Devoir d'informatique 5

D.Malka – MPSI 2018-2019 – Lycée Jeanne d'Albret

07.06.2019

Durée de l'épreuve : 2h00

L'usage de la calculatrice n'est pas autorisé.

L'énoncé de ce devoir comporte 7 pages.

- *Si, au cours de l'épreuve, vous repérez ce qui vous semble être une erreur d'énoncé, signalez le sur votre copie et poursuivez votre composition en expliquant les raisons des initiatives que vous êtes amené à prendre.*
- *Il ne faudra pas hésiter à formuler des commentaires. Le barème tiendra compte de ces initiatives ainsi que des qualités de rédaction de la copie.*
- *La numérotation des exercices doit être respectée. Les résultats doivent être systématiquement encadrés. Les pages doivent être numérotées de la façon suivante : n°page courante/nombre total de page.*

Données pour tous les exercices.

Rappels :

- l'instruction t.`append(elt)` ajoute l'argument `elt` en queue du tableau `t`,
- la fonction `len(s)` renvoie un entier égal à la longueur de la séquence `s` passée en argument,
- la fonction `float(n)` renvoie la valeur l'argument de l'argument `n` sous forme d'un flottant,
- la fonction `int(x)` renvoie la valeur l'argument `x` sous forme d'un entier. Si `x` est un flottant, la fonction `int` renvoie la partie entière de ce nombre.
- slicing : `T[i:j]` renvoie la copie du sous tableau de `T` constitué des éléments d'indice i (inclus) à j (exclus).
- slicing total : `T[:]` renvoie la copie de `T` dans son intégralité.
- `range(a,b)` renvoie un objet qui génère une séquence d'entiers de a (inclus) à b (exclus). Si seul b est précisé, $a = 0$.
- `range(a,b,p)` renvoie un objet qui génère une séquence d'entiers de a (inclus) à b (exclus) par pas de p. Si seul b est précisé, $a = 0$.
- `array(l)` du module `numpy` convertit la liste `l` en `ndarray`. Les opérations algébriques sont possibles sur des objets de type `ndarray` (ex : [1 2]+[3 4] renvoie [4 6]) pas sur les objets de type `list`.
- séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1 : `arange(0,10.1,0.1)` ou `linspace(0,10,100)`.
- tableau à deux dimensions (matrice) : `M=array([1,2,3],[3,4,5])`
- accéder à un élément : `M[1,2]` renvoie 5
- extraire une portion de tableau (2 premières colonnes) : `M[:,0:2]`
- tableau de 0 (2 lignes, 3 colonnes) : `zeros((2,3))`

1 Modélisation de la propagation d'une épidémie par automate cellulaire

Dans ce qui suit, on appelle *grille* de *taille* $n \times n$ une liste de n listes de longueur n, où n est un entier strictement positif.

Pour prendre en compte à la fois la dépendance temporelle et la la dépendance spatiale d'une contagion, il est possible de simuler la propagation d'une épidémie à l'aide d'une grille. Chaque case de la grille peut être dans un des quatre états suivants : saine, infectée, rétablie, décédée. On choisit de représenter ces quatre états par les entiers :

0 (Sain), 1 (Infecté), 2 (Rétabli) et 3 (Décédé).

L'état des cases d'une grille évolue au cours du temps selon des règles simples. On considère un modèle où l'état d'une case à l'instant $t+1$ ne dépend que de son état à l'instant t et de l'état de ses huit cases voisines à l'instant t (une case du bord n'a que cinq cases voisines et trois pour une case d'un coin). Les *règles de transition* sont les suivantes :

— une case décédée reste décédée ;
— une case infectée devient décédée avec une probabilité p_1 ou rétablie avec une probabilité $(1-p_1)$;
— une case rétablie reste rétablie ;
— une case saine devient infectée avec une probabilité p_2 si elle a au moins une case voisine infectée et reste saine sinon.

On initialise toutes les cases dans l'état sain, sauf une case choisie au hasard dans l'état infecté.

