TP3: Particle filter for mobile robots localization RO12 - Navigation pour les systèmes autonomes

Gallego, Natalia
AST
ENSTA Paris
natalia.gallego@ensta-paris.fr

I. Introduction

Dans ces travaux pratiques, nous aborderons la mise en œuvre et l'analyse **d'un filtre particulier (PF)** pour la localisation et la navigation d'un véhicule mobile. Ce filtre est une approche statistique qui permet d'évaluer la position d'un véhicule à mesures de capteurs, tout en tenant compte des incertitudes inhérentes à la dynamique du système et aux mesures.

Dans un premier temps, nous examinons la structure du code quatre, identifions les différents paramètres du filtre et déterminons l'action des grandes parties du code, en notant la simulation du véhicule, les capteurs, l'odomètre et le filtre particulier. Complétez maintenant le code avec les équations du filtre particulier, et vérifiez les étapes de prédiction, de correction et d'ajustement, ainsi que les modèles dynamiques et de mesure.

Nous analysons l'impact des différents paramètres du filtre sur ses performances, en faisant varier le bruit de dynamique et de mesure, ainsi que le seuil de réé échantillonnage. De plus, nos simulations des scénarios de perte de mesure et étudiéons commentent ces conditions affectant le comportement du filtre. En explorant le nombre d'amers, nous évaluons comment cela influence la précision de l'estimation de la position.

Enfin, nous proposons une alternative à la méthode rééchantillonnage existante, dans le codant et en commentant ses résultats. Ce travail nous permettra non seulement de renforcer notre compréhension des filtres particuliers, mais aussi d'appliquer ces concepts dans un contexte pratique, en nous confrontant à des situations réelles que l'on peut rencontrer dans le domaine de la robotique et de la navigation autonome. .

II. Question 1 : Explication initiale du code (simulation de véhicule, capteurs, odométrie, filtre de Kalman).

Le code donné est divisé en 5 sections principales, chacune d'elles et les fonctions qui font partie de chacune de ces sections seront expliquées.

Voici une explication du code fourni qui simule la localisation d'un robot mobile à l'aide d'un filtre de particules. Le code est divisé en plusieurs sections, comprenant la simulation du monde, le modèle d'évolution, le modèle d'observation, ainsi que l'implémentation du filtre de particules. Chaque fonction sera décrite en détail.

A. Simulation du monde

1. **`Simulation.__init__`**: Cette méthode initialise la simulation avec le temps final, le pas de prédiction, les états du robot (vérité terrain et odométrie), la carte, les covariances des bruits, et le pas de mesure.

Paramètres :

- `Tf`: Temps final de la simulation.
- `dt_pred`: Intervalle entre les prédictions dynamiques.
- `xTrue`: État initial réel du robot.
- `QTrue`: Covariance du bruit de mouvement réel.
- `xOdom`: État initial du robot pour l'odométrie.
- `Map`: Coordonnées des points de repère.
- `RTrue`: Covariance du bruit de mesure.
- `dt_meas`: Intervalle entre les mises à jour des mesures.

2. **`get_robot_control`**: Génère une commande de contrôle pour le robot en suivant une trajectoire sinusoïdale.

Paramètre :

- `k`: Étape de temps actuelle.

3. **`simulate_world`**: Simule la nouvelle position réelle du robot en appliquant la commande de contrôle et en mettant à jour sa position réelle.

Paramètre :

- `k`: Étape de temps actuelle.

4. **`get_odometry`**: Génère une odométrie bruitée à partir du modèle de mouvement et du bruit simulé.

Paramètre :

- `k`: Étape de temps actuelle.

5. **`get_observation`**: Simule une observation bruitée d'un point de repère aléatoire sur la carte à un instant donné.

Paramètre :

- `k`: Étape de temps actuelle.

B. Modèles du filtre de particules

1. **`motion_model`**: Modèle de mouvement du robot qui met à jour l'état en fonction de la commande de contrôle bruitée (**_tilda**) et du bruit de mouvement estimé (**`QEst`**).

Paramètres :

- `x`: État actuel estimé du robot.
- `u_tilda`: Commande de contrôle bruitée.
- `dt_pred`: Intervalle de temps de prédiction.
- `QEst`: Covariance du bruit de mouvement estimé.

2. **observation_model**: Modèle d'observation qui calcule une mesure prévisionnelle d'un point de repère observé en fonction de l'état du robot et des coordonnées du point de repère.

Paramètres :

- `xVeh`: État actuel du robot.
- `iFeature`: Indice du point de repère observé.
- `Map`: Carte contenant les coordonnées de tous les points de repère.

C. Filtre de particules

1. **re_sampling**: Effectue le rééchantillonnage des particules avec un algorithme de rééchantillonnage à faible variance. Cela permet de concentrer les particules dans les zones où les poids sont plus élevés.

Paramètres :

- `px`: États des particules.
- `pw`: Poids associés à chaque particule.

D. Fonctions utilitaires

1. **angle_wrap**: S'assure que l'angle est toujours dans l'intervalle $[-,]$, pour éviter les discontinuités dans les calculs d'angle.

Paramètre :

- `a`: Angle à ajuster.

2. **tcomp**: Compose deux transformations en appliquant un contrôle bruité au robot pour mettre à jour sa position et son orientation.

Paramètres :

- `tab`: Transformation actuelle du robot (position et orientation).
- `tbc`: Commande de contrôle bruitée.
- `dt`: Intervalle de temps de prédiction.

3. **plotParticles**: Affiche les trajectoires réelles, odométriques et estimées, ainsi que les particules actuelles dans une simulation.

Paramètres :

- `simulation`: Objet de simulation contenant les informations de l'environnement.
- `k`: Étape de temps actuelle.
- `iFeature`: Indice du point de repère observé.
- `hxTrue`, `hxOdom`, `hxEst`: Trajectoires réelles, odométriques et estimées.

E. Programme principal

La simulation est paramétrée pour durer 1000 secondes avec un pas de temps de 1 seconde entre chaque prédiction et mesure. Les points de repère sont placés de manière aléatoire sur une carte, et 300 particules sont utilisées pour estimer la position du robot.

