



IP PARIS

INSTITUT POLYTECHNIQUE DE PARIS  
ENSTA PARIS

---

# CSC\_5RO12\_TA, Navegation Robotique

TP2, Extented Kalman Filter

---

by

Guilherme NUNES TROFINO

supervised by  
David FILLIAT  
Nicolas MERLINGE

**Confidentiality Notice**  
Non-confidential and publishable report

ROBOTIQUE  
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET COMMUNICATION

---

Paris, FR  
19 octobre 2024

# Table des matières

---

<b>1 Question 1</b>	<b>2</b>
1.1 Structure du Code . . . . .	2
1.2 Execution Initiale . . . . .	3
1.2.1 Utilization . . . . .	3
<b>2 Question 2</b>	<b>4</b>
2.1 EKF . . . . .	4
2.1.1 motion_model_prediction() . . . . .	4
2.1.2 observation_model_prediction() . . . . .	4
2.1.3 F() . . . . .	5
2.1.4 G() . . . . .	5
2.1.5 H() . . . . .	6
<b>3 Question 3</b>	<b>7</b>
<b>4 Question 4</b>	<b>8</b>
<b>5 Question 5</b>	<b>9</b>
<b>6 Question 6</b>	<b>10</b>
<b>7 Question 7</b>	<b>11</b>
<b>8 Question 8</b>	<b>12</b>
<b>9 Question 9</b>	<b>13</b>

## 1. Question 1

**Remarque.** Des modifications ont été faites par rapport à l'algorithme original, notamment avec la création de différentes classes et la définition d'une fonction `main`.

### 1.1. Structure du Code

L'algorithme utilisé dans ce projet suit la structure suivante :

1. `EKF` : Extended Kalman Filter sous forme de dataclass, qui contient les méthodes suivantes :
  - (a) `motion_model_prediction()` ;
  - (b) `observation_model_prediction()` ;
  - (c) `F()` ;
  - (d) `G()` ;
  - (e) `H()` ;
2. `Simulation` : classe pour la création du environnement de simulation du robot ;
  - (a) `get_observation()` : retourne une mesure bruitée d'une amère aléatoire ;
  - (b) `get_odometry()` : retourne une mesure bruitée de l'odométrie et de la commande du robot ;
  - (c) `get_robot_control()` : retourne la véritable commande du robot ;
  - (d) `simulate_world()` : simule le système à l'instant  $k$  ;
3. `Utils` : classe pour stocker des fonctions d'assistance pour l'exécution de l'algorithme ;
  - (a) `plot_covariance_ellipse()` : affiche la matrice de covariance de l'estimation sous forme d'ellipse ;
  - (b) `compute_motion()` : calcule le mouvement du robot selon son équation de mouvement ;
  - (c) `convert_angle()` : rappelle un angle entre  $[-\pi, +\pi]$  ;

Les méthodes de `EKF` seront précisées dans la Question 2.

**Remarque.** Des commentaires et de la documentation ont été ajoutés à l'algorithme pour faciliter sa compréhension et ne seront pas répétés ici.

**Remarque.** Entre chaque exécution, seule une variable a été variée tandis que les autres sont restées inchangées, garantissant ainsi que l'analyse se concentre uniquement sur la variable en question.

**Remarque.** L'application de la fonction `Utils.convert_angle()` a été appliquée à tous les angles calculés dans l'algorithme pour garantir que les résultats affichés sur le graphique soient contenus dans  $[-\pi, +\pi]$ .

**Remarque.** Un vecteur  $\tilde{\mathbf{a}}$  contient des données bruitées.

**Remarque.** Un vecteur  $\hat{\mathbf{a}}$  est une prédition.

## 1.2. Execution Initiale

Après l'implementation des fonctions et variables qui manque au code source, quelques modifications sur le graphique ont été fait et le résultat initiale, utilisé comme benchmark, est donnée par l'image ci-dessous :