1. On a écrit en `Python` la fonction `grille(n)` suivante :

```
1  def grille(n):
2      M=[]
3      for i in range(n):
4          L=[]
5          for j in range(n): L.append(0)
6          M.append(L)
7      return M
```

   Décrire ce que retourne cette fonction.

   On pourra dans la question suivante utiliser la fonction `randrange(p)` de la bibliothèque `random` qui, pour un entier positif p, renvoie un entier choisi aléatoirement entre 0 et $p-1$ inclus.

2. Écrire en `Python` une fonction `init(n)` qui construit une grille G de taille $n \times n$ ne contenant que des cases saines, choisit aléatoirement une des cases et la transforme en case infectée, et enfin renvoie G.

3. Écrire en `Python` une fonction `compte(G)` qui a pour argument une grille G et renvoie la liste `[n0, n1, n2, n3]` formée des nombres de cases dans chacun des quatre états.

   D'après les règles de transition, pour savoir si une case saine peut devenir infectée à l'instant suivant, il faut déterminer si elle est exposée à la maladie, c'est-à-dire si elle possède au moins une case infectée dans son voisinage. Pour cela, on écrit en `Python` la fonction `est_exposee(G, i, j)` suivante.

```
1  def est_exposee(G,i,j):
2      n=len(G)
3      if i==0 and j==0:
4          return (G[0][1]-1)*(G[1][1]-1)*(G[1][0]-1)==0
5      elif i==0 and j==n-1:
6          return (G[0][n-2]-1)*(G[1][n-2]-1)*(G[1][n-1]-1)==0
7      elif i==n-1 and j==0:
8          return (G[n-1][1]-1)*(G[n-2][1]-1)*(G[n-2][0]-1)==0
9      elif i==n-1 and j==n-1:
10         return (G[n-2][n-1]-1)*(G[n-2][n-2]-1)*(G[n-1][n-2]-1)==0
11     elif i==0:
12         # a completer
13     elif i==n-1:
14         return (G[n-1][j-1]-1)*(G[n-1][j+1]-1)*(G[n-2][j-1]-1)*(G[n-2][j]-1)*(G[n-2][j
               +1]-1)==0
15     elif j==0:
16         return (G[i-1][0]-1)*(G[i+1][0]-1)*(G[i-1][1]-1)*(G[i][1]-1)*(G[i+1][1]-1)==0
17     elif j==n-1:
18         return (G[i-1][n-1]-1)*(G[i+1][n-1]-1)*(G[i-1][n-2]-1)*(G[i][n-2]-1)*(G[i+1][n
               -2]-1)==0
19     else:
20         # a completer
```

4. Quel est le type du résultat renvoyé par la fonction `est_exposee` ?
5. Compléter les lignes 12 et 20 de la fonction `est_exposee`.

6. Écrire une fonction `suivant(G, p1, p2)` qui fait évoluer toutes les cases de la grille G à l'aide des règles de transition et renvoie une nouvelle grille correspondant à l'instant suivant. Les arguments `p1` et `p2` sont les probabilités qui interviennent dans les règles de transition pour les cases infectées et les cases saines. On pourra utiliser la fonction `bernoulli(p)` suivante qui simule une variable aléatoire de Bernoulli de paramètre p : `bernoulli(p)` vaut 1 avec la probabilité `p` et 0 avec la probabilité `(1 - p)`.

```
def bernoulli(p):
    x=rd.random()
    if x<=p:
        return 1
    else:
        return 0
```

On reproduit ci-dessous le descriptif de la documentation `Python` concernant la fonction `random` de la bibliothèque `random` :

> `random.random()` : `return the next random floating point number in the range [0.0, 1.0).`

Avec les règles de transition du modèle utilisé, l'état de la grille évolue entre les instants t et $t+1$ tant qu'il existe au moins une case infectée.