III. Question 2 : Code complet

A. Fonction `motion_model`

Cette fonction implémente le modèle de mouvement du véhicule, basé sur un modèle cinématique. La position du véhicule est mise à jour à l'aide des équations de mouvement d'un corps rigide dans le plan :

$$\alpha = \tilde{v}_k^x + W_k^{V_x} \tag{1}$$

$$\beta = \tilde{v}_k^y + W_k^{V_y} \tag{2}$$

$$x_k = \begin{pmatrix} x_{k-1} + ((\alpha)\cos(\theta_{k-1}) - (\beta)\sin(\theta_{k-1}))\Delta t \\ y_{k-1} + ((\alpha)\sin(\theta_{k-1}) - (\beta)\cos(\theta_{k-1}))\Delta t \\ \theta_{k-1} + (\tilde{w}_k + W_k^w)\Delta t \end{pmatrix} \tag{3}$$

Ces équations modélisent le mouvement du véhicule en termes de vitesse et de taux de virage. L'angle θ est ensuite ajusté à l'aide de la fonction **angle_wrap** pour garantir que la valeur reste dans la plage $[-\pi, \pi]$.

```
def motion_model(x, u_tilda, dt_pred, QEst):
    x_pos, y_pos, theta = x[0, 0], x[1, 0], x[2, 0]
    Vx, Vy, omega = u_tilda[0, 0], u_tilda[1, 0],
    u_tilda[2, 0]
    w_k = np.random.multivariate_normal([0, 0, 0],
    QEst)

    xPred = np.array([[x_pos + (Vx + w_k[0]) * cos(
    theta) * dt_pred - (Vy + w_k[1]) * sin(theta) *
    dt_pred],
                      [y_pos + (Vx + w_k[0]) * sin(
    theta) * dt_pred + (Vy + w_k[1]) * cos(theta) *
    dt_pred],
                      [angle_wrap(theta) + (omega +
     w_k[2]) * dt_pred]])
    return xPred
```

B. Fonction `observation_model`

La fonction implémente le modèle d'observation du filtre à particules. Le véhicule peut mesurer la distance (**range**) et l'angle (**bearing**) par rapport à un élément observé. Les équations utilisées sont les suivantes :

$$y_k = h(x_k) = \begin{pmatrix} r_k \\ \varphi_k \end{pmatrix} = \begin{pmatrix} sqrt(x_k^P - x_k)^2 + (y_k^P - y_k)^2) \\ atan\frac{(y_k^P - y_k)}{(x_k^P - x_k)} - \theta_k \end{pmatrix} \tag{4}$$

La range est la distance euclidienne entre la position du véhicule et l'élément observé, et le **bearing** est l'angle relatif entre l'orientation du véhicule et l'élément observé. Encore une fois, l'angle de **bearing** est défini sur $[-\pi, \pi]$ avec **angle_wrap**.

```
def observation_model(xVeh, iFeature, Map):
    x_veh, y_veh, theta_veh = xVeh[0, 0], xVeh[1,
    0], xVeh[2, 0]
    x_feat, y_feat = Map[0, iFeature], Map[1,
    iFeature]

    dx = x_feat - x_veh
    dy = y_feat - y_veh
    range_ = np.sqrt(dx**2 + dy**2)
```

```
    bearing = atan2(dy, dx) - theta_veh
    bearing = angle_wrap(bearing)

    z = np.array([[range_], [bearing]])
    return z
```

C. Boucle temporelle

Il s'agit de la boucle principale qui exécute la simulation au fil du temps. À chaque itération, les étapes suivantes sont effectuées :

1. **Initialisation des particules :**

Au début de la simulation, les particules x_0^i sont initialisées à partir d'une distribution gaussienne, comme décrit dans l'équation :

$$x_0^i \sim \mathcal{N}(\hat{x_o},\, \hat{P}_0)_{\,i\in[1,N]} \tag{5}$$

Cela signifie que les particules initiales sont distribuées selon une distribution normale centrée sur une estimation initiale $\hat{x}_0$, avec une covariance initiale $\hat{P}_0$.

2. **Prédiction du mouvement :** Lors de la simulation, à chaque étape k, les particules sont mises à jour à l'aide du modèle de mouvement (fonction **`motion_model`**), qui est décrit par l'équation d'évolution :

$$x_k^i = f(x_{k-1}^i, u_k, v_k^i) \tag{6}$$

Ici f est le modèle de mouvement, u_k est la commande de contrôle et v_k^i est le bruit du processus, qui suit une distribution gaussienne :

$$v_k^i \sim \mathcal{N}(0,\, Q_k)\, \forall i \in [1, N] \tag{7}$$

Dans le code, ceci est implémenté dans la boucle for p in range(nParticles) où chaque particule est mise à jour avec **`motion_model`**.

3. Prédiction de l'observation et du calcul de l'innovation :

Ensuite, si des observations sont disponibles (z n'est pas Aucun), l'observation prédite est calculée pour chaque particule à l'aide du modèle d'observation (fonction **`observation_model`**). La différence entre l'observation réelle z et l'observation prédite $h(x_k^i)$ est l'innovation :

$$Innov = y_k - h(x_k^i) \tag{8}$$

Cela se fait dans le code avec **`Innov = z - zPred`** et l'angle est enveloppé avec **`angle_wrap`**.

4. Mise à jour du poids des particules :

Les poids des particules w_k^i sont mis à jour en fonction de l'innovation. Tout d'abord, un poids temporel $\tilde{w}_k^i$ est calculé selon l'équation :

$$\tilde{w}_k^i = w_{k-1}^i exp(-\frac{1}{2}(y_k - h(x_k^i))^T R_k^{-1}((y_k - h(x_k^i)))) \tag{9}$$

Les poids sont ensuite normalisés pour que leur somme soit égale à 1 :

$$w_k^i = \frac{\tilde{w}_k^i}{\sum_i \tilde{w}_k^i} \tag{10}$$

5. Estimation de l'état et de la covariance :

Après mise à jour des poids, l'état estimé $\hat{x}_k$ est la moyenne pondérée des particules :

$$\hat{x}_k = \sum_{i=1}^{N} w_k^i x_k^i \tag{11}$$

La covariance estimée $\hat{P}_k$ est également calculée comme une moyenne pondérée :

$$\hat{P}_k = \sum_{i=1}^{N} w_k^i (x_k^i - \hat{x}_k)(x_k^i - \hat{x}_k)^T \tag{12}$$

6. Condition de rééchantillonnage :

Le nombre effectif de particules N_{eff} est calculé, ce qui permet de vérifier si les particules doivent être rééchantillonnées :

$$N_{eff} = \frac{1}{\sum_i (w_k^i)^2} < \theta_{eff} N \tag{13}$$

Si N_{eff} est inférieur à un seuil $\theta_{eff} N$, les particules sont rééchantillonnées pour empêcher quelques particules de dominer.

Ce cycle est répété à chaque étape de la simulation, ajustant l'état estimé du véhicule en fonction des observations et du mouvement prévu.

```
for k in range(1, simulation.nSteps):
    htime.append(k*simulation.dt_pred)
    simulation.simulate_world(k)
    xOdom, u_tilda = simulation.get_odometry(k)

    for p in range(nParticles):
        xParticles[:, p:p+1] = motion_model(
    xParticles[:, p:p+1], u_tilda, simulation.
    dt_pred, QEst)

    [z, iFeature] = simulation.get_observation(k)

    if z is not None:
        for p in range(nParticles):
            zPred = observation_model(xParticles[:,
    p:p+1], iFeature, simulation.Map)

            Innov = z - zPred
            Innov[1] = angle_wrap(Innov[1])

            wp[p] *= np.exp(-0.5 * Innov.T @ np.
    linalg.inv(REst) @ Innov)
    wp /= np.sum(wp)

    xEst = np.average(xParticles, axis=1, weights=wp
    )
    xEst = np.expand_dims(xEst, axis=1)

    PEst = np.cov(xParticles, aweights=wp, ddof=0)
    xSTD = np.sqrt(np.diag(PEst))
    xSTD = np.expand_dims(xSTD, axis=1)

    theta_eff = 0.1
    Nth = nParticles * theta_eff
    Neff = 1.0 / np.sum(wp**2)
    if Neff < Nth:
        xParticles, wp = re_sampling(xParticles, wp)
```

D. Résultat de l'utilisation du filtre à particules

Le premier résultat du filtre à particules est présenté ci-dessous. En cela, vous pouvez voir deux graphiques principaux.

