Shounak Chakraborty
0474348

Assignment 3

1.

Name:- Shounak
474348

Assignment 3

1. $$Q_k = \begin{bmatrix} \sigma^2_{v_x} & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{v_y} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{v_z} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{\omega_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{\omega_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma^2_{\omega_z} \end{bmatrix} * T_k^2$$

$$= \begin{bmatrix} 0.0026 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0.0021 & 0 & \cdots & \cdots & \vdots \\ \vdots & & 0.0008 & & & \vdots \\ \vdots & & & 0.0090 & & \vdots \\ \vdots & & & & 0.0120 & 0 \\ \vdots & \cdots & \cdots & \cdots & 0 & 0.174 \end{bmatrix} * T_k^2$$

$$R_k^{j} = \begin{bmatrix} \sigma^2_{u_l} & 0 & 0 & 0 \\ 0 & \sigma^2_{v_l} & 0 & 0 \\ 0 & 0 & \sigma^2_{u_r} & 0 \\ 0 & 0 & 0 & \sigma^2_{v_r} \end{bmatrix} = \begin{bmatrix} 38.0046 & 0 & 0 & 0 \\ 0 & 129.85 & 0 & 0 \\ 0 & 0 & 41.86 & 0 \\ 0 & 0 & 0 & 132.3 \end{bmatrix}$$

Shounak Chakraborty
0474348

It can be observed that the fitted Gaussians are approximately zero-mean and capture much of the variance in the histograms. Nevertheless, the zero-mean Gaussian noises are reasonable.

Shounak Chakraborty
0474348

2.

2. Translation vector → $r_i^{v_k i}$

Rotation matrix → $C_{v_k i}$

Pose matrix → $T_k = T_{v_k i}$

$$= \begin{bmatrix} C_{v_k i} & -C_{v_k i} r_i^{v_k i} \\ 0^T & 1 \end{bmatrix}$$

The state we want to estimate :-

$$X_{k_1:k_2} = \left\{ r_i^{v_{k_1} i}, C_{v_{k_1} i}, \ldots r_i^{v_{k_2} i}, C_{v_{k_2} i} \right\}$$

$$= \left\{ T_{v_{k_1} i}, \cancel{T_{v_{k_1+1} i}}, \ldots T_{v_{k_2} i} \right\}$$

Translational velocity, $\nu_{v_k}^{i v_k}$

Angular velocity, $\omega_{v_k}^{i v_k}$

$$\varpi = \begin{bmatrix} \nu_{v_k}^{i v_k} \\ \omega_{v_k}^{i v_k} \end{bmatrix}$$

Shounak Chakraborty
0474348

for timestep k_1 to k_2

$$v = \{ \check{T}_{k_1}, \varpi_{k_1+1}, \ldots, \varpi_{k_2} \}$$

$\check{T}_{k_1} \rightarrow$ prior at $t = k_1$

→ at timestep k, M_k landmarks are observed.

Measurements →

$$y = \{ y^1_{k_1}, \ldots y^{M_{k_1}}_{k_1}, \ldots y^1_{k_2} \ldots y^{M_{k_2}}_{k_2} \}$$

Now defining the error terms; for the inputs $\check{T}_{k_1}$ & ϖ_k, we have,

$$e_{v,k}(\cancel{x}) = \begin{cases} \ln(\check{T}_k \cdot T_k^{-1})^{\vee} & k = 1 \\ \ln(\Xi_k T_{k-1} T_k^{-1})^{\vee} & k = k_1+1, \ldots k_2 \end{cases}$$

where, $\Xi_k = \exp(\Delta t_k \varpi_k^{\wedge})$

Shounak Chakraborty
0474348

for the measurement, y_k^j we have

$$e_{y,j,k}(x) = y_k^j - \bar{g}(P_{c_k})$$

$$= y_k^j - \bar{g}(D T_{cv} T_k P_i^{p_j,i})$$

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad T = \begin{bmatrix} C_{cv} & -C_{cv}\rho_v^{cv} \\ 0^T & 1 \end{bmatrix}$$

$$P_i^{p_j,i} = \begin{bmatrix} \rho_i^{p_j,i} \\ 1 \end{bmatrix}$$

The objective f^n that we seek to minimize →

$$J(x_{k_1:k_2}) = \frac{1}{2} e(x_{k_1:k_2})^T W^{-1} e(x_{k_1:k_2})$$

where we stack all the error terms & weighing matrices:-

Shounak Chakraborty
0474348

$$e(x_{k_1:k_2}) = \begin{bmatrix} e_{v,k_1}(x_{k_1:k_2}) \cdots e_{v,k_2}(x_{k_1:k_2}) \\ e_{y,1k_1}(x_{k_1}) \cdots e_{y,m_{k_1}k_1}(x_{k_1}) \cdots \\ e_{y,1k_2}(x_{k_2}) \cdots e_{y,m_{k_2}k_2}(x_{k_2}) \end{bmatrix}^T$$