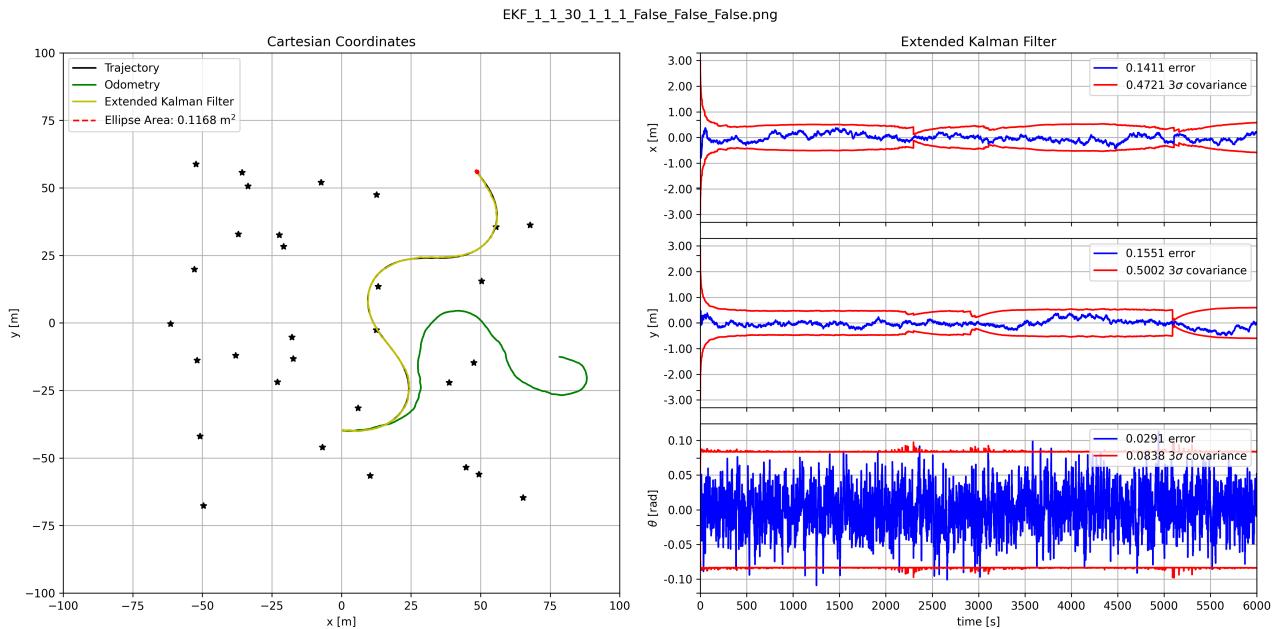


FIGURE 1.1 : Résultats Initiales

**Remarque.** Dans ce projet, chaque image aura sur son titre les informations sur l'exécution : EKF\_i\_j\_k\_l\_m\_n\_o\_p\_q.png où :

1. j : dt\_measurement : intervalle entre deux mesures consécutives en secondes ;
2. i : dt\_prediction : intervalle entre deux prédictions consécutives en secondes ;
3. k : n\_landmarks : nombre de références sur la simulation ;
4. l : P\_constant : constant de la matrice de covariance d'état  $P_k$  ;
5. m : Q\_constant : constant de la matrice de covariance du bruit du processus  $Q_k$  ;
6. n : R\_constant : constant de la matrice de covariance du bruit de mesure  $R_k$  ;
7. o : black\_out : une manque de mesures entre 2500 et 3000 secondes ? ;
8. p : range\_only : seulement les mesures de distances disponibles ? ;
9. q : angle\_only : seulement les mesures d'angles disponibles ? ;

### 1.2.1. Utilization

Pour répondre aux questions, l'algorithme suivant a été utilisé :

```

1 def execution(...) -> None:
2     ...
3
4     plt.suptitle(file_name)
5     if save_result: plt.savefig(file_path, dpi=300)
6     if show_result: plt.show()
7
8 def main():
9     arr = [...]
10
11    for var in arr:
12        execution(var, show_result=False, save_result=True)
13
14 if __name__ == "__main__":
15    main()

```

## 2. Question 2

### 2.1. EKF

Pour calculer l'Extended Kalman Filter, une dataclass a été utilisée, car la classe ne contient que des méthodes, qui sont expliquées ci-dessous.

#### 2.1.1. motion\_model\_prediction()

**Définition 2.1.** La prédiction du modèle de mouvement est donnée par l'équation suivante :

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1}, \tilde{\mathbf{u}}_k) = \begin{bmatrix} x_{k-1} + (\tilde{v}_k^x \cos(\theta_{k-1}) - \tilde{v}_k^y \sin(\theta_{k-1}) \cdot \Delta t) \\ y_{k-1} + (\tilde{v}_k^x \sin(\theta_{k-1}) + \tilde{v}_k^y \cos(\theta_{k-1}) \cdot \Delta t) \\ \theta_{k-1} + \tilde{\omega}_k \Delta t \end{bmatrix} \quad (2.1)$$

Où :

1. **Robot State** :  $\mathbf{x}_k = [x_k \ y_k \ \theta_k]^\top$  à l'instant  $k$ , relative à l'origine du plan cartésien.
2. **Noised Odometry** :  $\tilde{\mathbf{u}}_k = [\tilde{v}_k^x \ \tilde{v}_k^y \ \tilde{\omega}_k]^\top \sim \mathcal{N}(\mathbf{u}_k, \mathbf{Q}_k)$  à l'instant  $k$ , relative au robot :
  - (a) **Robot Control** :  $\mathbf{u}_k = [v_k^x \ v_k^y \ \omega_k]^\top$  à l'instant  $k$ , relative au robot.
  - (b) **Process Noise Covariance** :  $\mathbf{Q}_k$  covariance du bruit Gaussian du processus.

La fonction `motion_model_prediction()` implémente la prédiction du modèle de mouvement :

```

1 def motion_model_prediction(
2     x: np.ndarray[float], u: np.ndarray[float], dt: float
3 ) -> np.ndarray[float]:
4     """
5     Returns motion model prediction as np.ndarray[float] of system state x at instant k.
6
7     Args:
8         x (np.ndarray[float]): system state at instant k-1.
9         u (np.ndarray[float]): control input, or odometry measurement, at instant k.
10        dt (float): simulation time step.
11    """
12    x_k, y_k, theta_k = x[0, 0], x[1, 0], x[2, 0]
13    vx_k, vy_k, w_k = u[0, 0], u[1, 0], u[2, 0]
14
15    x_prediction = np.array([
16        [x_k + (vx_k * np.cos(theta_k) - vy_k * np.sin(theta_k)) * dt],
17        [y_k + (vx_k * np.sin(theta_k) + vy_k * np.cos(theta_k)) * dt],
18        [theta_k + w_k * dt],
19    ])
20    x_prediction[2, 0] = Utils.convert_angle(x_prediction[2, 0])
21
22    return x_prediction

```

Listing 1 : `motion_model_prediction()`

**Remarque.** Ça definition correspond à la fonction `Utils.compute_motion()`.

#### 2.1.2. observation\_model\_prediction()

**Définition 2.2.** La prédiction du modèle d'observation est donnée par l'équation suivante :

$$\hat{\mathbf{y}}_k = h(\hat{\mathbf{x}}_k) = \begin{bmatrix} \sqrt{(x_k^p - x_k)^2 + (y_k^p - y_k)^2} \\ \arctan\left(\frac{y_k^p - y_k}{x_k^p - x_k}\right) - \theta_k \end{bmatrix} \quad (2.2)$$