7. Écrire en `Python` une fonction `simulation(n, p1, p2)` qui réalise une simulation complète avec une grille de taille $n \times n$ pour les probabilités `p1` et `p2`, et renvoie la liste `[x0, x1, x2, x3]` formée des proportions de cases dans chacun des quatre états à la fin de la simulation (une simulation s'arrête lorsque la grille n'évolue plus).
8. Quelle est la valeur de la proportion des cases infectées `x1` à la fin d'une simulation ? Quelle relation vérifient `x0, x1, x2 et x3` ? Comment obtenir à l'aide des valeurs de `x0, x1, x2 et x3` la valeur `x_atteinte` de la proportion des cases qui ont été atteintes par la maladie pendant une simulation ?

On fixe `p1` à 0,5 et on calcule la moyenne des résultats de plusieurs simulations pour différentes valeurs de `p2`. On obtient la courbe de la figure fig.1.

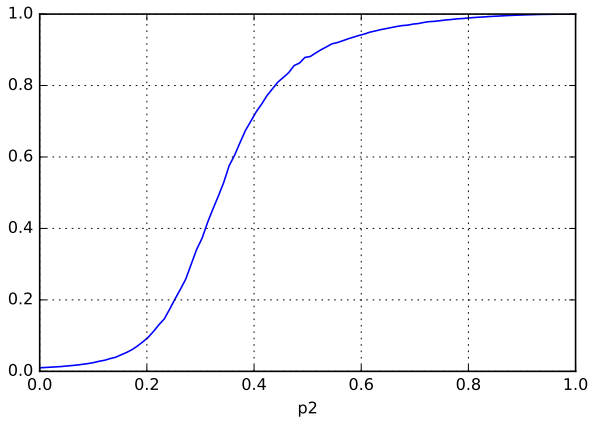


FIGURE 1 – Représentation de la proportion de la population qui a été atteinte par la maladie pendant la simulation en fonction de la probabilité p_2.

9. On appelle *seuil critique de pandémie* la valeur de `p2` à partir de laquelle plus de la moitié de la population a été atteinte par la maladie à la fin de la simulation. On suppose que les valeurs de `p2` et `x_atteinte` utilisées pour tracer la courbe de la figure 1 ont été stockées dans deux listes de même longueur `Lp2` et `Lxa`. Écrire en `Python` une fonction `seuil(Lp2, Lxa)` qui détermine par dichotomie un encadrement `[p2cmin, p2cmax]` du seuil critique de pandémie avec la plus grande précision possible. On supposera que la liste `Lp2` croît de 0 à 1 et que la liste `Lxa` des valeurs correspondantes est croissante.

Pour étudier l'effet d'une campagne de vaccination, on immunise au hasard à l'instant initial une fraction q de la population. On a écrit la fonction `init_vac(n, q)`.

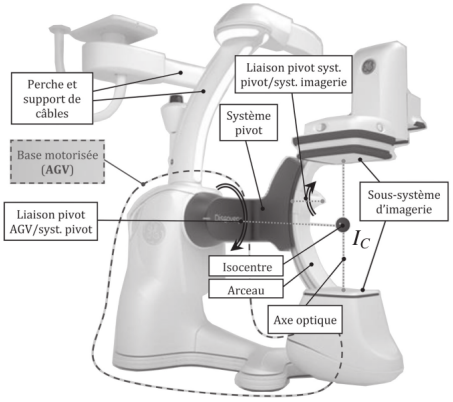
```
1  def init_vac(n,q):
2      G=init(n)
3      nvac=int(q*n**2)
4      k=0
5      while k<nvac:
6          i=rd.randrange(n)
7          j=rd.randrange(n)
8          if G[i][j]==0:
9              G[i][j]==2
10             k+=1
11     return G
```