Dans le graphique de gauche, nous pouvons voir plusieurs éléments, à savoir :

- Une ligne noire indique la trajectoire réelle du robot, c'est-à-dire sa véritable position à chaque pas de temps.
- Une ligne verte montre la trajectoire estimée du robot basée uniquement sur les lectures d'odométrie, qui contiennent des erreurs cumulées.
- Une ligne rouge représente l'estimation de la trajectoire du robot basée sur le filtre à particules. Les points rouges montrent l'estimation de la position actuelle.
- Une ligne bleue qui représente l'observation d'un point de repère depuis la position actuelle du robot. Cette ligne relie la position actuelle du robot à la position du point de référence observé.
- Certains points orange représentent les positions estimées des particules. La dispersion des particules reflète le degré d'incertitude de l'estimation du filtre.

D'autre part, le graphique de droite montre les erreurs dans les estimations du filtre à particules par rapport à la véritable trajectoire du robot, pour le cas de x, y et θ :

La courbe bleue représente l'erreur réelle et les lignes rouges montrent la plage de 3 écarts types ($\pm\ 3\sigma$) de l'erreur. Si la courbe bleue se situe entre les lignes rouges, cela signifie que le filtre fonctionne correctement.

En regardant les résultats, et le fait que pour le graphique de gauche, la ligne rouge suit correctement la trajectoire réelle (ligne noire), et pour le graphique de droite, toutes les erreurs sont dans l'intervalle désigné, nous pouvons conclure. Après cela configuration initiale des paramètres, le filtre à particules remplit sa fonction et corrige correctement la trajectoire du robot.

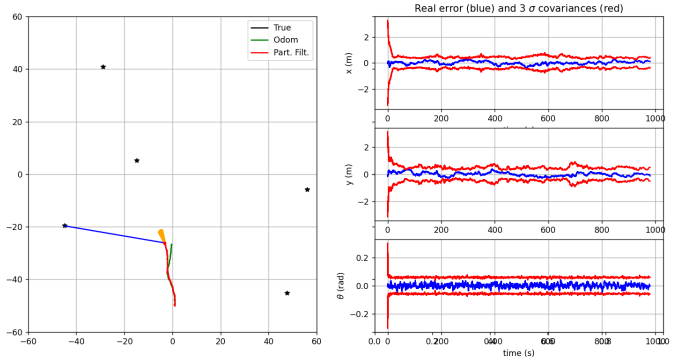

Fig. 1. Résultat du filtre à particules.

IV. Question 3: Variation du bruit de dynamique du filtre (matrice **QEst**)

La variable **QEst** dans un filtre à particules représente la covariance de l'estimation du modèle de mouvement, indiquant le niveau d'incertitude ou de bruit associé à l'odométrie du robot. La variation des valeurs QEst influence de manière significative le comportement du filtre et la dispersion des particules qui représentent les positions possibles du robot.

A. Petites valeurs de **QEst** (faible incertitude en odométrie)

Lorsque **QEst** prend de petites valeurs, le filtre à particules considère que l'odométrie du robot est très précise et qu'il y a peu de bruit dans ses mouvements. Dans ce cas, les particules ont tendance à rester plus proches les unes des autres et autour de l'estimation actuelle de l'état par le robot.

Comme le montre le graphique 2, cela se traduit par un regroupement dense de particules (lignes orange), montrant peu de dispersion lorsque le robot se déplace. Le recours à l'odométrie permet au filtre de suivre de près la trajectoire estimée.

D'un autre côté, comme nous pouvons le voir sur les résultats 3, le filtre sous-estime la véritable incertitude sur l'orientation, ce qui provoque de grosses erreurs dans θ, et ces erreurs dépassent même les limites autorisées (lignes rouges). Cela indique que le filtre n'est pas suffisamment flexible pour prendre en compte les erreurs cumulées d'odométrie, ce qui est particulièrement critique dans l'estimation des relèvements.

De plus, cela conduit finalement à ce que les estimations de position s'écartent également des limites autorisées (lignes rouges).

B. Valeurs **QEst** élevées (incertitude élevée en odométrie)

En revanche, lorsque **QEst** a des valeurs élevées, le filtre suppose que l'odométrie présente un niveau de bruit considérable et que les mouvements du robot sont incertains. Dans ce scénario, les particules se dispersent davantage dans l'environnement à mesure que le robot se déplace, ce qui indique une confiance moindre dans l'estimation basée sur l'odométrie. Ceci est visible dans les graphiques, car lorsque **QEst** prend la valeur 20, les lignes orange semblent plus dispersées.

Cela fait que la trajectoire estimée dépend davantage des observations de points de repère plutôt que du suivi direct de l'odométrie. En conséquence, le chemin semble plus sinueux ou instable par rapport au chemin réel.

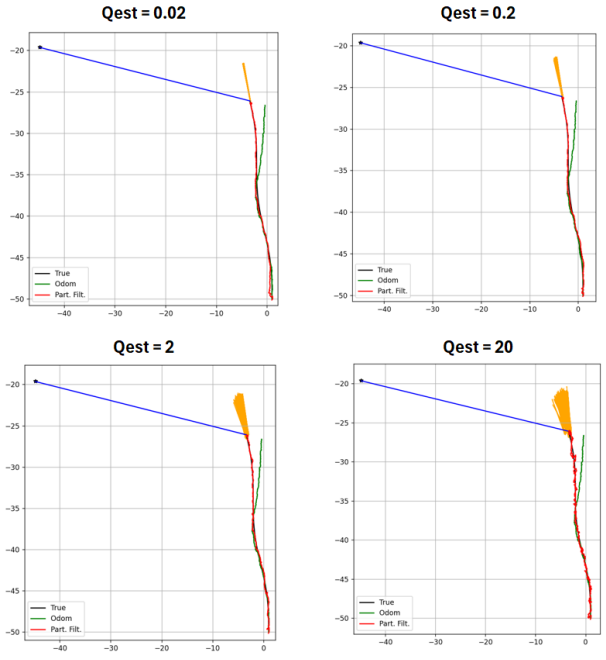

Fig. 2. Résultats du filtre à particules faisant varier QEst.

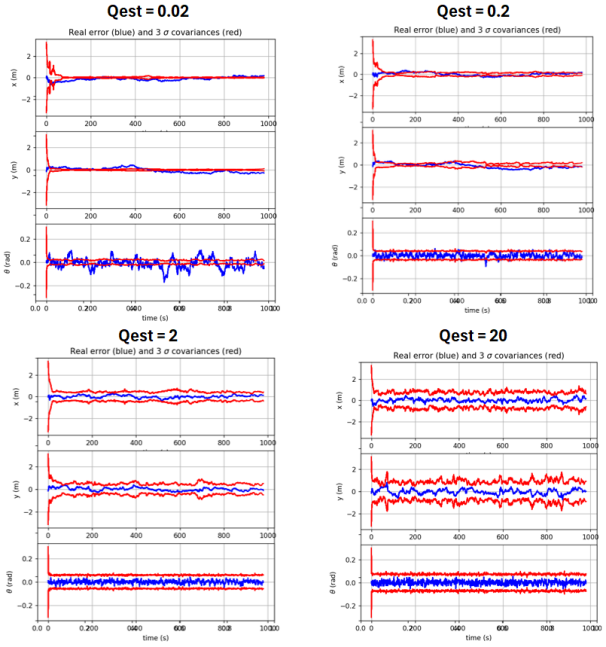

Fig. 3. Erreurs dans la prédiction du filtre à particules variant QEst.