$$W^{-1} = \mathrm{diag}\left(\check{P}_{k_1}^{-1}\ Q_{k_1+1}^{-1} \cdots Q_{k_2}^{-1}\ R_{k_1}^{1\,-1} \cdots R_{k_1}^{m_{k_1}\,-1} \cdots R_{k_2}^{1\,-1} \cdots R_{k_2}^{m_{k_2}\,-1} \right)$$

Shounak Chakraborty
0474348

3.

8. We linearize the input and measurement errors at the operating point x_{op}.

Consider; $T_k = \exp(\epsilon_k^\wedge)\check{T}_k$

First i/p error

$$e_{v,k_1}(x) = \ln\left(\check{T}_{k_1} T_{k_1}^{-1}\right)^\vee$$

$$= \ln\left(\check{T}_{k_1} \check{T}_{op,k_1}^{-1} \exp(-\epsilon_{k_1}^\wedge)\right)^\vee$$

$$\approx e_{v,k_1}(x_{op}) - \epsilon_{k_1}$$

Later i/p error, the linearizat^n.

$$e_{v,k}(x) = \ln\left(\Xi_k T_{k-1} T_k^{-1}\right)^\vee$$

$$\approx e_{v,k}(x_{op}) + \underbrace{\mathrm{Ad}\left(T_{op,k} T_{op,k-1}^{-1}\right)}_{F_{k-1}} \epsilon_{k-1} - \epsilon_k$$

Shounak Chakraborty
0474348

where, $e_{v,k}(x_{op}) = \ln\left(\Xi_k T_{op,k-1} T_{op,k}^{-1}\right)^\vee$
is the error at operating point

for measurement errors,

$$e_{y,jk}(x) = y_k^j - \bar{g}\left(p_{c_k}^{p_j c_k}\right)$$

$$= y_k^j - \bar{g}\left(D T_{cv} T_k p_i^{p_j,i}\right)$$

$$\approx y_k^j - \bar{g}\left(D T_c \exp\left(\epsilon_k^\wedge\right) T_{op,k} p_i^{p_j,i}\right)$$

$$\approx y_k^j - \bar{g}\left(D T_{cv}\left(1 + \epsilon_k^\wedge\right) T_{op,k} p_i^{p_j,i}\right)$$

$$\approx \underbrace{y_k^j - \bar{g}\left(D T_{cv} T_{op,k} p_i^{p_j,i}\right)}_{e_{y,jk}(x_{op})} - \underbrace{\left.\frac{\partial \bar{g}}{\partial z}\right|_{z = \left(D T_{cv} T_{op,k} p_i^{p_j,i}\right)} \left(D T_{cv}\left(T_{op,k} p_i^{p_j,i}\right)^{\odot}\right)}_{G_{jk}} \epsilon_k$$

Shounak Chakraborty
0474348

Defining the stacked quantities

$$\delta x = \left[\epsilon_{k_1} \; \epsilon_{k_1+1} \; \cdots \; \epsilon_{k_2} \right]^T$$

$$e(x_{op}) = \left[e_{v,k_1}(x_{op}) \cdots e_{v,k_2}(x_{op}) \; e_{y,1k_1}(x_{op}) \cdots e_{y,M_{k_1}k_1}(x_{op}) \cdots e_{y,M_{k_2}k_2}(x_{op}) \right]^T$$

$$H = \begin{bmatrix} 1 & & & & \\ -F_{k_1} & 1 & & & \\ & -F_{k_1+1} & 1 & \ddots & \\ & & & \ddots & 1 \\ G_{1,k_1} & & & & \\ G_{2,k_1} & & & & \\ \vdots & & & & \\ G_{M_{k_1},k_1} & & & & \\ & G_{1,k_1+1} & & & \\ & \vdots & & & \\ & G_{M_{k_1+1},k_1+1} & & & \\ & & \ddots & & \\ & & & & G_{M_{k_2},k_2} \end{bmatrix}$$

