Perception_PS3

March 12, 2020

1 Task 1

1.1 Task A. Mahalanobis distances

Task 1

$$z = \begin{bmatrix} r \\ \phi \\ id \end{bmatrix} \rightarrow z = \begin{bmatrix} r \\ \phi \end{bmatrix}$$

$$x = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

1A

$$u = \begin{bmatrix} \delta_{rot1} \\ \delta_{trans} \\ \delta_{rot2} \end{bmatrix}$$

We have:

G, H, M, Q, Σ_0.

Mahalanobis distance ~~for~~:

1) Transition:

$$\|g_i(x_{i-1}, u_i) - x_i\|^2_{\Sigma_i} = \left(g_i(x^0_{i-1}, u_i) + G_i^{i-1} \cdot \delta x_{i-1} - x_i\right)^T \cdot V_i \cdot M \cdot V_i^T \cdot \left(g_i(x^0_{i-1}, u_i) + G_i^{i-1} \cdot \delta x_{i-1} - x_i\right) =$$

$$= \left(G_i^{i-1} \cdot \delta x_{i-1} - I \delta x_i - \left(I x_i^0 - g_i(x^0_{i-1}, u_i)\right)\right)^T V_i \cdot M \cdot V_i^T \cdot$$

$$\cdot \left(G_i^{i-1} \delta x_{i-1} - I \delta x_i - \left(I x_i^0 - g_i(x^0_{i-1}, u_i)\right)\right)$$

2) Observations:

$$\|h_k(x_{ik}, m_{jk}) - z_k\|^2_{\Sigma_k} = \left(\underbrace{h_k(x^0_{ik}, m^0_{jk}) +}_{} H_k^{ik} \delta x_{ik} + J_k^{jk} \delta m_{jk} - z_k\right)^T \cdot Q \cdot \left(h_k(x^0_{ik}, m^0_{jk}) + H_k^{ik} \delta x_{ik} + J_k^{jk} \delta m_{jk} - z_k\right) =$$

$$= \left(H_k^{ik} \delta x_{ik} + J_k^{jk} \delta m_{jk} - \left(z_k - h_k(x^0_{ik}, m^0_{jk})\right)\right)^T \cdot Q \cdot \left(H_k^{ik} \delta x_{ik} + J_k^{jk} \delta m_{jk} - \left(z_k - h_k(x^0_{ik}, m^0_{jk})\right)\right)$$

3) Initial condition:

$$\|-x_0\|^2_{\Sigma_0} = x_0^T \cdot \Sigma_0 \cdot x_0$$

$$H_k^{ik} = \left.\frac{\partial h}{\partial x_{ik}}\right|_{(x^0_{ik}, m^0_{jk})} \qquad G_i^{i-1} = \left.\frac{\partial g_i(x_{i-1}, u_i)}{\partial x_{i-1}}\right|_{x^0_{i-1}}$$

$$J_k^{jk} = \left.\frac{\partial h}{\partial m_{jk}}\right|_{(x^0_{ik}, m^0_{jk})}$$

1.2 Task B. Pre-multiplying matrices

1.2.1 The code

[5]:
```
import tools.jacobian as j
import tools.task as t
import numpy as np
from field_map import FieldMap
```

[13]:
```
u = np.array([0, 10, 0])
x = [180, 50, 0]
alphas = np.array([0.05**2, 0.001**2, 0.05**2, 0.01**2])

_, V = j.state_jacobian(x ,u)
print('V:\n', V, '\n')
M = t.get_motion_noise_covariance(u, alphas)

print('M:\n', M, '\n')

print('Pre-multiplying matrice for transition:\n', np.linalg.cholesky(np.linalg.
↪inv(V.dot(M.dot(V.T)))).T, '\n')

Q = np.array([[100, 0], [0, 100]])
print('Pre-multiplying matrice for observation:\n', np.linalg.cholesky(np.
↪linalg.inv(Q)).T, '\n')
```

```
V:
 [[-0.  1.  0.]
 [10.  0.  0.]
 [ 1.  0.  1.]]

M:
 [[1.0e-04 0.0e+00 0.0e+00]
 [0.0e+00 2.5e-01 0.0e+00]
 [0.0e+00 0.0e+00 1.0e-04]]

Pre-multiplying matrice for transition:
 [[  2.          0.          0.        ]
 [  0.         14.14213562 -70.71067812]
 [  0.          0.          70.71067812]]

Pre-multiplying matrice for observation:
 [[0.1 0. ]
 [0.  0.1]]
```

1.2.2 Paper version

1B

For transition:

$\Sigma_i^{-\frac{T}{2}} = (V_i M V_i)^{-\frac{T}{2}}$

For observation.