La fonction `observation_model_prediction()` implémente la prédiction du modèle d'observation :

```

1 def observation_model_prediction(
2     x: np.ndarray[float], i: int, landmarks: np.ndarray[float]
3 ) -> np.ndarray[float]:
4     """
5     Returns observation model prediction as np.ndarray[float] of system command y at instant k.
6
7     Args:
8         x (np.ndarray[float]): system state at instant k-1.
9         i (int): observed landmark index.
10        landmarks (np.ndarray[float]): coordinates x and y of all landmarks.
11    """
12    x_k, y_k, theta_k = x[0, 0], x[1, 0], x[2, 0]
13    x_i, y_i = landmarks[0, i], landmarks[1, i]
14
15    h = np.array([
16        [np.sqrt((x_i - x_k)**2 + (y_i - y_k)**2)],
17        [np.arctan2((y_i - y_k), (x_i - x_k)) - theta_k],
18    ])
19    h[1, 0] = Utils.convert_angle(h[1, 0])
20
21    return h

```

Listing 2 : `observation_model_prediction()`

### 2.1.3. F()

**Définition 2.3.** Le jacobien du modèle de prédition de mouvement, dans le cadre d'un filtre de Kalman étendu (EKF), est utilisé pour linéariser la fonction de mouvement autour de l'état actuel. Ce jacobien est noté  $\mathbf{F}_k$  et est calculé à partir de la dérivée partielle de la fonction de mouvement par rapport à l'état  $\mathbf{x}_k$ .

L'équation générale du jacobien pour la prédition de mouvement est donnée par :

$$\mathbf{F}_k = \frac{\partial \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \tilde{\mathbf{u}}_k)}{\partial \hat{\mathbf{x}}_k} = \begin{bmatrix} 1 & 0 & (-v_k^x \cdot \sin(\theta_k) - v_k^y \cdot \cos(\theta_k)) \cdot \Delta t \\ 0 & 1 & (+v_k^x \cdot \cos(\theta_k) - v_k^y \cdot \sin(\theta_k)) \cdot \Delta t \\ 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

La fonction F() implémente le jacobian du modèle de prédition de mouvement :

```

1 def F(x: np.ndarray[float], u: np.ndarray[float], dt: float) -> np.ndarray[float]:
2     """
3     Return motion model state jacobian matrix F(x) as np.ndarray[float].
4
5     Note: jacobian with respect of system state.
6
7     Args:
8         x (np.ndarray[float]): system state at instant k-1.
9         u (np.ndarray[float]): control input, or odometry measurement, at instant k.
10        dt (float): simulation time step.
11    """
12    _, _, theta_k = x[0, 0], x[1, 0], x[2, 0]
13    vx_k, vy_k, _ = u[0, 0], u[1, 0], u[2, 0]
14
15    F = np.array([
16        [1, 0, (-vx_k * np.sin(theta_k) - vy_k * np.cos(theta_k)) * dt],
17        [0, 1, (+vx_k * np.cos(theta_k) - vy_k * np.sin(theta_k)) * dt],
18        [0, 0, 1]
19    ])
20
21    return F

```

Listing 3 : `F()`

### 2.1.4. G()

**Définition 2.4.** Le jacobien du modèle de prédition de contrôle, utilisé dans un filtre de Kalman étendu (EKF), est également une matrice qui exprime la relation entre les variations des commandes de contrôle  $\mathbf{u}_k$  et les changements dans l'état  $\mathbf{x}_k$  du système.

L'équation générale pour le jacobien du modèle de prédition de contrôle  $\mathbf{G}_k$  est donnée par :

$$\mathbf{G}_k = \frac{\partial \mathbf{f}(\tilde{\mathbf{x}}_{k-1}, \tilde{\mathbf{u}}_k)}{\partial \mathbf{w}} = \frac{\partial \mathbf{f}(\tilde{\mathbf{x}}_{k-1}, \tilde{\mathbf{u}}_k)}{\partial \tilde{\mathbf{u}}} = \begin{bmatrix} +\cos(\theta_k) \cdot \Delta t & -\sin(\theta_k) \cdot \Delta t & 0 \\ +\sin(\theta_k) \cdot \Delta t & +\cos(\theta_k) \cdot \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix} \quad (2.4)$$

La fonction  $\mathbf{G}()$  implémente le jacobian du modèle de prédition de contrôle :

```

1 def G(x: np.ndarray[float], u: np.ndarray[float], dt: float) -> np.ndarray[float]:
2     """
3         Return motion model control jacobian matrix G(x) as np.ndarray[float].
4
5         Note: jacobian with respect of noised odometry.
6
7         Args:
8             x (np.ndarray[float]): system state at instant k-1.
9             u (np.ndarray[float]): control input, or odometry measurement, at instant k.
10            dt (float): simulation time step.
11        """
12    _, _, theta_k = x[0, 0], x[1, 0], x[2, 0]
13
14    G = np.array([
15        [np.cos(theta_k) * dt, -np.sin(theta_k) * dt, 0],
16        [np.sin(theta_k) * dt, +np.cos(theta_k) * dt, 0],
17        [0, 0, dt]
18    ])
19
20    return G

```

Listing 4 :  $\mathbf{G}()$

## 2.1.5. $\mathbf{H}()$

**Définition 2.5.** Le jacobien du modèle de prédition de l'observation, dans le cadre d'un filtre de Kalman étendu (EKF), est une matrice qui exprime la relation entre l'état du système  $\mathbf{x}_k$  et les observations  $\mathbf{y}_k$  à l'instant  $k$ . Ce jacobien est crucial pour linéariser le modèle d'observation dans un EKF.