Shounak Chakraborty
0474348

$$W = \text{diag}\left(\check{P}_{k_1}, Q_{k_1+1} \cdots Q_{k_2}, R^1_{k_1} \cdots R^{M_{k_1}}_{k_1} \cdots R^1_{k_2} \cdots R^{M_{k_2}}_{k_2}\right)$$

$$J(x) \approx J(x_{op}) - b^T \delta x + \frac{1}{2} \delta x^T A \delta x$$

where,

$$A = H^T W^{-1} H$$

$$b = H^T W^{-1} e(x_{op})$$

Minimizing w.r.t δx,

$$A \, \delta x^* = b$$

for optimal perturbation,

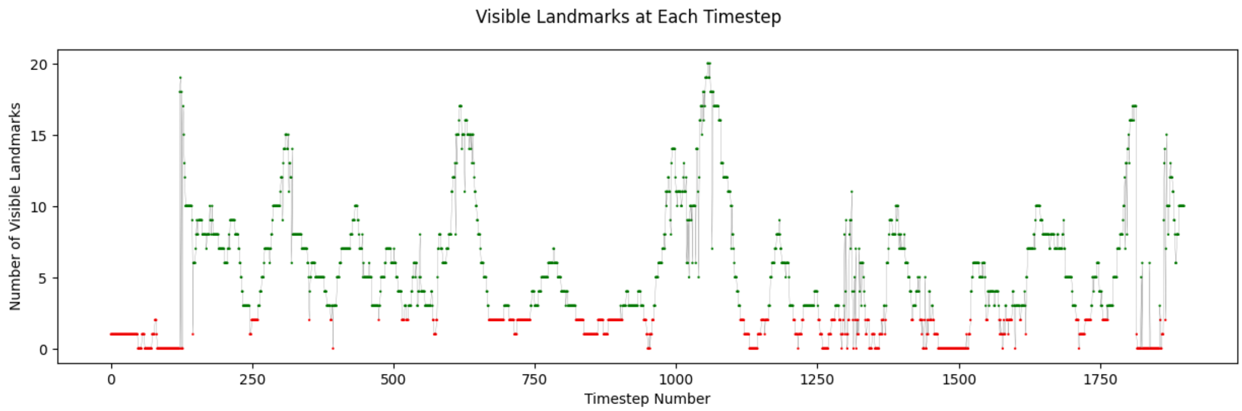
$$\delta x^* = \left[\epsilon^*_{k_1} \; \epsilon^*_{k_1+1} \cdots \epsilon^*_{k_2}\right]^T$$

finally, we update our operating point through the perturbation scheme,

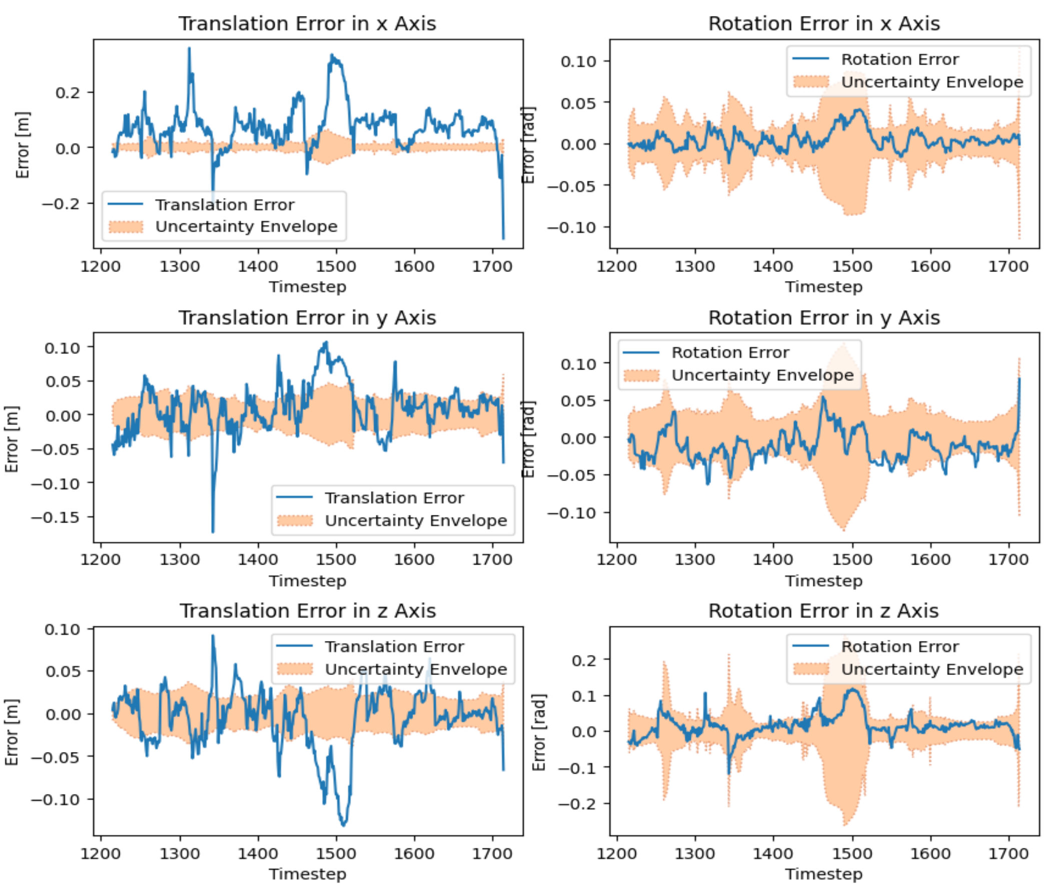
$$T_{op,k} \leftarrow \exp\left(\epsilon^{*\wedge}_k\right) T_{op,k}$$