V. Question 4 : Variation du bruit de dynamique du filtre (matrice **REst**)

REst représente la covariance du bruit de mesure ou de l'incertitude associée aux mesures des capteurs dans un filtre à particules. Cela modifie la façon dont le filtre interprète la précision des observations qu'il reçoit.

A. *Petites valeurs de **REst** (faible incertitude de mesure)*

Cela signifie que le filtre s'appuie fortement sur les mesures reçues par les capteurs. Autrement dit, le filtre suppose que les observations sont très précises, avec peu de biais ou de bruit. Cela fait, comme on peut le voir sur la figure 4, que les particules se regroupent plus rapidement et plus étroitement autour de la position réelle du robot (lignes orange). De plus, nous voyons comment l'estimation de la position du robot s'ajuste de manière plus instable à la trajectoire réelle du robot. Ceci, combiné aux résultats précédents, ne nous permet pas d'identifier l'effet du bruit de l'odométrie sur les résultats de notre filtre à particules.

D'autre part, en examinant les résultats de la figure 5, nous constatons que l'erreur de prédiction a tendance à sortir plus fréquemment des limites rouges ($\pm 3\sigma$), ce qui indique que le filtre fait des prédictions plus serrées, mais aussi plus susceptible de être en dehors des marges de confiance. En effet, en s'appuyant trop sur les mesures, le filtre devient moins tolérant au bruit ou aux imprécisions des observations.

B. *Valeurs **REst** élevées (incertitude de mesure élevée)*

Cela indique que le filtre s'appuie moins sur les mesures et suppose que les observations contiennent un plus grand degré de bruit ou d'incertitude. Cela fait, comme on peut le constater, que les particules sont plus dispersées car le filtre ne fait pas autant confiance aux mesures, ce qui reflète un plus grand degré d'incertitude sur la position du robot.

De plus, en examinant les résultats de la figure 4, nous constatons que les erreurs dans les prédictions ont tendance à se situer plus fréquemment dans les limites rouges, ce qui signifie que le filtre fait des prédictions plus conservatrices et est moins susceptible de s'écarter de manière significative des marges de erreur. Le filtre s'ajuste moins aux mesures individuelles, répartissant les risques et étant plus tolérant au bruit.

Si l'on compare les résultats obtenus dans les figures 4 et 5, avec ceux des figures 2 et 3 on constate qu'ils sont très similaires. En effet, **QEst** et **REst** influencent tous deux la façon dont les particules se répartissent et évoluent. Lorsque nous augmentons l'un ou l'autre, le filtre gère davantage d'incertitude, que ce soit dans le modèle ou dans les mesures. Cela conduit à une plus grande dispersion des particules, car le filtre est moins fiable. D'un autre côté, une valeur élevée de QEst et de REEst, dans notre cas, rend le filtre plus conservateur, ce qui rend les estimations moins précises dans les deux cas.

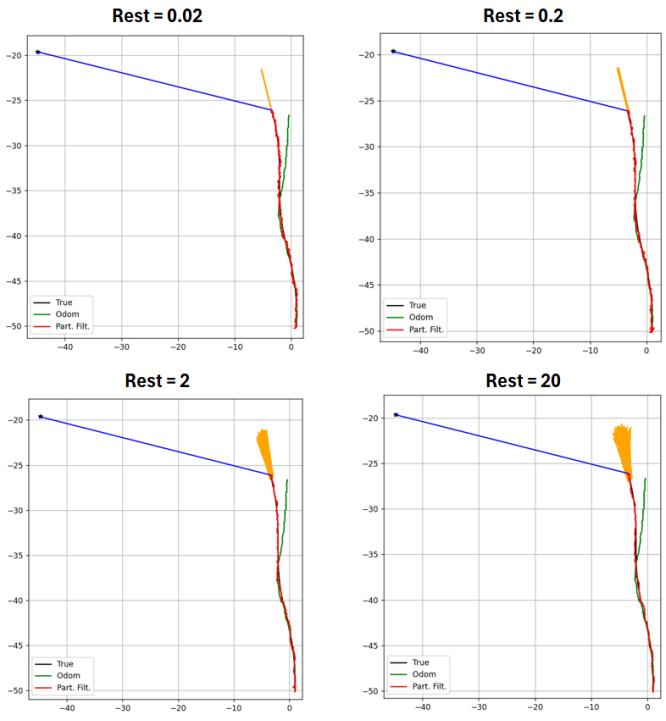

Fig. 4. Résultats du filtre à particules faisant varier REst.

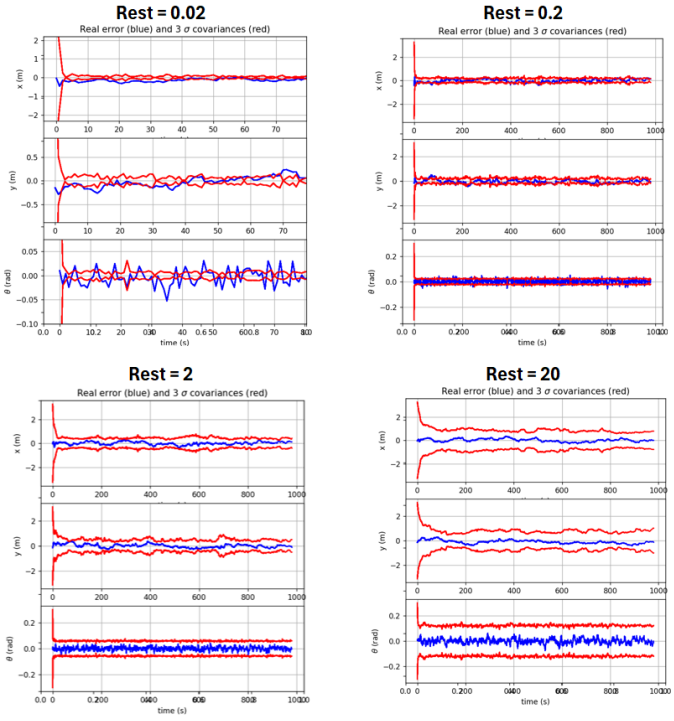

Fig. 5. Erreurs dans la prédiction du filtre à particules variant REst.

VI. Question 5 : Faites varier le seuil de rééchantillonnage `theta_eff` et tracez les histogrammes des poids

La valeur **`theta_eff`** (efficacité des particules) est une mesure clé pour évaluer la répartition des poids des particules dans un filtre à particules. Cette valeur indique combien de particules sont effectivement utiles pour estimer la position du robot.

