

## TD3: Localisation et cartographie par EKF et évitement d'obstacles

### Solution des Exercices 1 et 2

#### **Exercice 1 : Créer et déplacer un robot mobile**

Utiliser la classe « Vehicle », qui simule un robot car-like (ou tricycle), pour créer un robot et le déplacer suivant une trajectoire aléatoire et récupérer sa position instantanée.

- Créer un robot « veh » avec une covariance odométrique (rappel  $\Sigma_{\Delta}$ ) et afficher son état initial.

```
>> veh = Vehicle(diag([0.1, 1*pi/180].^2));  
>> veh =  
Vehicle object  
L=1, maxspeed=5, alphasim=0.5, T=0.100000, nhist=0  
V=(0,0)  
x=0, y=0, theta=0
```

- Utiliser la fonction « update » pour appliquer une vitesse de 0.2 m/s avec une orientation de 0.1 rad pour un pas de temps. Afficher la nouvelle position du robot.

```
>> odo = veh.update([0.2, 0.1]);  
>> veh =  
Vehicle object  
L=1, maxspeed=5, alphasim=0.5, T=0.100000, nhist=1  
V=(0,0)  
x=0.02, y=0, theta=0.002
```

Ci-dessus, « odo », représente l'estimation odométrique bruitée.

- Attacher un pilote (« driver ») au robot pour le faire déplacer suivant une trajectoire aléatoire dans une région  $-10 < x < 10$  mètres,  $-10 < y < 10$  mètres.

```
>> veh.add_driver(RandomPath(10))
```

- Visualiser le mouvement du robot pour  $N = 1000$  instants en utilisant l'option « run », et récupérer la pose instantanée du robot.

```
>> veh.run(1000)
```

#### **Exercice 2 : Utilisation du filtre de Kalman étendu pour la localisation et la cartographie**

Cet exercice utilise la classe « EKF » qui peut être utilisée pour la localisation basée sur une carte, pour la cartographie et pour la localisation et cartographie simultanées (SLAM).

##### **a) Estimation de la position d'un robot**

- Créer un véhicule « veh » avec une covariance odométrique  $S$ .

```
>> veh = Vehicle(S);
```

- Ajouter un pilote avec une trajectoire aléatoire

```
>> veh.add_driver(RandomPath(20, 2));
```

- Créer un filtre de Kalman avec une covariance estimée  $V_{\text{est}}$  et une covariance initiale de l'état  $P_0$ .

```
>> ekf = EKF(veh, Vest, P0);
```

- Exécuter le filtre pour N pas de temps.

```
>> ekf.run(N);
```

- Afficher le chemin du véhicule, superposer le chemin estimé, et ajouter des ellipses d'incertitude tous les 10 pas de temps.

```
>> hold on;  
>> veh.plot_xy('b');  
>> ekf.plot_xy('r');  
>> ekf.plot_ellipse(10);
```

## b) Localisation basée carte

- Créer un véhicule « veh » avec une covariance odométrique  $S$ .

```
>> veh = Vehicle(S);
```

- Ajouter un pilote avec une trajectoire aléatoire.

```
>> veh.add_driver(RandomPath(20, 2));
```

- Créer une carte avec 20 points caractéristiques.

```
>> map = Map(20);
```

- Créer un capteur qui utilise la carte et l'état du véhicule pour estimer les primitives du capteur avec une covariance  $W$ .

```
>> sensor = RangeBearingSensor(veh, map, W);
```

- Définir le filtre de Kalman avec les covariances estimées  $V_{\text{est}}$  et  $W_{\text{est}}$  et la covariance de l'état initial du véhicule  $P_0$ , puis exécuter le filtre pour N pas de temps.

```
>> ekf = EKF(veh, Vest, P0, sensor, West, map);  
>> ekf.run(N);
```

- Tracer la carte, afficher le chemin du véhicule, superposer le chemin estimé, et ajouter des ellipses d'incertitude tous les 10 pas de temps.

```
>> map.plot();  
>> veh.plot_xy('b');  
>> ekf.plot_xy('r');  
>> ekf.plot_ellipse([], 'g');
```

## c) Création de cartes par un robot

- Créer un véhicule « veh » avec une covariance odométrique  $S$ .

```
>> veh = Vehicle(S);
```

- Ajouter un pilote avec une trajectoire aléatoire.

```
>> veh.add_driver(RandomPath(20, 2));
```

- Créer un capteur qui utilise la carte et l'état du véhicule pour estimer les primitives du capteur avec une covariance  $\mathbf{W}$ .

```
>> sensor = RangeBearingSensor(veh, map, W);
```

- Définir le filtre de Kalman avec la covariance estimée  $\mathbf{W}_{\text{est}}$  et un véhicule « parfait » (définir une matrice de covariance  $[\mathbf{I}]$ ), puis exécuter le filtre pour  $N$  pas de temps.

```
>> ekf = EKF(veh, [], [], sensor, West, []);  
>> ekf.run(N);
```

- Tracer la carte et superposer les ellipses d'incertitude 3 sigma estimées.

```
>> map.plot();  
>> ekf.plot_map(3, 'g');
```

#### d) Localisation et cartographie simultanées (SLAM)

- Créer un véhicule « veh » avec une covariance odométrique  $\mathbf{S}$ .

```
>> veh = Vehicle(S);
```

- Ajouter un pilote avec une trajectoire aléatoire.

```
>> veh.add_driver(RandomPath(20, 2));
```

- Créer une carte avec 20 points caractéristiques.

```
>> map = Map(20);
```

- Créer un capteur qui utilise la carte et l'état du véhicule pour estimer les primitives du capteur avec une covariance  $\mathbf{W}$ .

```
>> sensor = RangeBearingSensor(veh, map, W);
```

- Définir le filtre de Kalman avec les covariances estimées  $\mathbf{V}_{\text{est}}$  et  $\mathbf{W}_{\text{est}}$  et la covariance de l'état initial du véhicule  $\mathbf{P}_0$ , puis exécuter le filtre pour  $N$  pas de temps pour estimer l'état du véhicule à chaque pas de temps et la carte.

```
>> ekf = EKF(veh, Vest, P0, sensor, W, []);  
>> ekf.run(N);
```

- Tracer la carte, afficher le chemin du véhicule, superposer le chemin estimé, et ajouter des ellipses d'incertitude tous les 10 pas de temps.

```
>> map.plot();  
>> veh.plot_xy('b');  
>> ekf.plot_xy('r');  
>> ekf.plot_ellipse(10, 'g');
```