Shounak Chakraborty
0474348

4.

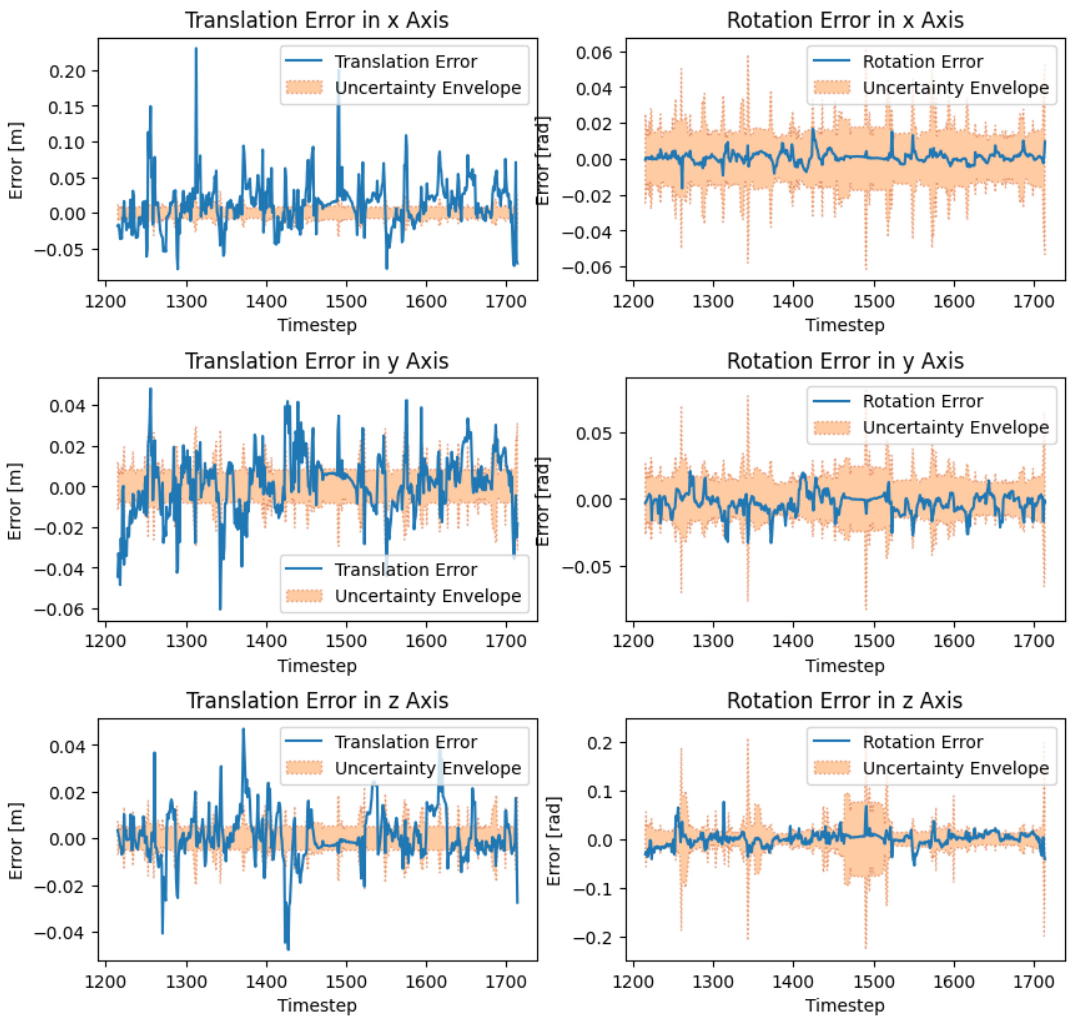


5.a. Batch
Avg Rot Err: 0.017241348974010612
Avg Trans Err: 0.04561352134807128



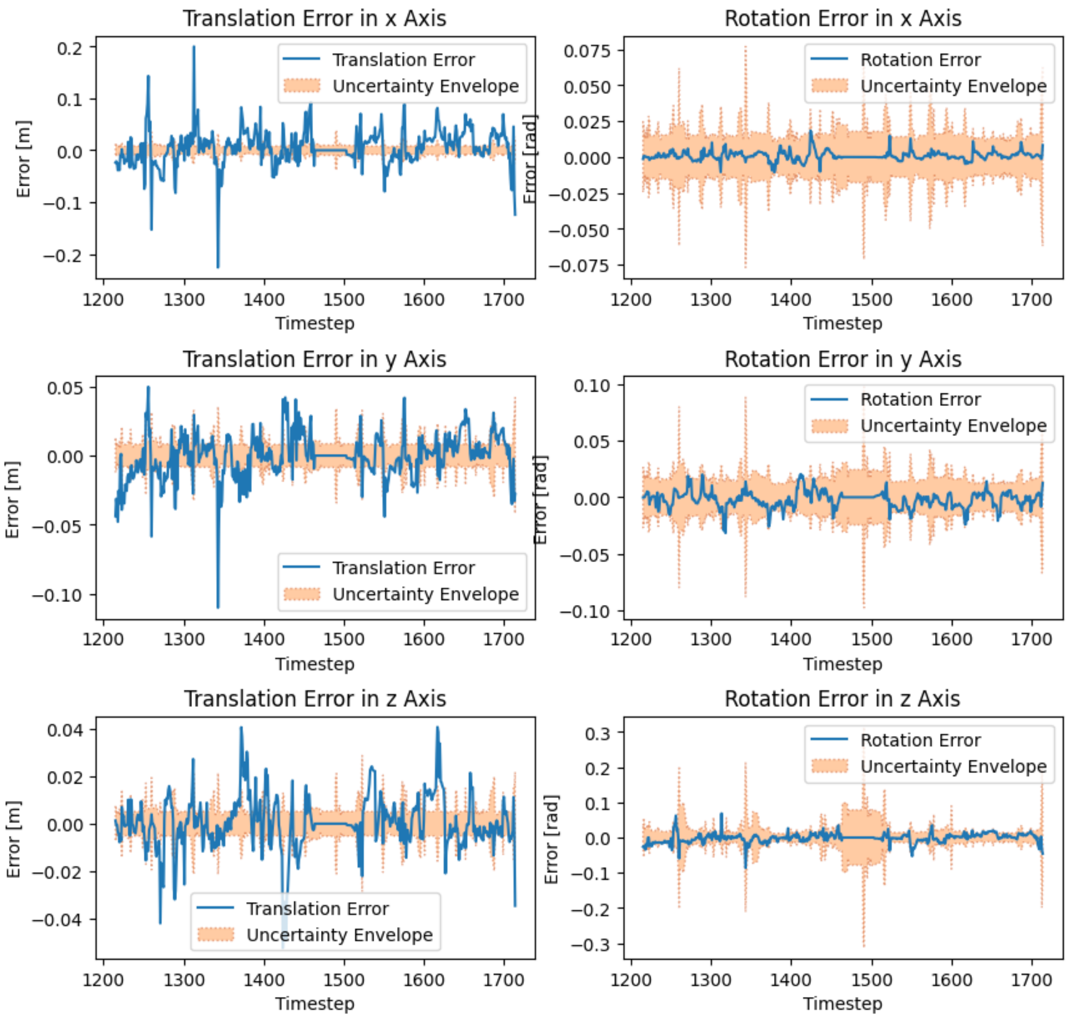
Shounak Chakraborty
0474348

5.b. Sliding , k=50
Avg Rot Err: 0.005923564767723943
Avg Trans Err: 0.015029974237953199



Shounak Chakraborty
0474348

5.c. Sliding Window kappa=10
Avg Rot Err: 0.005487947898589087
Avg Trans Err: 0.014454356639586942



Shounak Chakraborty
0474348

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
matplotlib.use('TkAgg')
import scipy.io
from scipy.linalg import expm, logm, block_diag