Lorsque **`theta_eff`** prend des valeurs élevées, les poids sont répartis plus uniformément entre les particules, ce qui signifie qu'un plus grand nombre de particules contribuent de manière égale à l'estimation. C'est idéal, car cela implique que toutes les particules sont pertinentes et qu'il n'y a pas de domination excessive de certaines sur d'autres.

En revanche, lorsque **`theta_eff`** est faible, cela signifie que les poids des particules ne sont pas bien répartis : quelques particules ont un poids nettement plus élevé que les autres. Dans ce cas, la plupart des particules ne contribuent pas à l'estimation et le filtre subit une dégénérescence.

La dégénérescence se produit lorsque, après plusieurs itérations, de nombreuses particules se retrouvent avec un poids très petit ou nul, alors que seules quelques-unes ont un poids élevé. Cela réduit la diversité des particules et leur capacité à représenter adéquatement l'espace d'état, puisque les particules de faible poids cessent de contribuer à l'estimation finale.

Pour illustrer les phénomènes décrits ci-dessus, des histogrammes des poids des particules ont été générés en fonction de différentes valeurs de **`theta_eff`**. Ces histogrammes montrent comment les poids sont répartis entre les itérations du filtre à particules :

Lorsque **`theta_eff`** est élevé (proche du nombre total de particules, par exemple 0.95), l'histogramme (figure 6) montre une répartition des poids plus uniforme, ce qui signifie que de nombreuses particules ont des poids similaires. C'est le signe que le filtre à particules fonctionne bien et que les particules y contribuent de manière égale.

Lorsque **`theta_eff`** est faible (cas comme 0.1 ou 0.35), l'histogramme révèle que seules quelques particules ont un poids élevé, tandis que de nombreuses particules ont un poids proche de zéro. C'est un signe évident de dégénérescence. Dans cette situation, il est recommandé d'effectuer un rééchantillonnage pour régénérer la diversité des particules, en éliminant les particules de faible poids et en dupliquant celles de poids élevé.

L'histogramme superposé qui accompagne cette analyse présente l'évolution de la répartition des poids dans différentes itérations. Les couleurs plus claires (jaune) correspondent aux premières itérations, tandis que les couleurs plus foncées (orange) représentent les itérations ultérieures. Cela nous permet de visualiser comment la répartition des poids évolue dans le temps, mettant en évidence la dégénérescence progressive dans le cas où le rééchantillonnage n'est pas effectué à temps.

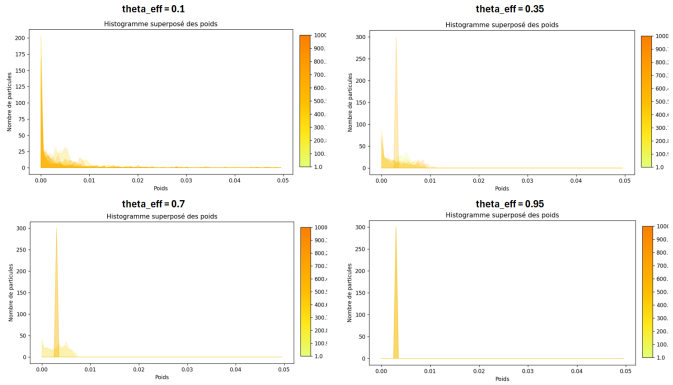

Fig. 6. Histogrammes des poids des filtres à particules en fonction des valeurs theta_eff.

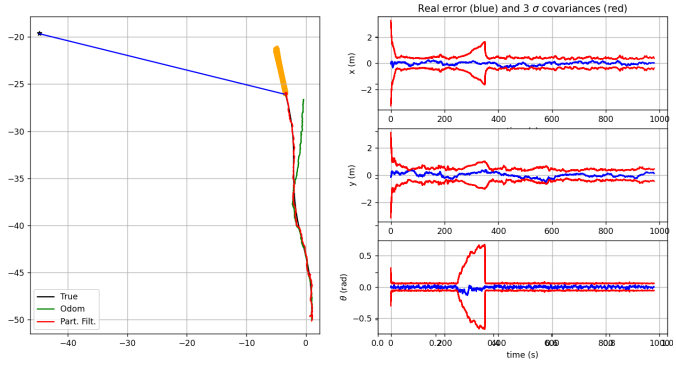

Fig. 7. Résultats du filtre à particules avec un écart de mesure compris entre 250 s et 350 s.

VII. Question 6 : Trou de mesures en utilisant la variable `notValidCondition`.

Lorsque nous définissons le paramètre **`notValidCondition`** sur **`True`** entre 250 et 350, nous spécifions une plage de pas de temps pendant lesquels le filtre à particules n'est pas valide. Lorsque le filtre à particules rencontre des conditions invalides, il s'appuie davantage sur le bruit du modèle de mouvement et moins sur les observations.

En examinant les résultats de la figure 7, nous pouvons voir que l'estimation du filtre à particules dans le graphique de trajectoire s'écarte un peu de la vraie trajectoire. Cela indique que le filtre tente d'estimer la position sur la base d'informations incorrectes ou insuffisantes. Cependant, comme nous l'avons montré, cet écart est très faible.

Dans les tracés d'erreurs, les erreurs dans les estimations de position (x, y) et d'orientation (θ) augmentent quelque peu sur cette plage. Bien que cette augmentation ne soit pas trop prononcée, nous ne voyons donc pas les courbes bleues dépasser les lignes rouges, indiquant que le filtre fonctionne correctement pendant ces étapes.

Le fait que les changements de trajectoire et les erreurs de trajectoire ne soient pas très élevés, malgré l'utilisation de la plage **`notValidCondition`**, nous pouvons conclure que le filtre à particules est robuste à certaines perturbations. Cela peut être dû au fait que le modèle de mouvement et le modèle d'observation sont bien conçus et que le filtre peut gérer des situations de bruit modéré sans que l'estimation ne s'écarte de manière significative.

De plus, nous pourrions supposer que la distribution des particules au début est efficace, de sorte que le filtre peut toujours effectuer des estimations précises même si elles se situent dans une plage de **`notValidCondition`**.

Pour ce qui précède, le code suivant a été utilisé.

```
if 250 <= k*self.dt_pred <= 350:
    notValidCondition = True
```

VIII. Question 7 : Modification de la fréquence de mesure à l'aide de `dt_meas`.

Le paramètre **`dt_meas`** représente l'intervalle de temps entre chaque mesure ou mise à jour du capteur (mesures de repère), donc sa modification fera varier la fréquence de mesure du système.

A. `dt_meas` prend de petites valeurs (fréquences de 10 et 100 Hz)

Lorsque nous réduisons **`dt_meas`**, les prédictions du système dépendent moins du mouvement réel entre les mesures, ce qui rend le filtre plus dépendant de l'odométrie. Cela peut être vu dans les résultats de la figure 8, où l'on voit que pour de petites valeurs de **`dt_meas`** la ligne rouge (prédiction) suit l'odométrie au lieu de la trajectoire réelle.

