MA01 - PC6

GELMI DE FREITAS SALVO, Guilherme
guilherme.gelmi@ensta-paris.fr

SANTOS COSTA, Rian Radeck
rian.santos@ensta-paris.fr

1 Cadre du filtre de Kalman

Le vecteur d'état considéré est constitué des coordonnées x et y du robot et de l'angle ϕ de référence arbitraire. Les équations du modèle d'état sont

$$\begin{cases} x_{k+1} = x_k + (V + \Delta V_k)T\cos\phi_k \\ y_{k+1} = y_k + (V + \Delta V_k)T\sin\phi_k \\ \phi_{k+1} = \phi_k + T(\omega + \Delta\omega_k) \end{cases}$$

où ΔV_k et $\Delta\omega_k$ sont des réalisations de deux bruits blancs respectivement sur sur la vitesse V et sur la vitesse de rotation ω_k. On définit donc, $\Delta V_k \sim \mathcal{N}(0, \sigma_v^2)$ et $\Delta\omega_k \sim \mathcal{N}(0, \sigma_\omega^2)$. L' équation de mesure est par ailleurs :

$$z_k = \sqrt{(x_k - x_b)^2 + (y_k - y_b)^2} + w_k$$

où x_b, y_b sont les coordonnées d'un point fixe arbitraire et $w_k \sim \mathcal{N}(0, \sigma_w^2)$.

2 Linéarisation

Pour effectuer la linéarisation locale du Filtre de Kalman Étendu (EKF), il est nécessaire de modéliser les équations du modèle d'état de la manière suivante :

$$\begin{cases} x_{k+1} &= f(x_k) + b_k^{(1)} \\ z_k &= h(x_k) + b_k^{(2)} \end{cases}$$

où

- f et h sont des fonctions différentiables connues.
- $b_k^{(1)}$ et $b_k^{(2)}$ sont des bruits mutuellement indépendants qui suivent respectivement des distributions normales $\mathcal{N}(0, \sigma_1^2)$ et $\mathcal{N}(0, \sigma_2^2)$, respectivement.

Pour modéliser les trois équations qui décrivent l'état du robot, on définit $u = \begin{bmatrix} V \\ \omega \end{bmatrix}$ et $X_k = \begin{bmatrix} x_k \\ y_k \\ \phi_k \end{bmatrix}$

et donc, on peut dire que f est exprimée par $f(X_k, u)$ parce que

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \phi_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + VT\cos\phi_k \\ y_k + VT\sin\phi_k \\ \phi_k + T\omega \end{bmatrix} + \begin{bmatrix} \Delta V_k T\cos\phi_k \\ \Delta V_k T\sin\phi_k \\ \Delta\omega_k T \end{bmatrix} \Leftrightarrow X_{k+1} = f(X_k, u) + b_k^{(1)}$$

De manière similaire, on procède de la même façon pour $h(X_k) = \sqrt{(x_k - x_b)^2 + (y_k - y_b)^2}$ et on note $b_k^{(2)} = w_k$. Cela est déjà suffisant pour démontrer l'énoncé de l'exercice 1.1.

Pour finaliser la approximation de la probabilité de transition, on note :

$$\Phi_k = \frac{\delta f(X)}{\delta X}\Big|_{X=\hat{X}_k} = \begin{bmatrix} 1 & 0 & -VT\sin\hat{\phi}_k \\ 0 & 1 & VT\cos\hat{\phi}_k \\ 0 & 0 & 1 \end{bmatrix}$$

$$\Gamma = \frac{\delta f(U)}{\delta U}\Big|_{U=u} = \begin{bmatrix} T\cos\hat{\phi}_k & 0 \\ T\sin\hat{\phi}_k & 0 \\ 0 & T \end{bmatrix}$$

finalement, en notant P_k la covariance à l'instant k du vecteur $\hat{X}_k$, la prédiction de cette covariance sera notée par

$$P_{k+1|k} = \Phi_k P_k \Phi_k^T + \Gamma \text{Cov}(\Delta V, \Delta\omega)\Gamma^T$$

où $\text{Cov}(\Delta V, \Delta\omega)$ est la matrice de covariance associée au vecteur $\begin{bmatrix}\Delta V & \Delta\omega\end{bmatrix}^T$.