$\Sigma_k^{-\frac{T}{2}} = Q^{-\frac{T}{2}}$

$$V = \begin{bmatrix} -\delta_{trans} \cdot \sin(\theta + \delta_{rot_1}) & \cos(\theta + \delta_{rot_1}) & 0 \\ \delta_{trans} \cdot \cos(\theta + \delta_{rot_1}) & \sin(\theta + \delta_{rot_1}) & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} \alpha_1 \delta_{rot_1}^2 + \alpha_2 \delta_{trans}^2 & 0 & 0 \\ 0 & \alpha_3 \delta_{trans}^2 + \alpha_4 (\delta_{rot_1}^2 + \delta_{rot_2}^2) & 0 \\ 0 & 0 & \alpha_1 \delta_{rot_1}^2 + \alpha_2 \delta_{trans}^2 \end{bmatrix}$$

Initial conditions:

$$u = \begin{bmatrix} 0 \\ 10 \\ 0 \end{bmatrix}$$

$$x = \begin{bmatrix} 180 \\ 50 \\ 0 \end{bmatrix}$$

Alphas:

$\alpha_1 = 0{,}05^2$

$\alpha_2 = 0{,}001^2$

$\alpha_3 = 0{,}05^2$

$\alpha_4 = 0{,}01^2$

$\Rightarrow$ Transition:

$\Sigma_{t=1}^{-\frac{T}{2}} = (VMV^T)^{-\frac{T}{2}}$

$$V = \begin{bmatrix} 0 & 1 & 0 \\ 10 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}; \; M = \begin{bmatrix} 100\alpha_2 & 0 & 0 \\ 0 & 100\alpha_3 & 0 \\ 0 & 0 & 100\alpha_2 \end{bmatrix} = \begin{bmatrix} 0{,}0001 & 0 & 0 \\ 0 & 0{,}25 & 0 \\ 0 & 0 & 0{,}0001 \end{bmatrix}$$

Calculations step by step.

1) VMV^T

2) $(VMV^T)^{-1}$ (inverse matrix)

3) $\cancel{(VMV)^T}$ $(VMV^T)^{-\frac{1}{2}} \leftarrow \left((VMV^T)^{-1} = L L^T \text{ Cholesky} \right)$

4) $\left((VMV^T)^{-\frac{1}{2}} \right)^T = (VMV^T)^{-\frac{T}{2}} = \Sigma_{t=1}^{-\frac{T}{2}}$

$$\Rightarrow \Sigma_{t=1}^{-\frac{T}{2}} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 14{,}14 & -70{,}71 \\ 0 & 0 & 70{,}71 \end{bmatrix}$$

Observation:

$\Sigma_{k=1}^{-\frac{T}{2}} = Q^{-\frac{T}{2}}$

$$Q = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \Rightarrow \Sigma_{k=1}^{-\frac{T}{2}} = \begin{bmatrix} 0{,}1 & 0 \\ 0 & 0{,}1 \end{bmatrix}$$

~~Q=~~

1.3 Task C. Adjacency matrix A

As a part of SAM filtering I created a method named “update” placed in “sam.py” in “SAM” class.

This method creates an adjacency matrix A.

“run.py” script calls this method and prints the matrix A for t=1.

“run.py” has on option to create the matrix A for any given number of observations. The parameter is called “The maximum number of observations to generate per time step”. The parameter is 2 by default.

1.3.1 Adjacency matrix A for t=1

[[-1.00e+06 0.00e+00 0.00e+00 -0.00e+00 -0.00e+00 -0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00]

[0.00e+00 -1.00e+06 0.00e+00 -0.00e+00 -0.00e+00 -0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00]

[0.00e+00 0.00e+00 -1.00e+06 -0.00e+00 -0.00e+00 -0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00]

[2.00e+00 0.00e+00 0.00e+00 -2.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00]

[0.00e+00 1.41e+01 7.07e+01 0.00e+00 -1.41e+01 7.07e+01 0.00e+00 0.00e+00 0.00e+00 0.00e+00]

[0.00e+00 0.00e+00 7.07e+01 0.00e+00 0.00e+00 -7.07e+01 0.00e+00 0.00e+00 0.00e+00 0.00e+00]

[0.00e+00 0.00e+00 0.00e+00 -1.00e-01 0.00e+00 0.00e+00 1.00e-01 -0.00e+00 0.00e+00 0.00e+00]

[0.00e+00 0.00e+00 0.00e+00 -0.00e+00 -0.00e+00 -5.70e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00]

[0.00e+00 0.00e+00 0.00e+00 -1.00e-01 0.00e+00 0.00e+00 0.00e+00 0.00e+00 1.00e-01 -0.00e+00]