De plus, cela entraîne une plus grande dispersion des particules à chaque itération de prédiction, car chaque particule suit la dynamique du modèle (basée sur l'odométrie) avec son propre bruit. Ensuite, parce que les mesures n'ont pas assez de temps pour corriger cette dispersion, les particules sont désordonnées et largement dispersées, sans converger vers une solution proche de la trajectoire réelle.

D'un autre côté, en regardant les résultats d'erreur de prédiction sur la figure 9, nous voyons que, lorsque **`dt_meas`** est petit, la trajectoire moyenne (courbe bleue) est plus lisse et moins bruyante, puisque de nombreuses prédictions sont faites sur la base de petites différences de odométrie, qui ont tendance à ne pas varier beaucoup d'une itération à l'autre. Cependant, nous pouvons voir à quel point les lignes rouges semblent diverger. Cela indique que la plage marquée par 3σ, qui est la plage de confiance, montre une incertitude dans les prédictions du filtre. En d'autres termes, le filtre a peu confiance dans les estimations qu'il effectue.

B. *`dt_meas` prend des valeurs plus grandes (fréquences de 0,1 et 1Hz)*

En revanche, en augmentant **`dt_meas`**, le filtre a plus de temps pour intégrer les mesures, permettant une correction plus importante par rapport à l'odométrie. Ceci explique pourquoi lors de la visualisation des résultats (8) la ligne rouge suit mieux la trajectoire réelle. Cependant, avec des valeurs très élevées comme 10 s, l'incertitude augmente davantage entre les mesures, ce qui rend la prédiction plus erratique ou sinueuse car le système dispose de moins d'informations entre les mises à jour.

En examinant les résultats de 9 à mesure que nous augmentons **`dt_meas`**, chaque fois que nous faisons une prédiction, nous extrapolons avec plus d'incertitude, car le filtre dispose de moins d'informations mises à jour pour ajuster les particules. Par conséquent, la courbe bleue (prédiction moyenne) devient plus bruyante et plus ondulée, car le modèle de mouvement (odométrie) a plus de temps pour dériver avant de recevoir une correction de mesure. Bien que la prédiction suive la trajectoire générale, une plus grande incertitude entraîne un plus grand bruit dans l'estimation.

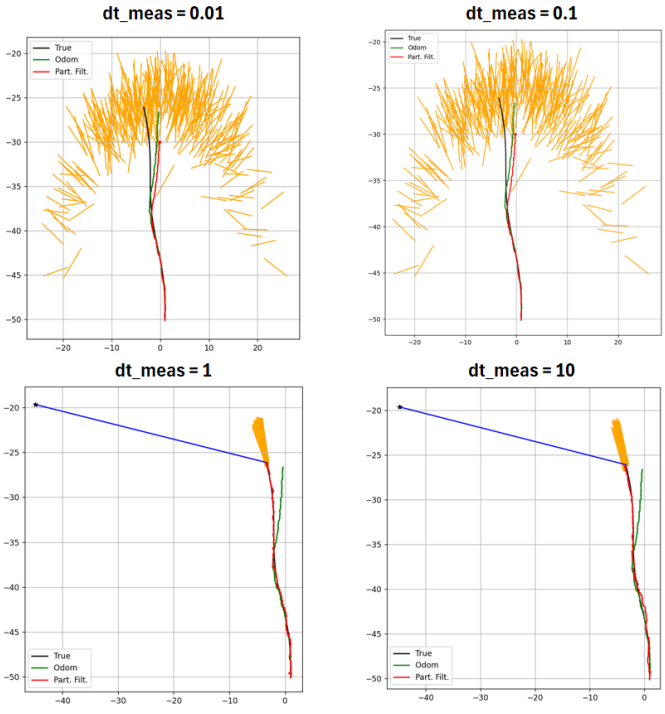

Fig. 8. Résultats du filtre à particules faisant varier dt_meas.

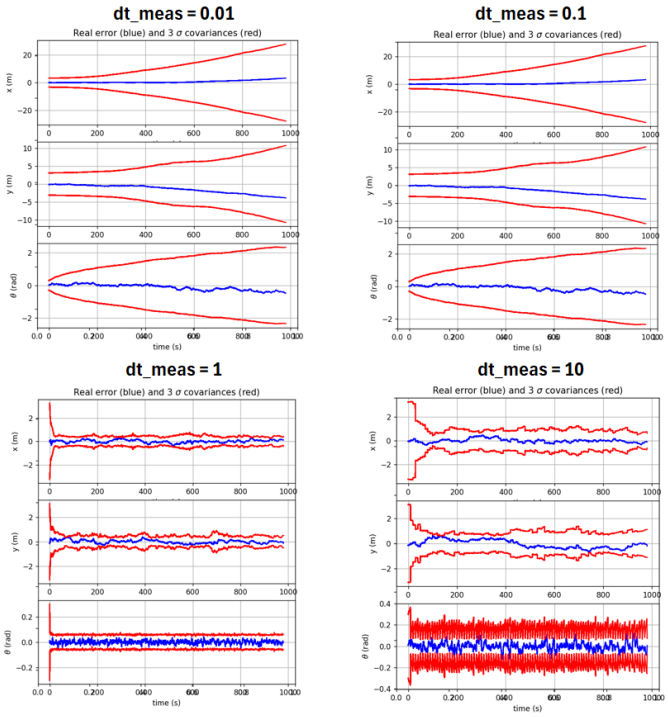

Fig. 9. Erreurs dans la prédiction du filtre à particules variant dt_meas.

IX. Question 8 : Variation du nombre d'amers (`nLandmarks`)

`nLandmarks` fait référence au nombre de points de repère utilisés pour la localisation et la cartographie. Il s'agit de points fixes dans l'environnement que le système peut détecter et utiliser pour aider à estimer la position du robot ou du capteur dans un espace donné.

A. *Petites valeurs `nLandmarks` (5, 10 points de repère)*

Avec un nombre réduit de points de repère, on pourrait s'attendre à ce que le filtre à particules ait des difficultés à estimer avec précision la position de l'objet, car il y a moins de points de repère pour guider l'estimation de la trajectoire.

Cependant, en regardant les résultats des figures 10 et 11 nous pouvons voir que ce n'est pas le cas, et que le filtre suit correctement la trajectoire réelle du robot.

En revanche, en regardant la figure 12, on voit que contrairement à ce qui était attendu, les erreurs de prédiction restent dans leur plage prédéfinie, et en les comparant avec les autres scénarios, on voit que les résultats sont presque identiques peu importe en nombre de points de référence.

B. *Grandes valeurs `nLandmarks` (50, 100 points de repère)*

L'augmentation du nombre de points de repère fournit plus d'informations au filtre, ce qui contribue à améliorer l'estimation de la position. En regardant les figures 10 et 11 nous voyons que la trajectoire estimée est correcte et correspond à la vraie trajectoire. Cependant, davantage de fluctuations sont observées dans la ligne rouge, ce qui peut être dû à une surabondance d'informations, en particulier

si les points de repère se trouvent dans des positions peu informatives ou bruyantes.

Ce même bruit peut être à l'origine du fait que la variabilité des erreurs de prédiction est un peu plus grande par rapport aux scénarios où il y a moins de points de référence (Figure 12).