Maintenant, on définit H_k pour l'approximation de la vraisemblance.

$$H_k = \frac{\delta h(X)}{\delta X}\Big|_{X=\hat{X}_{k+1|k}} = \begin{bmatrix} \frac{\hat{x}_{k+1|k}-x_b}{\sqrt{(\hat{x}_{k+1|k}-x_b)^2+(\hat{y}_{k+1|k}-y_b)^2}} \\ \frac{\hat{y}_{k+1|k}-y_b}{\sqrt{(\hat{x}_{k+1|k}-x_b)^2+(\hat{y}_{k+1|k}-y_b)^2}} \\ 0 \end{bmatrix}$$

Avec cela, on dispose de tout ce qui est nécessaire pour exécuter l'algorithme EKF.

3 Le algorithme du filtre de Kalman étendu

3.1 Initialisation

On prend une estimation $\hat{X}_0$ arbitraire parmi les valeurs possibles de X, et une matrice de covariance arbitrairement grande (par exemple $P_0 = \alpha I$ avec α arbitrairement grand).

3.2 Itération

Pour $k \geq 0$

— Prédiction

$$\begin{cases} \hat{X}_{k+1|k} = \hat{X}_k + \begin{bmatrix} VT\cos\hat{\phi}_k & VT\sin\hat{\phi}_k & T\omega \end{bmatrix}^T \\ P_{k+1|k} = \Phi_k P_k \Phi_k^T + \Gamma \text{Cov}(\Delta V, \Delta\omega)\Gamma^T \end{cases}$$

— Correction

Calcul du gain de Kalman :

$$K_{k+1} = P_{k+1|k} H_{k+1}^T (H_{k+1} P_{k+1|k} H_{k+1}^T + w_{k+1})^{-1}$$

Mise à jour :

$$\begin{cases} \hat{X}_{k+1} = \hat{X}_{k+1|k} + K_{k+1}(z_{k+1} - H_k \hat{X}_{k+1|k}) \\ P_{k+1} = (I - K_{k+1} H_{k+1z}) P_{k+1|k} \end{cases}$$

4 Mise en œuvre

On a mis en œuvre l'algorithme sur python :

```python
# Paramètres donnés
T = 0.01
sigma_v = 0.1
sigma_omega = 0.01
sigma_w = 0.2
V = 3.0  # m/s
omega = 2 * np.pi / 3  # rad/s
x_b, y_b = 2, 5  # Coordonnées du point fixe
x0 = np.array([[0, 0, 0]]).T  # État initial Estimated
P0 = 10 * np.eye(3)  # Covariance initiale