from utils import *

#dataset
dataset = scipy.io.loadmat("dataset3.mat")
# 3xK matrix where kth column is the axis-angle representation of the gt
value of C_{v_k,i}
theta_vk_i = dataset['theta_vk_i']
# 3xK matrix where the kth column is the gt value of r_i^{v_k,i}
r_i_vk_i = dataset['r_i_vk_i']
# 1xK matrix of time values t(k)
t = dataset['t']
# 3xK matrix where kth column is the measured rotational velocity
w_{v_k}^{v_k,i}
w_vk_vk_i = dataset['w_vk_vk_i']
# 3x1 matrix of the computed variances of rotational speeds
w_var = dataset['w_var']
# 3xK matrix where the kth column is the measured translational velocity,
v_{v_k}^{v
v_vk_vk_i = dataset['v_vk_vk_i']
# 3x1 matrix of the computed variances of translational speeds
v_var = dataset['v_var']
# 3x20 matrix where jth column is the position of feature j [m]
rho_i_pj_i = dataset['rho_i_pj_i']
# 4xKx20 matrix of observations, y_k^j [pixels]
y_k_j = dataset['y_k_j']
# 4x1 matrix of the computed variances of stereo measurements
y_var = dataset['y_var']
# 3x3 matrix giving the rotation from the vehicle frame to the camera
frame
C_c_v = dataset['C_c_v']
# 3x1 matrix giving the translation from the vehicle frame to the camera
frame, rho_v^{c,v}
```

Shounak Chakraborty
0474348

```
rho_v_c_v = dataset['rho_v_c_v']
fu = dataset['fu'].item()  # stereo camera horizontal focal length
[pixels]
fv = dataset['fv'].item()  # stereo camera vertical focal length [pixels]
cu = dataset['cu'].item()  # stereo camera horizontal optical center
[pixels]
cv = dataset['cv'].item()  # stereo camrea vertical optical center
[pixels]
b = dataset['b'].item()  # stereo camera baseline [m]

# Create transformation matrices
T_c_v = Tmat(C_c_v, -C_c_v @ rho_v_c_v)

# Create camera matrices
M = np.array([
    [fu, 0, cu, 0],
    [0, fv, cv, 0],
    [fu, 0, cu, -fu*b],
    [0, fv, cv, 0]
])
def dfdp(p):
    df = np.array([
        [1, 0, -p[0].item()/p[2].item(), 0],
        [0, 1, -p[1].item()/p[2].item(), 0],
        [0, 0, 0, 0],
        [0, 0, -p[3].item()/p[2].item(), 1]
    ])
    return M @ ((1/p[2].item())*df)

def get_initial_guess(k1=1215, k2=1714, T_gt=None):
    if T_gt is None:
        C_gt = aa_to_C(theta_vk_i[:, k1])
        r_gt = - C_gt @ r_i_vk_i[:, k1]
        T_gt = Tmat(C_gt, r_gt)

    Ts = np.zeros((t.shape[1], 4, 4))
    Ts[k1] = T_gt
    for k in range(k1+1, k2+1):
        Tk = t[0][k] - t[0][k-1]
```

Shounak Chakraborty
0474348

```
        # Rotation
        C_kp = getC(Ts[k-1])
        phi = wrap_to_pi(w_vk_vk_i[:, k-1] * Tk)
        dC = aa_to_C(phi)
        C_k = dC @ C_kp
        # Translation
        r_kp = -C_kp.T @ getR(Ts[k-1])
        d = v_vk_vk_i[:, k-1] * Tk
        dr = C_kp.T @ d
        r_k = r_kp + dr
        r_k = -C_k @ r_k
        # Transformation
        Ts[k] = Tmat(C_k, r_k)

    return Ts

def ggt(k1=1215, k2=1714):
    Ts = np.zeros((t.shape[1], 4, 4))
    for k in range(k1, k2+1):
        C_gt = aa_to_C(theta_vk_i[:, k])
        r_gt = -C_gt @ r_i_vk_i[:, k]
        T_gt = Tmat(C_gt, r_gt)
        Ts[k] = T_gt
    return Ts

def g_batch_est(T_op, k1=1215, k2=1714, iters=7):
    K = k2 - k1
    # T_op = ggt(k1, k2)


    for _ in range(iters):
        Fs = []
        Gs = []
        e_v_ks = []

        e_y_ks = []
        Qs = []
        Rs = []
```

```
        # Error for timestep 0
        C_gt = aa_to_C(theta_vk_i[:, k1])
        r_gt = -C_gt @ r_i_vk_i[:, k1]
        T_gt = Tmat(C_gt, r_gt)
        e_v_ks.append(get_inv_cross_op(logm(T_gt @
np.linalg.inv(T_op[k1]))))

        for k in range(k1+1, k2+1):
            dt = t[0][k] - t[0][k-1]
            T_op_k = T_op[k]
            T_op_kp = T_op[k - 1]
            omega_k = np.hstack((-v_vk_vk_i[:, k], -w_vk_vk_i[:,
k])).reshape((6, 1))

            xi_k = expm(dt * get_cross_op(omega_k))
            e_v_k = get_inv_cross_op(logm(xi_k @ T_op_kp @
np.linalg.inv(T_op_k)))
            e_v_ks.append(e_v_k)