10. Peut-on supprimer le test en ligne 8 ?
11. Que renvoie l'appel `init_vac(5, 0.2)` ?

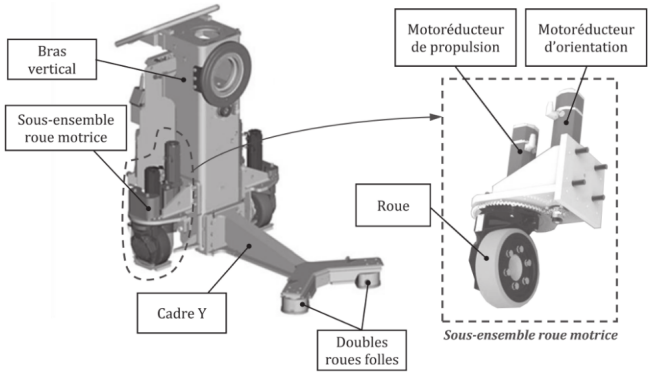
2 Contrôle et prédiction de la position d'un véhicule automatisé

Les questions d'informatique peuvent être traitées en langage Python ou langage Scilab. Il est demandé au candidat de bien préciser sur sa copie le choix du langage et de rédiger l'ensemble de ses réponses dans ce langage. L'annexe page 16 rappelle les principales commandes utiles à la manipulation des vecteurs et matrices. Dans tout le sujet, il sera supposé que les modules et bibliothèques sont déjà importés dans le programme.

On s'intéresse à la localisation de la base motorisée, aussi appelée AGV (pour Automated Guided Vehicle, soit véhicule à guidage automatique) d'un appareil d'imagerie médicale mobile (le Discovery IGS 730, fig.2a).



(a) Composants du Discovery IGS 730



(b) Éléments du sous-système AGV, carter et sous-système d'imagerie enlevés

FIGURE 2

La base motorisée AGV (figure 2b) est constituée :
— d'une structure support, ou châssis, composée du bras vertical et du cadre Y ;
— de deux sous-ensembles roue motrice et motorisation associée (un motoréducteur d'orientation et un motoréducteur de propulsion pour chaque roue) ;
— de deux doubles roues « folles » non motorisées.

Localisation par odométrie. La gestion des déplacements du système au sein de la salle d'intervention (figure 3) nécessite de déterminer la position de la plateforme mobile, repérée par le vecteur position p, tel que :

$$\begin{pmatrix} x(t) \\ y(t) \\ \theta(t) \end{pmatrix}$$

où $x(t)$, $y(t)$ et $\theta(t)$ sont les paramètres de position définis dans le repère global (O, x_0, y_0) associé à la salle d'intervention.

Une première stratégie de localisation a conduit à estimer les paramètres de position par odométrie : partant d'une position initiale connue, la position est actualisée au cours du mouvement en estimant le déplacement