L'équation générale pour le jacobien du modèle d'observation  $\mathbf{H}_k$  est donnée par :

$$\mathbf{H}_k = \frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \begin{bmatrix} -\frac{x_k^p - x_k}{\sqrt{(x_k^p - x_k)^2 + (y_k^p - y_k)^2}} & -\frac{y_k^p - y_k}{\sqrt{(x_k^p - x_k)^2 + (y_k^p - y_k)^2}} & +0 \\ \frac{y_k^p - y_k}{(x_k^p - x_k)^2 + (y_k^p - y_k)^2} & -\frac{x_k^p - x_k}{(x_k^p - x_k)^2 + (y_k^p - y_k)^2} & -1 \end{bmatrix} \quad (2.5)$$

La fonction  $\mathbf{H}()$  implémente le jacobian du modèle de prédition de l'observattion :

```

1 def H(x: np.ndarray[float], i: int, landmarks: np.ndarray[float]) -> np.ndarray[float]:
2     """
3         Return observation model jacobian matrix H(x) as np.ndarray[float].
4
5         Args:
6             x (np.ndarray[float]): system state at instant k-1.
7             i (int): observed landmark index.
8             landmarks (np.ndarray[float]): coordinates x and y of all landmarks.
9        """
10    x_k, y_k = x[0, 0], x[1, 0]
11    x_p, y_p = landmarks[0, i], landmarks[1, i]
12
13    delta_x = x_p - x_k
14    delta_y = y_p - y_k
15    distance = np.sqrt(delta_x**2 + delta_y**2)
16
17    H = np.array([
18        [-delta_x/distance, -delta_y/distance, 0],
19        [+delta_y/distance**2, -delta_x/distance**2, -1]
20    ])
21
22    return H

```

Listing 5 :  $\mathbf{H}()$

### 3. Question 3

Ci-dessous, sont présentées quelques interactions résultant de la variation de la fréquence de mesure :

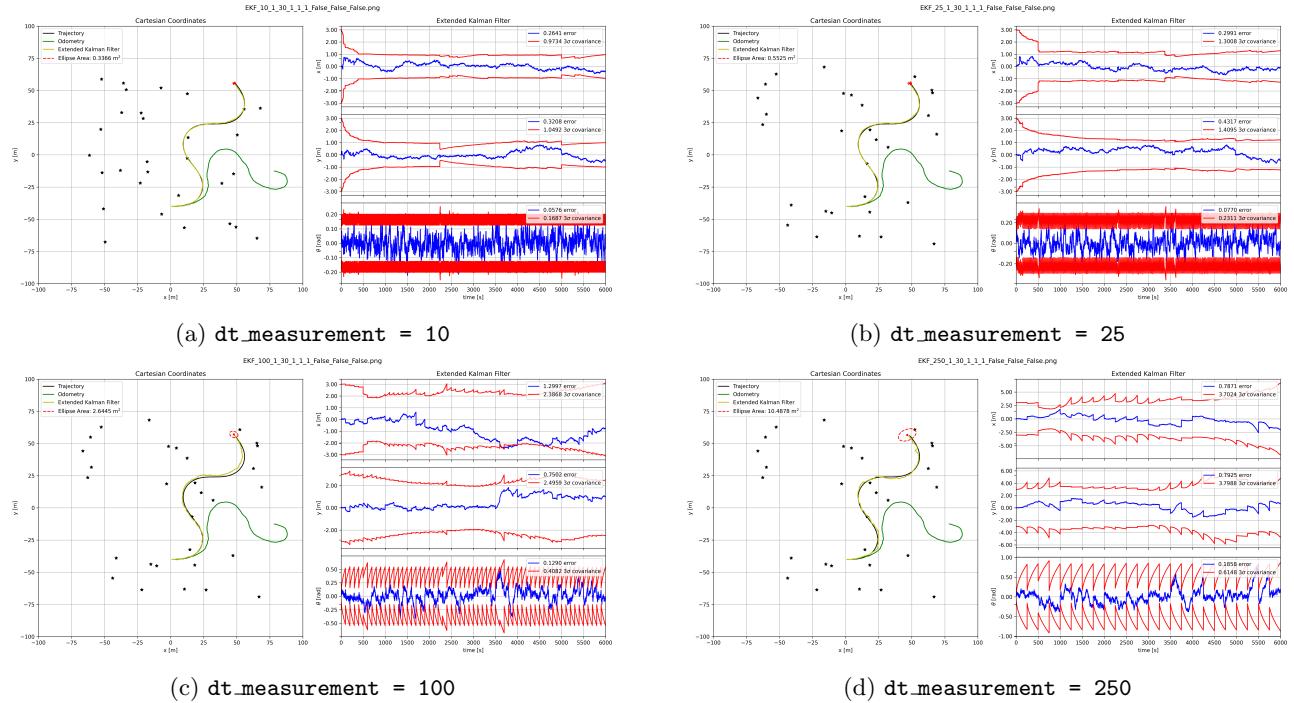


FIGURE 3.1 : Variation de la fréquence de mesure  $dt\_measurement$

Il est à noter qu'une augmentation de l'intervalle de mesure entraîne une détérioration significative de la trajectoire, ce qui indique une perte de précision dans le calcul du filtre de Kalman. Ce phénomène se manifeste par plusieurs indicateurs :

1. l'augmentation de l'erreur sur les états du système ;
2. l'accroissement de l'écart-type ;
3. l'élargissement de l'ellipse de covariance ;

Ce phénomène peut être expliqué par le fait qu'un élargissement de l'intervalle de mesure entraîne un nombre accru d'itérations entre deux mesures, ce qui conduit à une accumulation d'erreurs entre les états du filtre de Kalman et à un résultat dégradé.