            F_kp = get_Ad(T_op_k @ np.linalg.inv(T_op_kp))
            Fs.append(F_kp)

        for k in range(k1, k2+1):
            dt = t[0][k] - t[0][k-1]
            T_op_k = T_op[k]

            G_ks = []
            e_ys = []
            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 = T_c_v @ T_op_k @ p_j
                G_ks.append(dfdp(p_j_c) @ T_c_v @ get_circ_op(T_op_k @
p_j))
                e_ys.append(y_j - M @ (p_j_c/p_j_c[2].item()))

            if G_ks:
```

```
                Gk = np.vstack(G_ks)
                e_y_k = np.vstack(e_ys)

                Gs.append(Gk)
                e_y_ks.append(e_y_k)
            else:
                Gs.append(np.zeros((0,6)))

            Qs.append(dt**2 * np.diag(np.vstack((v_var, w_var)).squeeze()))
            for _ in range(len(G_ks)):
                Rs.append(np.diag(y_var.squeeze()))

        H_top = np.eye(K * 6 + 6)
        for i in range(len(Fs)):
            H_top[6*i+6:6*i+12, 6*i:6*i+6] = -Fs[i]
        H_bot = block_diag(*Gs)
        H = np.vstack((H_top, H_bot))

        e_top = np.vstack(e_v_ks)
        if len(e_y_ks) == 0:  # In case we have no valid measurements
            e = e_top
        else:
            e_bot = np.vstack(e_y_ks)
            e = np.vstack((e_top, e_bot))
        # print(f"e_top: {np.average(np.abs(e_top))}")
        # #print(f"e_bot: {np.average(np.abs(e_bot))}")
        # print(f"e_bot_max: {np.max(np.abs(e_bot))}")
        # print(f"e_bot_argmax: {np.argmax(np.abs(e_bot))}")

        W = block_diag(*(Qs + Rs))
        Winv = W.copy()
        np.fill_diagonal(Winv, 1/W.diagonal())
        HTWinv = H.T @ Winv

        A = HTWinv @ H
        b = HTWinv @ e

        dx_opt = np.linalg.inv(A) @ b
```

```
        for k in range(K+1):
            T_op[k + k1] = expm(get_cross_op((iters-_)/iters *
dx_opt[6*k:6*k+6])) @ T_op[k + k1]

        print(f"dx_opt: {np.average(np.abs(dx_opt))}")
        print("----------------------------------------")

    return T_op, A

def plot_figure(T_op):
    rs = []
    for i in range(T_op.shape[0]):
        T = T_op[i]
        if T[3][3] == 0: continue
        rs.append(getR(T))
    rs = np.array(rs)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(xs=rs[:, 0], ys=rs[:, 1], zs=rs[:,2])
    plt.show()

    print("done")


def plot_errs(T_op, A, k1, k2):
    T_gt = ggt(k1, k2)

    rot_err = []
    trans_err = []
    for k in range(k1, k2+1):
        C_gt = getC(T_gt[k])
        C_op = getC(T_op[k])

        r_gt = getR(T_gt[k])
        r_op = getR(T_op[k])

        rot_err.append(get_inv_cross_op(np.eye(3) - C_op @ C_gt.T))
        trans_err.append(r_op - r_gt)
```

```
    rot_err = np.array(rot_err)
    trans_err = np.array(trans_err)

    print(f"Avg Rot Err: {np.average(np.abs(rot_err))}")
    print(f"Avg Trans Err: {np.average(np.abs(trans_err))}")

    var = np.linalg.inv(A).diagonal()
    var_tx = var[0::6]
    var_ty = var[1::6]
    var_tz = var[2::6]
    var_rx = var[3::6]
    var_ry = var[4::6]
    var_rz = var[5::6]

    var_ts = [var_tx, var_ty, var_tz]
    var_rs = [var_rx, var_ry, var_rz]


    t = np.arange(k1, k2+1)

    fig, ax = plt.subplots(3, 2, figsize=(10, 20))
    # fig.tight_layout()
    plt.subplots_adjust(hspace=0.4)
    axis = ["x", "y", "z"]
    for i in range(3):
        ax[i][0].plot(t, trans_err[:, i], label="Translation Error")
        ax[i][0].fill_between(t, -3 * np.sqrt(var_ts[i]), +3 *
np.sqrt(var_ts[i]), edgecolor='#CC4F1B',
                          facecolor='#FF9848', alpha=0.5, linestyle=':',
label='Uncertainty Envelope')
        ax[i][0].set_xlabel("Timestep")
        ax[i][0].set_ylabel("Error [m]")
        ax[i][0].set_title(f"Translation Error in {axis[i]} Axis")
        ax[i][0].legend()

        ax[i][1].plot(t, rot_err[:, i], label="Rotation Error")
        ax[i][1].fill_between(t, -3 * np.sqrt(var_rs[i]), +3 *
np.sqrt(var_rs[i]), edgecolor='#CC4F1B',
```