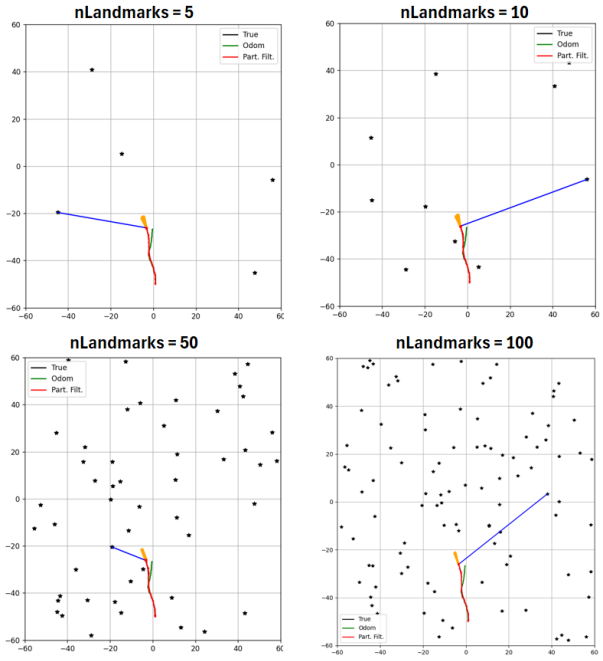

Fig. 10. Résultats du filtre à particules faisant varier nLandmarks.

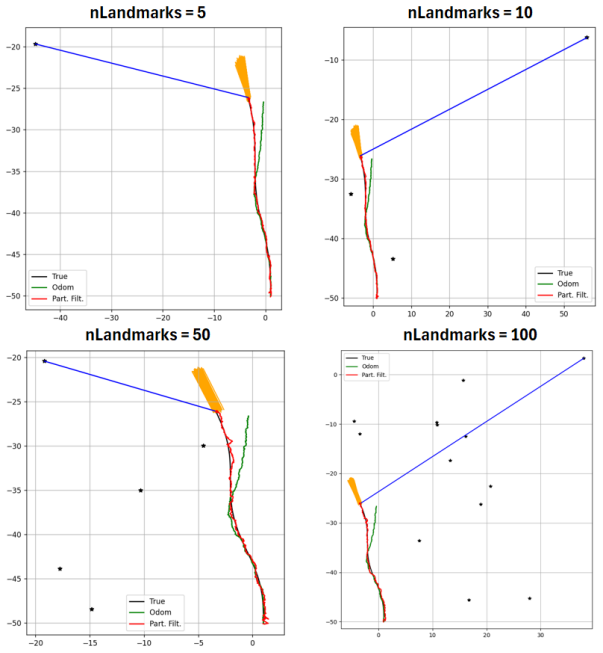

Fig. 11. Résultats du filtre à particules faisant varier nLandmarks, approche (zoom).

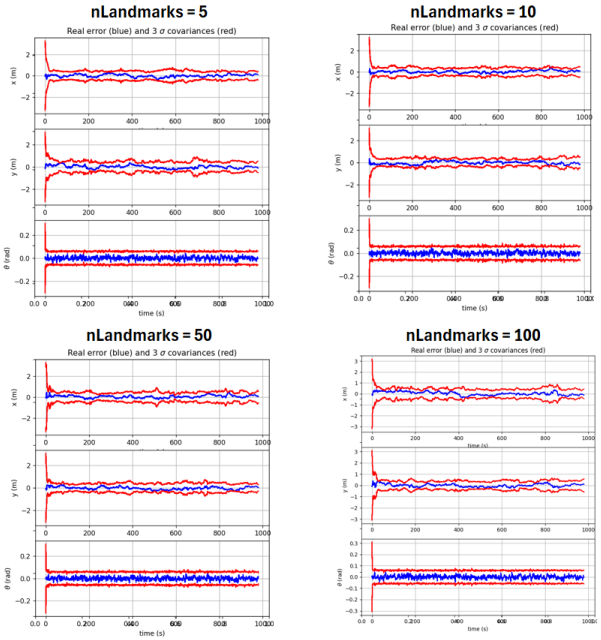

Fig. 12. Erreurs dans la prédiction du filtre à particules variant nLandmarks.

X. Question 9 : Algorithme de rééchantillonnage de réallocation

Il s'agit d'une méthode de rééchantillonnage utilisée dans les filtres à particules qui optimise la représentation de l'état du système grâce à une approche adaptative de la sélection des particules. Dans cet algorithme, un ensemble de particules est pris ($\{x_t^{(m)}\}$) avec leurs poids correspondants ($\{w_t^{(m)}\}$) et un nombre total taille de particule souhaitée (N). Le processus commence par évaluer chaque poids ($w_t^{(m)}$) pour déterminer combien de fois chaque particule doit être rééchantillonnée.

Pour les particules de poids significatif (c'est-à-dire ($w_t^{(m)} \geq \frac{1}{N}$)), le nombre de rééchantillons ($N_t^{(m)}$) est calculé en utilisant l'équation suivante :

$$N_t^{(m)} Floor(N \times w_t^{(m)}) \tag{14}$$

Cette valeur détermine combien de fois la particule ($x_t^{(m)}$) sera incluse dans le nouvel ensemble de particules ($\{\tilde{x}_t^{(n)}\}$). Pour chaque rééchantillonnage de la particule, son poids est ajusté comme suit :

$$\tilde{w}_t^{(n)} = \frac{w_t^{(m)}}{N_t^{(m)}} \tag{15}$$

Dans le cas de particules de poids inférieur à ($\frac{1}{N}$), l'algorithme introduit un élément d'aléatoire. Un nombre aléatoire est généré ($u \sim U(0, \frac{1}{N}]$) et il est vérifié si ($w_t^{(m)}$) est supérieur ou égal à u. Si cette condition est remplie, la particule est rééchantillonnée avec un poids uniforme ($\tilde{w}_t^{(n)} = \frac{1}{N}$).

À la fin du processus, toutes les particules rééchantillonnées sont compilées dans le nouvel ensemble, qui a une taille totale N^* donnée par :

$$N^* = n \tag{16}$$

où n est le nombre total de particules sélectionnées lors du rééchantillonnage. Cette approche garantit que les particules les plus pertinentes sont priorisées, ce qui améliore la précision de l'estimation de l'état du système et minimise l'influence des particules de faible poids.