**Remarque.** Malgré cette dégradation, il convient de souligner que le filtre de Kalman continue de fournir des résultats acceptables, même lorsque l'intervalle de mesure a été multiplié par 250 pour atteindre ce niveau de dégradation.

## 4. Question 4

Ci-dessous, quelques interactions ont été réalisées en faisant varier le bruit dynamique du filtre :

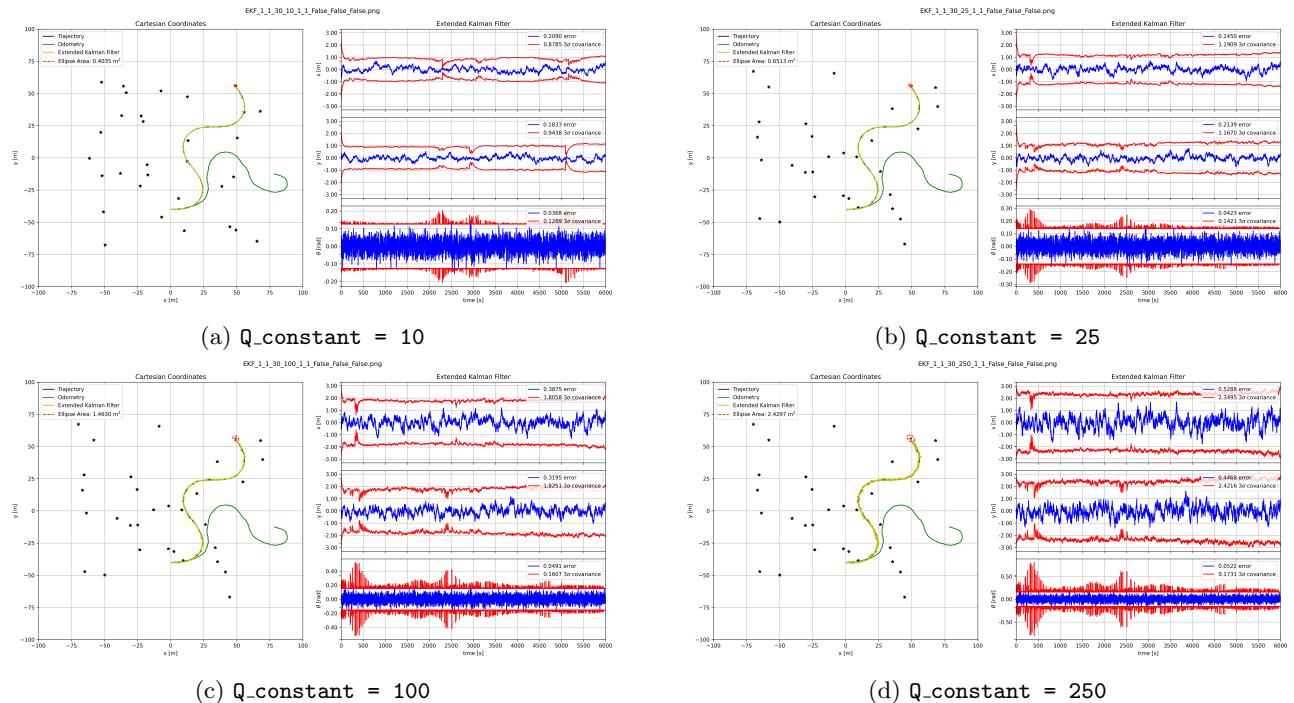


FIGURE 4.1 : Variance du bruit dynamique Q.constant

Il est à noter qu'à mesure que le bruit dynamique augmente, l'instabilité des résultats s'accroît. Cela indique que l'erreur de calcul du filtre de Kalman n'augmente pas de manière significative, mais que les résultats deviennent plus bruyants entre chaque itération, entraînant des courbes moins lisses.

**Remarque.** Le parcours effectué par le robot demeure satisfaisant, bien que l'ellipse de covariance augmente plus rapidement qu'auparavant.

## 5. Question 5

Ci-dessous quelques interactions ont été faits en variant le bruit de mesure du filtre :