```
                           facecolor='#FF9848', alpha=0.5, linestyle=':',
label='Uncertainty Envelope')
       ax[i][1].set_xlabel("Timestep")
       ax[i][1].set_ylabel("Error [rad]")
       ax[i][1].set_title(f"Rotation Error in {axis[i]} Axis")
       ax[i][1].legend()
   plt.show()

def q4():
   valids = np.where(y_k_j == -1, 0, 1)
   num_valids = valids[0, :, :].sum(-1)
   colors = np.array(['g' if num>=3 else 'r' for num in num_valids])
   t_sq = t.squeeze()

   fig, ax = plt.subplots(figsize=(15,4))
   fig.suptitle('Visible Landmarks at Each Timestep')
   ax.scatter(np.arange(0, t_sq.shape[0]), num_valids, s=0.5, c=colors)
   ax.plot(np.arange(0, t_sq.shape[0]), num_valids, linewidth=0.1, c='k')
   ax.yaxis.get_major_locator().set_params(integer=True)
   ax.set_ylabel("Number of Visible Landmarks")
   ax.set_xlabel("Timestep Number")
   plt.savefig("q4.png", bbox_inches='tight')

   fig, ax = plt.subplots(figsize=(15, 4))
   fig.suptitle('Visible Landmarks at Each Timestep for Timesteps
1215-1714')
   ax.scatter(np.arange(1215, 1715), num_valids[1215:1715], s=0.5,
c=colors[1215:1715])
   ax.plot(np.arange(1215, 1715), num_valids[1215:1715], linewidth=0.1,
c='k')
   ax.yaxis.get_major_locator().set_params(integer=True)
   ax.set_ylabel("Number of Visible Landmarks")
   ax.set_xlabel("Timestep Number")
   plt.savefig("q4_zoomed.png", bbox_inches='tight')

def q5a():
   k1, k2 = 1215, 1714
   T_op = get_initial_guess(k1, k2)
   T_opt, A = g_batch_est(T_op, k1, k2, iters=10)
```

```
    plot_errs(T_opt, A, k1, k2)

def q5b():
    k1, k2 = 1215, 1714
    kappa = 50

    T_op = get_initial_guess(k1, k1+kappa)
    T_opt, A = g_batch_est(T_op, k1, k1+kappa, iters=10)

    Ts = np.zeros_like(T_opt)
    As = np.zeros(((k2-k1+1)*6, (k2-k1+1)*6))
    Ts[k1] = T_opt[k1]
    var = np.linalg.inv(A).diagonal()
    As[0:6, 0:6] = np.linalg.inv(np.diag(var[0:6]))

    for k in range(k1+1, k2+1):
        print(f"Current timestep: {k}")
        T_op = get_initial_guess(k, k+kappa, T_gt=Ts[k-1])
        T_opt, A = g_batch_est(T_op, k, k+kappa, iters=10)
        Ts[k] = T_opt[k]
        var = np.linalg.inv(A).diagonal()
        As[(k-k1)*6:(k-k1)*6+6, (k-k1)*6:(k-k1)*6+6] =
np.linalg.inv(np.diag(var[0:6]))

    plot_errs(Ts, As, k1, k2)

def q5c():
    k1, k2 = 1215, 1714
    kappa = 10

    T_op = get_initial_guess(k1, k1+kappa)
    T_opt, A = g_batch_est(T_op, k1, k1+kappa, iters=10)

    Ts = np.zeros_like(T_opt)
    As = np.zeros(((k2-k1+1)*6, (k2-k1+1)*6))
    Ts[k1] = T_opt[k1]
    var = np.linalg.inv(A).diagonal()
    As[0:6, 0:6] = np.linalg.inv(np.diag(var[0:6]))
```

Shounak Chakraborty
0474348

```
    for k in range(k1+1, k2+1):
        print(f"Current timestep: {k}")
        T_op = get_initial_guess(k, k+kappa, T_gt=Ts[k-1])
        T_opt, A = g_batch_est(T_op, k, k+kappa, iters=10)
        Ts[k] = T_opt[k]
        var = np.linalg.inv(A).diagonal()
        As[(k-k1)*6:(k-k1)*6+6, (k-k1)*6:(k-k1)*6+6] =
np.linalg.inv(np.diag(var[0:6]))

    plot_errs(Ts, As, k1, k2)

if __name__ == "__main__":
    #q4()
    #q5a()
    #q5b()
    q5c()
```