215, 1714+1): # Change this to the timestep from each question
    C_true = extC(T_true[k])
    C_op = extC(T_op[k])
    r_true = extr(T_true[k])
    r_op = extr(T_op[k])

    e_th.append(se3_inv(np.eye(3) - C_op @ C_true.T))
    e_r.append(r_op - r_true)

e_th = np.array(e_th)
e_r = np.array(e_r)

# Extract covariance matrix
var = np.linalg.inv(B).diagonal()
var_rx = var[0::6]
var_ry = var[1::6]
var_rz = var[2::6]
var_thx = var[3::6]
var_thy = var[4::6]
var_thz = var[5::6]

time = np.arange(1215, 1714+1)


# e_r_x plot
plt.plot(time, e_r[:,0])
plt.plot(time, -3 * np.sqrt(var_rx), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_rx), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()
```

```
# e_r_y plot
plt.plot(time, e_r[:,1])
plt.plot(time, -3 * np.sqrt(var_ry), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_ry), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_r_z plot
plt.plot(time, e_r[:,2])
plt.plot(time, -3 * np.sqrt(var_rz), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_rz), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_x plot
plt.plot(time, e_th[:,0])
plt.plot(time, -3 * np.sqrt(var_thx), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thx), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_y plot
plt.plot(time, e_th[:,1])
plt.plot(time, -3 * np.sqrt(var_thy), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thy), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_z plot
plt.plot(time, e_th[:,2])
plt.plot(time, -3 * np.sqrt(var_thz), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thz), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()
```

```
# Q5(b)


# Get groundtruth pose
T_true = np.zeros((t.shape[1], 4, 4))
for i in range(1215, 1714+1): # Change this to the timestep from each question
    C_true = Psi_func(theta_vk_i[:, i])
    r_true = -C_true @ r_i_vk_i[:, i]
    T_n = T_pose(C_true, r_true)
    T_true[i] = T_n

e_th = []
e_r = []

kappa = 50
k1 = 1215
k2 = 1714
T_initial = initial_guess(1215, 1215+kappa)
# Change this to the timestep from each question
T_op, B = GN(T_initial, 1215, 1215+kappa)
# Change this to the timestep from each question
Tn = np.zeros_like(T_op)
Bn = np.zeros(((1714-1215+1)*6, (1714-1215+1)*6))
# Change this to the timestep from each question
Tn[1215] = T_op[1215] # Change this to the timestep from each question
var = np.linalg.inv(B).diagonal()
Bn[0:6, 0:6] = np.linalg.inv(np.diag(var[0:6]))

for i in range(1215+1, 1714+1): # Change this to the timestep from each question
    T_initial = initial_guess(i, i+kappa, T_ig=Tn[i-1])
    T_op, B = GN(T_initial, i, i+kappa)
    Tn[i] = T_op[i]
    var = np.linalg.inv(B).diagonal()
    Bn[(i-1215)*6:(i-1215)*6+6, (i-1215)*6:(i-1215)*6+6] =
                                np.linalg.inv(np.diag(var[0:6]))
                                # Change this to the timestep from each question

for k in range(1215, 1714+1): # Change this to the timestep from each question
    C_true = extC(T_true[k])
    C_op = extC(Tn[k])
    r_true = extr(T_true[k])
    r_op = extr(Tn[k])

    e_th.append(se3_inv(np.eye(3) - C_op @ C_true.T))
    e_r.append(r_op - r_true)
```

```
e_th = np.array(e_th)
e_r = np.array(e_r)

# Extract covariance matrix
var = np.linalg.inv(Bn).diagonal()
var_rx = var[0::6]
var_ry = var[1::6]
var_rz = var[2::6]
var_thx = var[3::6]
var_thy = var[4::6]
var_thz = var[5::6]

time = np.arange(1215, 1714+1)