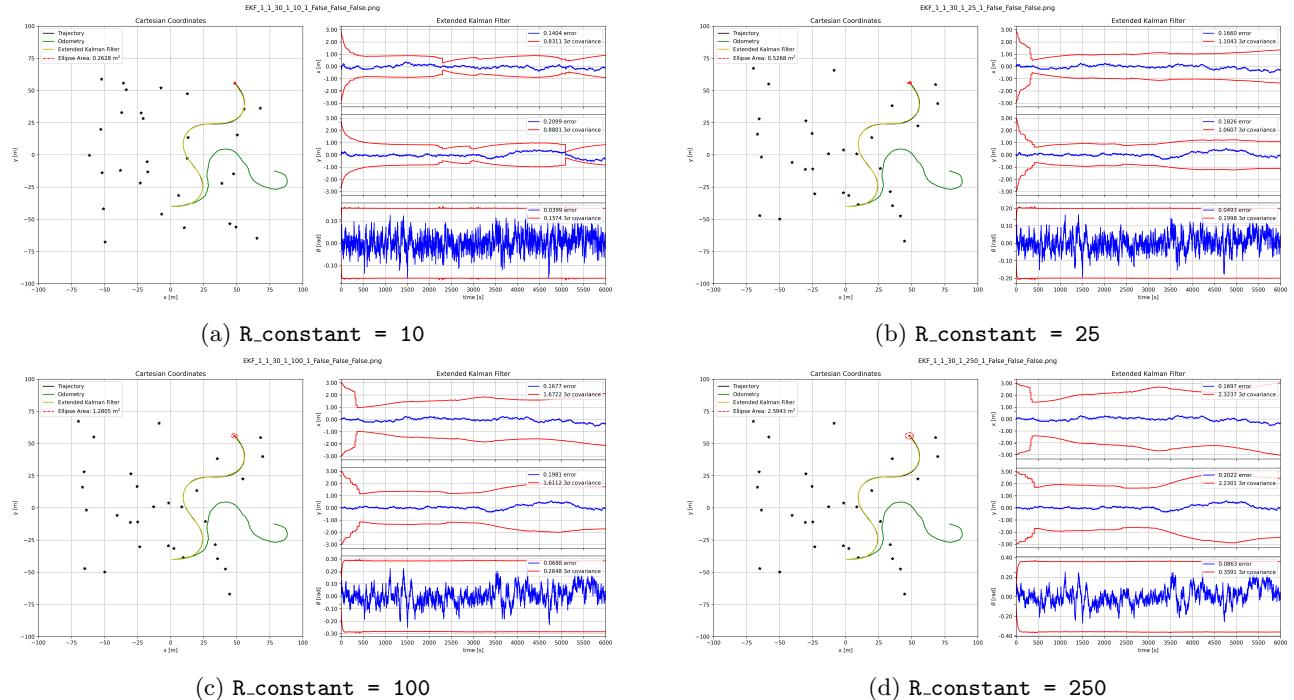


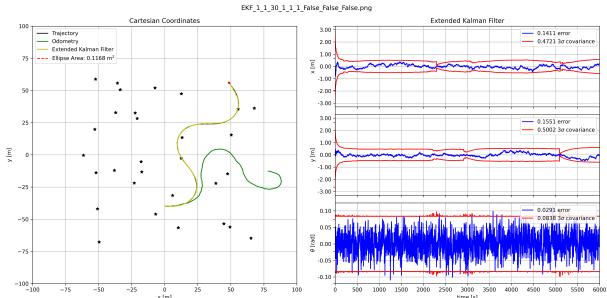
FIGURE 5.1 : Variation du bruit de mesure  $R_{\text{constant}}$

Il est à noter qu'à mesure que le bruit de mesure augmente, la variance des mesures s'accroît, bien que l'erreur ne montre pas de changement significatif. Il est également observable que la covariance de la position augmente, ce qui se traduit par un agrandissement de l'ellipse, tandis que la trajectoire demeure relativement inaltérée.

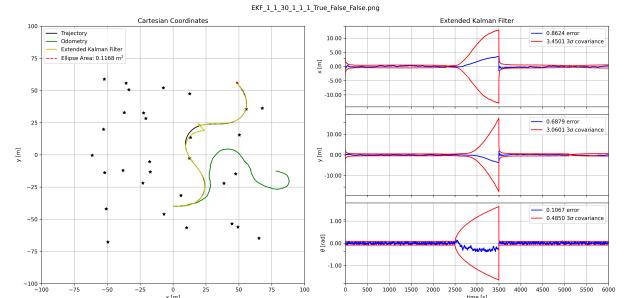
**Remarque.** Cela indique que lorsque le bruit des mesures augmente, le filtre de Kalman accorde moins de confiance aux mesures et fait davantage confiance aux estimations, ce qui explique le maintien de sa trajectoire pratiquement inchangée, malgré l'erreur pratiquement constante, tandis que sa covariance augmente.

## 6. Question 6

Ci-dessous, un trou de mesures simulé entre 2500 et 3500 secondes :



(a) `black_out = False`



(b) `black_out = True`

FIGURE 6.1 : Simulation d'un trou `black_out`

Il est évident que durant la coupure de mesures entre 2500 et 3000 secondes, l'erreur et la covariance augmentent, car aucune correction n'est effectuée par le filtre de Kalman. En revanche, une fois les mesures rétablies, le filtre de Kalman parvient à corriger le problème et à revenir à un état relativement proche de celui observé dans la version complète en quelques itérations.

**Remarque.** Cela démontre que le filtre de Kalman est robuste face à la perte d'informations, même pendant de longues périodes, se corrigeant rapidement après quelques itérations.

## 7. Question 7

Ci-dessous, quelques interactions ont été réalisées en faisant varier le nombre d'amers sur la carte :