Ci-dessous le pseudocode de cet algorithme (extrait de [1]) et le code python qui a été créé pour l'utiliser dans notre filtre à particules

Algorithm 1 Reallocation Resampling

$[\{\tilde{x}_t^{(n)}, \tilde{w}_t^{(n)}\}_{n=1}^{N^*}] =$ (Reallocation)Resample$[\{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}]$
$n = 0$
for $m = 1$ to M **do**
  if $w_t^{(m)} \geq 1/N$ **then**
    $N_t^{(m)} = Floor(N \times w_t^{(m)})(or N_t^{(m)} = Floor(N \times w_t^{(m)}) + 1)$
    for $h = 1$ to $N_t^{(m)}$ **do**
      $n = n + 1$
      $\tilde{x}_t^{(n)} = x_t^{(m)}; \tilde{w}_t^{(n)} = w_t^{(m)} / N_t^{(m)}$
    end for
  else
    $u \sim U(0, 1/N]$
    if $w^{(m)_t} \geq u$ **then**
      $n = n + 1$
      $\tilde{x}_t^{(n)} = x_t^{(m)}; \tilde{w}_t^{(n)} = 1/N$
    end if
  end if
end for
$N^* = n$
=0

```python
def reallocation_resampling(particles, weights):
    M = weights.shape[0]
    N = nParticles
    n = 0
    resampled_particles = []
    resampled_weights = []

    for m in range(M):
        if weights[m] >= 1 / N:
            N_m_t = int(np.floor(N * weights[m]))
            for h in range(N_m_t):
                n += 1
                resampled_particles.append(particles[:, m])
                resampled_weights.append(weights[m] / N_m_t)
        else:
            u = np.random.uniform(0, 1 / N)
            if weights[m] >= u:
                n += 1
                resampled_particles.append(particles[:, m])
                resampled_weights.append(1 / N)

    while len(resampled_particles) < N:
        idx = np.random.choice(M)
        resampled_particles.append(particles[:, idx])

    resampled_particles = np.array(resampled_particles).T
    resampled_weights = np.array(resampled_weights)

    return resampled_particles, resampled_weights
```

Les résultats obtenus sont présentés dans la Figure 13 en utilisant cet algorithme de ré-échantillonnage au lieu de celui qui a été utilisé dans chacune des questions précédentes.

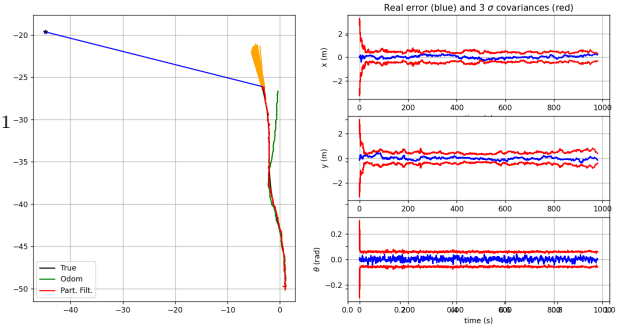

Fig. 13. Résultats utilisant l'algorithme de rééchantillonnage de réallocation.

A. *Comparaison entre les algorithmes de rééchantillonnage*

La méthode de rééchantillonnage initial présente une stratégie différente pour le rééchantillonnage des particules. Dans cette approche, les indices de rééchantillonnage sont générés sur la base de pondérations cumulées. Tout d'abord, la somme cumulée des poids est calculée :

$$w_{\text{cum}}^{(m)} = \sum_{j=1}^{m} w_t^{(j)} \tag{17}$$

Ensuite, des identifiants de rééchantillonnage uniformément distribués (r_i) sont créés sur l'intervalle ([0, 1]) :

$$r_i = i \cdot \frac{1}{N} + U(0, \frac{1}{N}) \quad \text{for } i = 1, 2, \ldots, N \tag{18}$$

L'algorithme parcourt ces identifiants et utilise la comparaison avec $(w_{\text{cum}}^{(m)})$ pour déterminer les particules à rééchantillonner. Chaque particule sélectionnée est prise selon que l'identifiant de rééchantillonnage tombe ou non dans la somme cumulée des poids.

La principale différence entre les deux méthodes réside dans leur approche. Alors que le premier rééchantillonnage utilise une stratégie systématique et structurée, le rééchantillonnage de réallocation permet une sélection plus flexible et adaptative des particules, en donnant la priorité à celles ayant un poids plus élevé et en réduisant l'influence des particules moins pertinentes. Cela peut aboutir à une meilleure représentation de l'état du système, en particulier dans les scénarios où la répartition du poids est inégale ou asymétrique, optimisant ainsi les performances du filtre à particules.

Même si le changement de méthode de rééchantillonnage devrait modifier considérablement les résultats obtenus, en comparant les figures 1 et 13, nous nous rendons compte

que les résultats obtenus avec les deux méthodes sont remarquablement similaires en pratique. Cela suggère qu'en termes de performances, les deux approches sont efficaces pour résoudre le problème du rééchantillonnage dans les filtres à particules, en fournissant des estimations comparables de l'état du système.

La cohérence des résultats peut être attribuée à la nature du processus de rééchantillonnage, qui a pour objectif principal de mettre en évidence les particules les plus significatives et de réduire l'influence de celles de faible poids. Les approches de réallocation et de faible variance remplissent cette fonction, bien qu'elles le fassent par le biais de mécanismes différents.

XI. Conclusions

Le filtre à particules s'est avéré être une approche robuste et efficace pour l'estimation dans les systèmes dynamiques, même dans diverses conditions d'incertitude et de mesure. L'un de ses principaux atouts est sa capacité à s'adapter à différentes configurations de **`QEst`** (incertitude dans le modèle) et **`REst`** (incertitude dans les mesures) sans perdre en précision dans les estimations. Cela signifie que malgré les variations dans la qualité du modèle et des mesures, le filtre continue de fournir des résultats cohérents et fiables.

De plus, le filtre à particules est insensible à la variabilité du nombre de points de repère. Que l'on utilise peu ou beaucoup de repères, les estimations de position et de trajectoire restent satisfaisantes. Même si un plus grand nombre de points de repère peut introduire du bruit en raison de mesures supplémentaires, le filtre conserve sa capacité à suivre la véritable trajectoire sans écarts significatifs.

Un aspect particulièrement remarquable est la bonne performance du filtre à particules lorsqu'il y a des périodes pendant lesquelles les mesures des capteurs ne sont pas obtenues. Par exemple, entre t = 250 s et t = 350 s, en utilisant la variable **`notValidCondition`**, le filtre continue de fournir des prédictions raisonnables, en s'appuyant sur son modèle de mouvement. Cela démontre sa capacité à gérer efficacement les lacunes dans les mesures et à éviter une dégradation significative des estimations pendant ces périodes sans données. La capacité de continuer à fonctionner sur une large gamme de fréquences de mesure renforce sa flexibilité et son applicabilité dans différents scénarios.

Enfin, le paramètre thêta joue un rôle crucial dans l'équilibrage des poids des particules. **`theta_eff`** contrôle la redistribution des poids, ce qui empêche quelques particules de dominer l'estimation, garantissant ainsi le maintien de la diversité des particules. Cela permet au filtre de rester précis et équilibré au fil du temps, même en présence d'incertitude ou de bruit, garantissant ainsi que toutes les particules pertinentes contribuent à l'estimation finale.

En conclusion, le filtre à particules est une méthode polyvalente et efficace qui peut être adaptée à un large éventail de situations d'incertitude et de conditions de mesure. Sa robustesse aux variations de paramètres, sa capacité à gérer les périodes sans mesures et sa flexibilité dans la configuration des amers le rendent idéal pour les systèmes dynamiques complexes.

XII. GitHub

Vous pouvez vous référer à la référence suivante pour voir les codes de nœud du projet: GitHub/RO12/TP3

References

[1] Tianqing LI, Miodrag BOLIC, and Petar M. DJURIC. Resampling methods for particle filtering: Classification, implementation, and strategies. *IEEE Signal Processing Magazine*, 32(3):70–86, 2015.