# e_r_x plot
plt.plot(time, e_r[:,0])
plt.plot(time, -3 * np.sqrt(var_rx), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_rx), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_r_y plot
plt.plot(time, e_r[:,1])
plt.plot(time, -3 * np.sqrt(var_ry), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_ry), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_r_z plot
plt.plot(time, e_r[:,2])
plt.plot(time, -3 * np.sqrt(var_rz), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_rz), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_x plot
plt.plot(time, e_th[:,0])
plt.plot(time, -3 * np.sqrt(var_thx), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thx), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_y plot
plt.plot(time, e_th[:,1])
plt.plot(time, -3 * np.sqrt(var_thy), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thy), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_z plot
plt.plot(time, e_th[:,2])
plt.plot(time, -3 * np.sqrt(var_thz), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thz), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# Q5(c)


# Get groundtruth pose
T_true = np.zeros((t.shape[1], 4, 4))
for i in range(1215, 1714+1): # Change this to the timestep from each question
    C_true = Psi_func(theta_vk_i[:, i])
    r_true = -C_true @ r_i_vk_i[:, i]
    T_n = T_pose(C_true, r_true)
    T_true[i] = T_n

e_th = []
e_r = []

kappa = 10
k1 = 1215
k2 = 1714
T_initial = initial_guess(1215, 1215+kappa)
# Change this to the timestep from each question
T_op, B = GN(T_initial, 1215, 1215+kappa)
```

```
# Change this to the timestep from each question
Tn = np.zeros_like(T_op)
Bn = np.zeros(((1714-1215+1)*6, (1714-1215+1)*6))
# Change this to the timestep from each question
Tn[1215] = T_op[1215] # Change this to the timestep from each question
var = np.linalg.inv(B).diagonal()
Bn[0:6, 0:6] = np.linalg.inv(np.diag(var[0:6]))

for i in range(1215+1, 1714+1): # Change this to the timestep from each question
    T_initial = initial_guess(i, i+kappa, T_ig=Tn[i-1])
    T_op, B = GN(T_initial, i, i+kappa)
    Tn[i] = T_op[i]
    var = np.linalg.inv(B).diagonal()
    Bn[(i-1215)*6:(i-1215)*6+6, (i-1215)*6:(i-1215)*6+6] =
                                np.linalg.inv(np.diag(var[0:6]))
                                # Change this to the timestep from each question

for k in range(1215, 1714+1): # Change this to the timestep from each question
    C_true = extC(T_true[k])
    C_op = extC(Tn[k])
    r_true = extr(T_true[k])
    r_op = extr(Tn[k])

    e_th.append(se3_inv(np.eye(3) - C_op @ C_true.T))
    e_r.append(r_op - r_true)

e_th = np.array(e_th)
e_r = np.array(e_r)

# Extract covariance matrix
var = np.linalg.inv(Bn).diagonal()
var_rx = var[0::6]
var_ry = var[1::6]
var_rz = var[2::6]
var_thx = var[3::6]
var_thy = var[4::6]
var_thz = var[5::6]

time = np.arange(1215, 1714+1)


# e_r_x plot
plt.plot(time, e_r[:,0])
plt.plot(time, -3 * np.sqrt(var_rx), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_rx), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_r_y plot
plt.plot(time, e_r[:,1])
plt.plot(time, -3 * np.sqrt(var_ry), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_ry), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_r_z plot
plt.plot(time, e_r[:,2])
plt.plot(time, -3 * np.sqrt(var_rz), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_rz), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_x plot
plt.plot(time, e_th[:,0])
plt.plot(time, -3 * np.sqrt(var_thx), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thx), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_y plot
plt.plot(time, e_th[:,1])
plt.plot(time, -3 * np.sqrt(var_thy), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thy), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()


# e_th_z plot
plt.plot(time, e_th[:,2])
plt.plot(time, -3 * np.sqrt(var_thz), linestyle=':')
plt.plot(time, 3 * np.sqrt(var_thz), linestyle=':')
plt.xlabel('Timestep')
plt.ylabel('Error')
plt.show()
```