```

```
Cov_u = np.diag([sigma_v**2, sigma_omega**2]) # Matrice de covariance du bruit de processus
W_k = sigma_w**2 # Matrice de covariance du bruit de mesure

def compute_Phi_k(V, T, phi_hat):
    return np.array([
        [1, 0, -V * T * np.sin(phi_hat)],
        [0, 1,  V * T * np.cos(phi_hat)],
        [0, 0, 1]
    ])

def compute_Gamma(T, phi_hat):
    return np.array([
        [T * np.cos(phi_hat), 0],
        [T * np.sin(phi_hat), 0],
        [0, T]
    ])


def compute_H_k(x_hat, y_hat):
    distance = np.sqrt((x_hat - x_b)**2 + (y_hat - y_b)**2)
    return np.array([
        [(x_hat - x_b) / distance],
        [(y_hat - y_b) / distance],
        [0]
    ]).T

X_hat = [x0]
P = [P0]

def step(z):
    def prediction():
        x_hat_k, y_hat_k, omega_hat_k = X_hat[-1][0][0], X_hat[-1][1][0], X_hat[-1][2][0]
        X_hat_pred = np.array([
            [x_hat_k + V * T * np.cos(omega_hat_k)],
            [y_hat_k + V * T * np.sin(omega_hat_k)],
            [omega_hat_k + T * omega]
        ])
        Phi_k = compute_Phi_k(V, T, omega_hat_k)
        Gamma = compute_Gamma(T, omega_hat_k)
        P_pred = Phi_k @ P[-1] @ Phi_k.T + Gamma @ Cov_u @ Gamma.T

        return X_hat_pred, P_pred
    def correction(z):
        X_hat_pred, P_pred = prediction()
        x_hat_pred, y_hat_pred, omega_hat_pred = X_hat_pred[0][0], X_hat_pred[1][0], X_hat_pred[2][0]
        # Kalman gain
        H_k = compute_H_k(x_hat_pred, y_hat_pred)
        S_k = H_k @ P_pred @ H_k.T + W_k
        K_k = P_pred @ H_k.T @ np.linalg.inv(S_k)

        # Update
        X_hat_k1 = X_hat_pred + K_k @ (z - H_k @ X_hat_pred)
        P_k1 = (np.eye(3) - K_k @ H_k) @ P_pred

        X_hat.append(X_hat_k1)
        P.append(P_k1)

    correction(z)
```

```
70
71 z_meas = np.loadtxt('./Kalmannonlin.txt')
72 for z in z_meas:
73     step(z)
```

L'implémentation est très simple et suit les itérations présentées dans la section 3.2. Pour mieux comprendre le code, nous pouvons établir les relations suivantes :

— $\hat{X}_{k+1|k}$ est appelé `X_hat_pred`

— $P_{k+1|k}$ est appelé `P_pred`

— $\hat{X}_{k+1}$ est appelé `X_hat_k1`

— P_{k+1} est appelé `P_k1`

— `np.eye(n)` crée une matrice identité de taille n par n

— L'opérateur `@` correspond à une multiplication de matrices

— L'index -1 d'une liste (`A[-1]`) accède au dernier élément de cette liste

— L'attribut `.T` est la matrice transposé. (X^T c'est `X.T` dans le code)