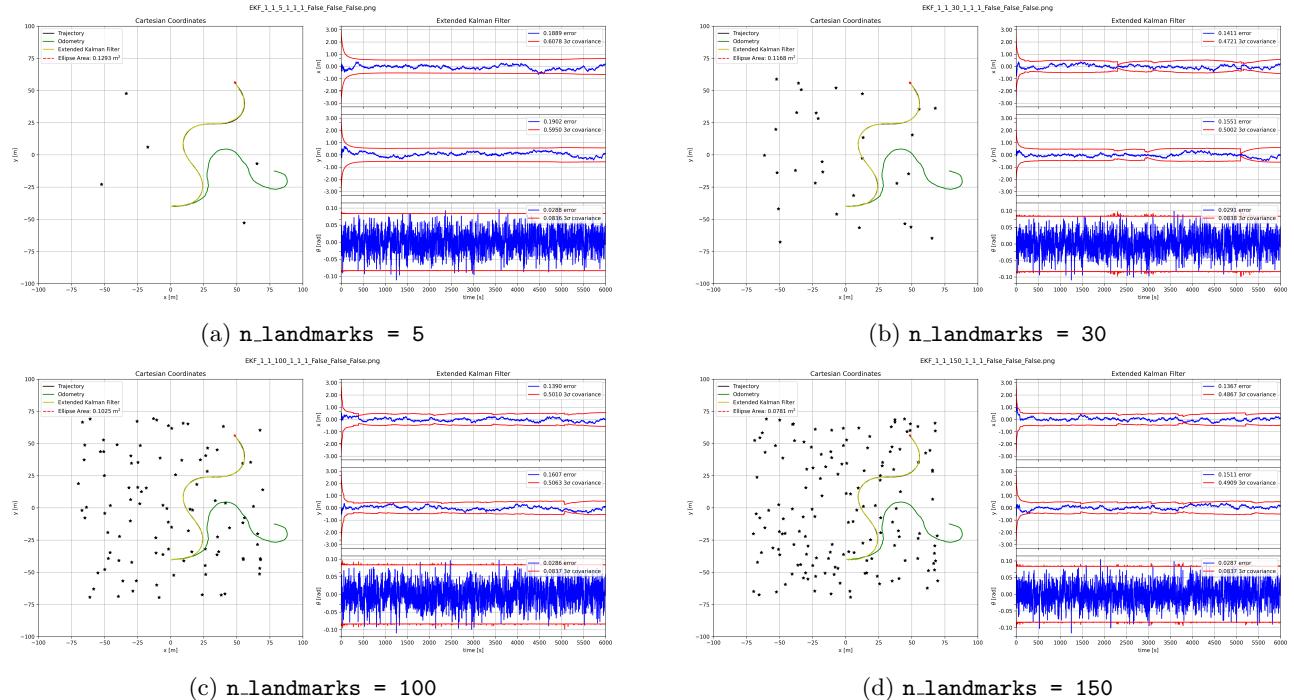


FIGURE 7.1 : Variation du nombre d'amers  $n\_landmarks$

On remarque qu'en augmentant le nombre de références dans l'environnement de simulation, le résultat du filtre de Kalman s'améliore légèrement, car l'erreur, la covariance et l'aire de l'ellipse diminuent, indiquant que plus il y a de références, mieux fonctionne l'algorithme.

Cela se produit probablement parce qu'en augmentant le nombre de références, on augmente la chance que l'algorithme sélectionne une référence ni trop éloignée ni trop proche, ce qui apporte un bon équilibre entre les erreurs de distance et d'angle. En effet, à grande distance, l'erreur d'angle est plus significative, tandis qu'à courte distance, l'erreur de distance l'est davantage.

**Remarque.** Cependant, il est à noter que dans cette implémentation, le filtre de Kalman connaît exactement la référence utilisée dans ses calculs.

Dans une application réelle, il y aurait une incertitude quant au choix des références, ce qui, dans un environnement avec un nombre élevé de références, pourrait augmenter la probabilité d'une correspondance erronée, entraînant ainsi une augmentation de l'erreur du filtre de Kalman et, par conséquent, un résultat dégradé dans ce scénario.

## 8. Question 8

Ci-dessous, quelques interactions ont été réalisées en faisant varier le nombre d'amers sur la carte quand seulement les mesures de distance sont disponibles :

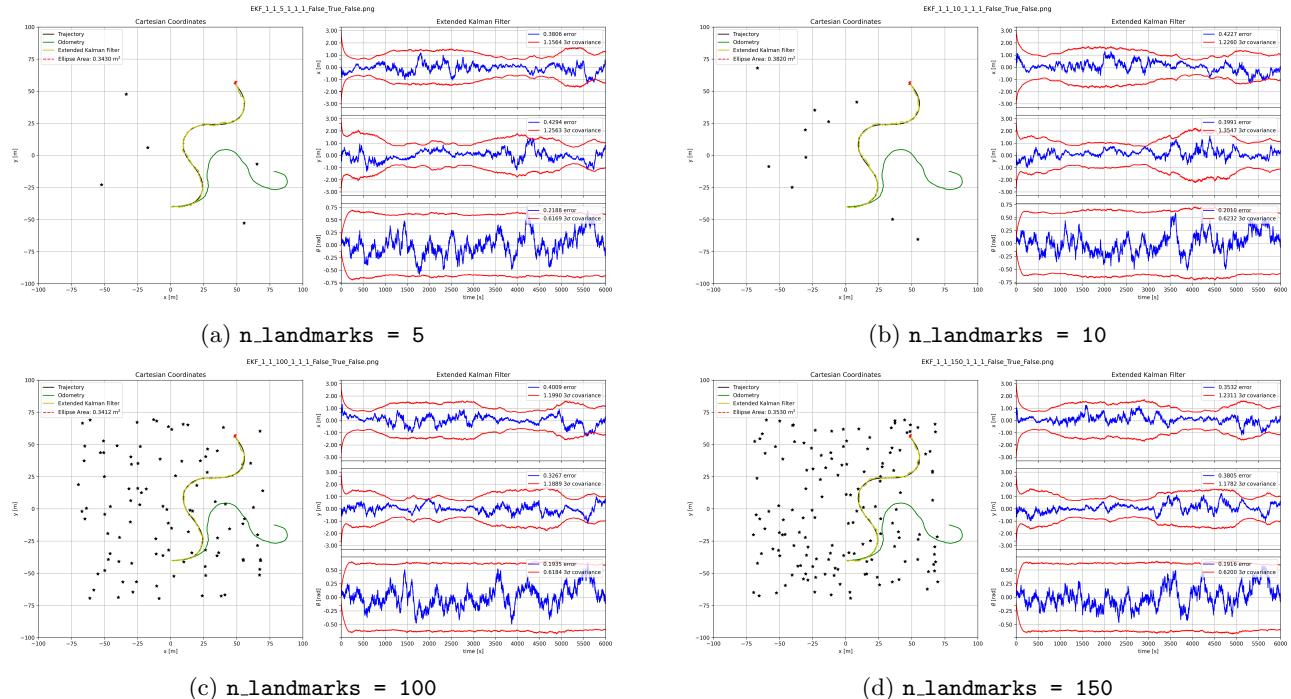


FIGURE 8.1 : Variation du nombre d'amers  $n_{\text{landmarks}}$  quand  $\text{range\_only} = \text{True}$