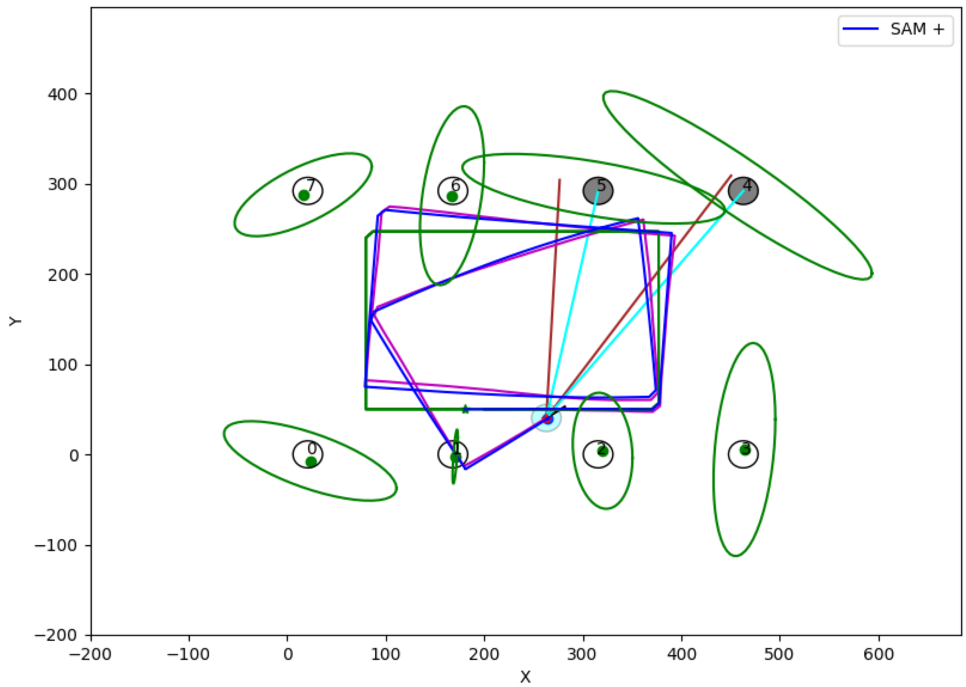
[0.00e+00 0.00e+00 0.00e+00 -0.00e+00 -0.00e+00 -5.70e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00]]

2 Task 2

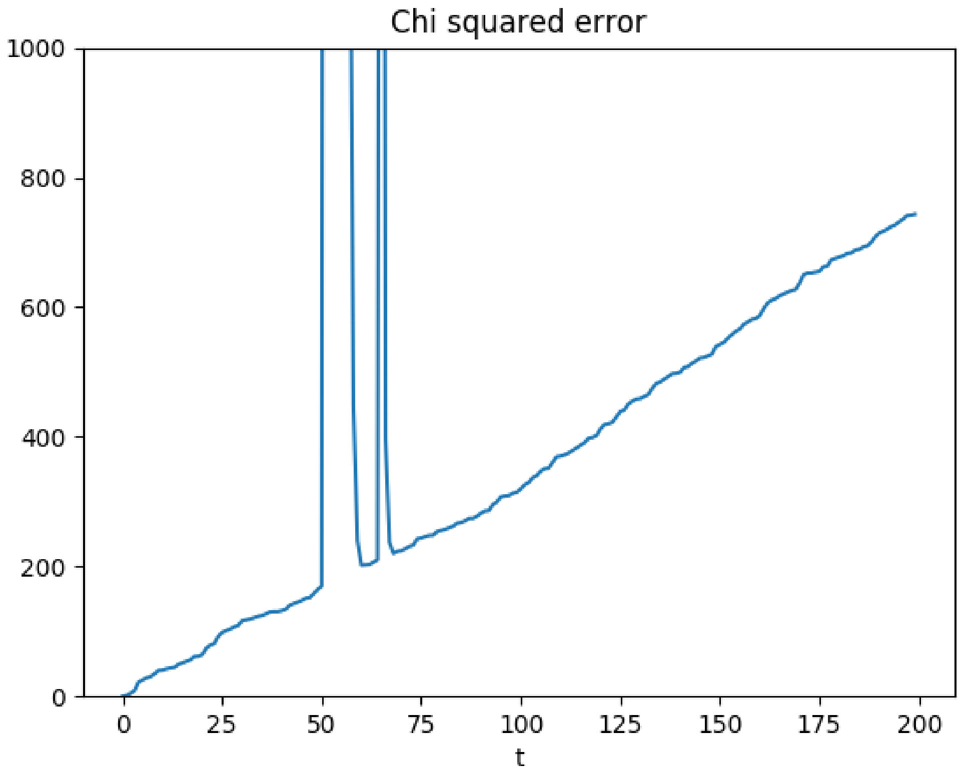
2.1 Task A. SAM is solved

2.2 Task B. Plot of the final map and movie creation (sam.mp4 is created)

2.2.1 Plot of the final map with 3-sigma covariances



2.3 Task C. Calculate and plot the Chi squared error at each instant of time



Some very high errors are shown on the figure above. These errors corespond to high disturbances shown in the video. I tried to do my best, but there is some mistake in wrapping angles. However, despite of high errors you can see a very high level of stability of the robot's trajectory.

```
[ ]:
```