5 Résultats du algorithme

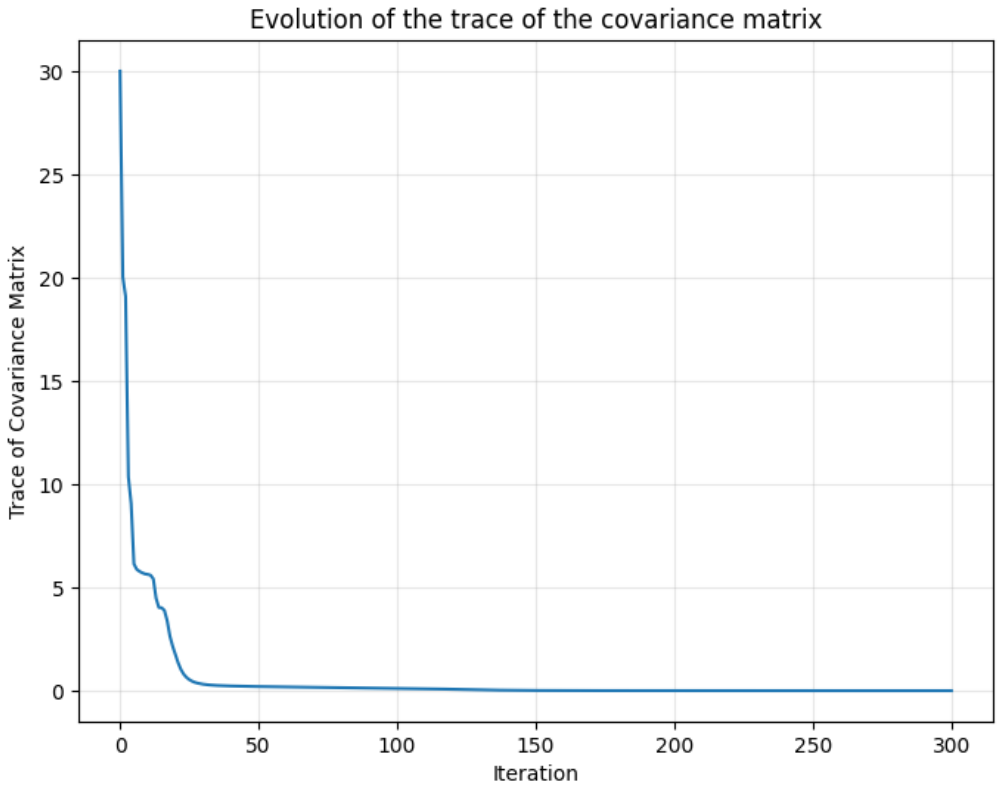

FIGURE 1 – Trace de la matrice de Covariance P_k de l'algorithme

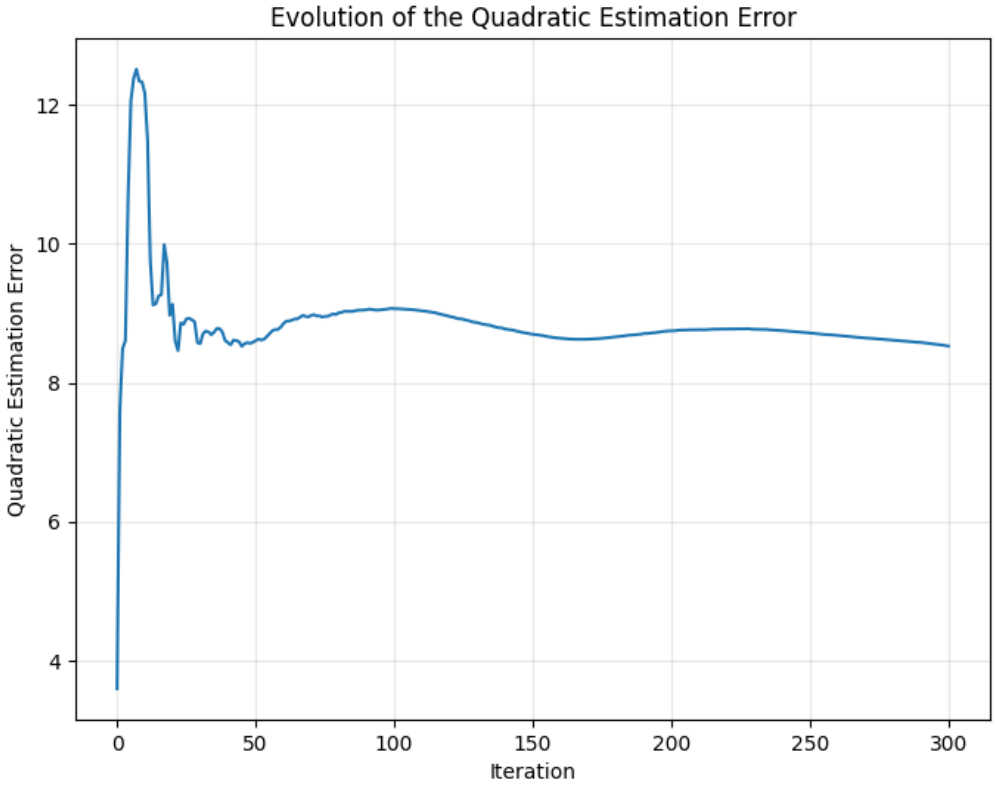

FIGURE 2 – L'erreur quadratique d'estimation $e_k = ||\hat{X}_k - X_k^v||$ où X_k^v est la vraie valeur de l'état fournie dans le fichier etatvrai.txt