Il est à noter que lorsque seules les mesures de distance sont disponibles pour le calcul du filtre de Kalman, les résultats deviennent plus instables, la variance de tous les états étant plus élevée que dans le cas de référence. Ce comportement se manifeste également par une trajectoire moins fluide générée par le filtre.

Malgré cela, le filtre demeure précis, l'aire de l'ellipse ne présentant pas de différences significatives par rapport à celle du cas de référence. Ainsi, la variation du nombre de références sur la carte n'entraîne pas de changements notables dans le fonctionnement du filtre de Kalman.

**Remarque.** Cela démontre la capacité du filtre de Kalman à estimer efficacement l'état du système, même lorsque seules certaines mesures sont disponibles, en s'appuyant davantage sur ses prévisions dans ce contexte.

## 9. Question 9

Ci-dessous, quelques interactions ont été réalisées en faisant varier le nombre d'amers sur la carte quand seulement les mesures d'angle sont disponibles :

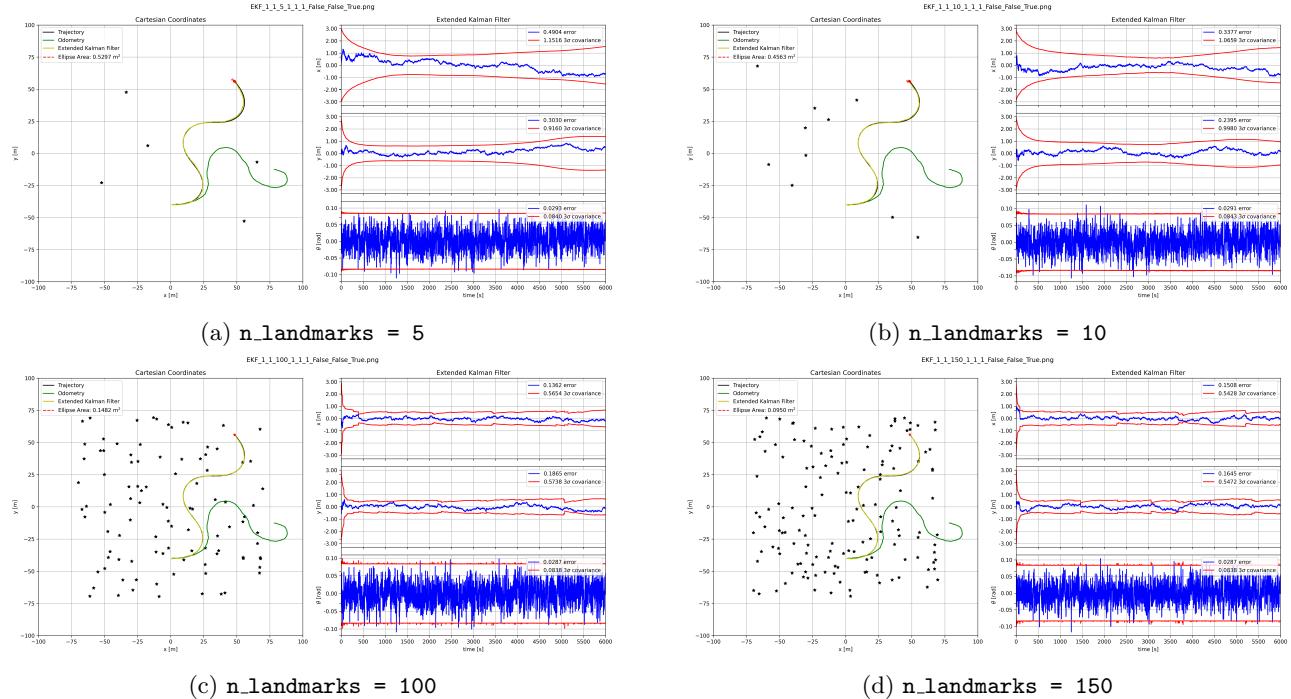


FIGURE 9.1 : Variation du nombre d'amers  $n\_landmarks$  quand  $\text{angle\_only} = \text{True}$

Il est à noter que lorsque seules les mesures d'angle sont disponibles pour le calcul du filtre de Kalman, les résultats restent relativement stables, comme l'indique la trajectoire calculée, même si la confiance dans l'estimation a diminué en raison de l'augmentation de la variance.

Dans ce cas, contrairement à la situation précédente, l'ajout de références supplémentaires dans la simulation améliore significativement l'estimation de la trajectoire. Cela indique que les angles sont plus sensibles à la présence ou à l'absence de références au cours d'une simulation.

**Remarque.** Cela suggère que les angles sont plus sensibles à l'estimation du filtre de Kalman, ce qui est logique, car une petite erreur dans un angle peut entraîner une propagation d'erreur significative dans la position de référence et, par conséquent, affecter la qualité de l'estimation de la trajectoire.