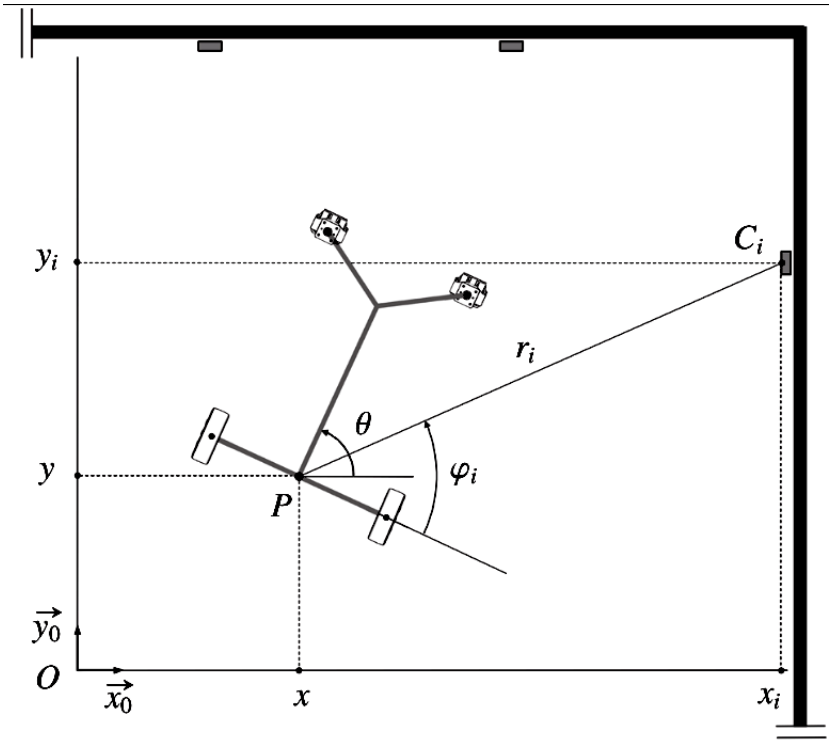


FIGURE 3 – Paramétrage

global du système à partir des mesures des déplacements angulaires des roues motrices délivrées par les codeurs de roue. Pour cette étude, l'orientation des roues motrices par rapport à la plateforme mobile suivant z_0 est supposée fixée, dans la configuration de la figure 3.

Soient $\Delta\theta_D$ et $\Delta\theta_G$ les déplacements angulaires des roues motrices droite et gauche et Δp le vecteur associé au déplacement global de la plateforme entre les instants t et $(t+\Delta t)$, Δt étant la durée d'une période d'échantillonnage séparant deux mesures. Il est admis ici que les composantes $(\Delta x, \Delta y, \Delta\theta)$ du déplacement global Δp peuvent être estimées à partir de $\Delta\theta_D$ et $\Delta\theta_G$ selon :

$$\begin{cases} \Delta x = r\dfrac{\Delta\theta_D + \Delta\theta_G}{2}\cos\left(\theta(t) + r\dfrac{\Delta\theta_D - \Delta\theta_G}{2e}\right) \\ \Delta y = r\dfrac{\Delta\theta_D + \Delta\theta_G}{2}\sin\left(\theta(t) + r\dfrac{\Delta\theta_D - \Delta\theta_G}{2e}\right) \\ \Delta\theta = r\dfrac{\Delta\theta_D + \Delta\theta_G}{e} \end{cases}$$

où r est le rayon d'une roue motrice et e la distance entre les deux roues motrices.

Le vecteur position à l'instant $(t+\Delta t)$ noté p' est alors calculé suivant : $p' = p + \Delta p$.

Le traitement numérique repose sur une fonction `position_odometrie(delta_d,delta_g, p0)` qui prend en arguments :

— `delta_d` et `delta_g`, déplacements angulaires des roues droite $(\Delta\theta_D)$ et gauche $(\Delta\theta_G)$;
— `p0`, tableau de dimension 3 associé à la position à l'instant t (vecteur p).

La fonction renvoie le tableau p1 de dimension 3 associé à la position à l'instant $(t+\Delta t)$ (vecteur p'). Les paramètres géométriques r et e sont déclarés comme variables globales dans le programme principal.

1. Écrire la fonction `position_odometrie(delta_d,delta_g, p0)` renvoyant le tableau `p1`.

Modèle d'évolution de l'incertitude associée à l'estimation de la position

Une approche statistique permet de modéliser l'évolution de l'incertitude sur l'estimation de la position par odométrie au cours du déplacement de l'AGV. En considérant les estimations x, y et θ comme des variables aléatoires continues et sous certaines hypothèses non détaillées ici, l'incertitude associée à l'estimation de la position p peut être quantifiée par un ensemble de paramètres associés aux variables aléatoires telles que la variance et la covariance, constituant la matrice $\Sigma_{p'}$:

$$\Sigma_{p'} = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\theta} \\ \sigma_{zx}^2 & \sigma_{zy} & \sigma_\theta^2 \end{bmatrix}$$

Pour le calcul de ses termes, celle-ci est décomposée suivant :

$$\Sigma_{p'} = \Sigma_1 + \Sigma_2$$

où :
— Σ_1 est une matrice associée aux incertitudes sur l'estimation du déplacement Δp entre t et $t + \Delta t$ à partir des mesures codeurs,
— Σ_2 est une matrice associée aux incertitudes sur l'estimation de la position p à l'instant t, cumulées depuis l'instant initial et qui se propagent à l'instant $t + \Delta t$. Il est admis ici que la matrice Σ_2 est obtenue à partir de la matrice de variance-covariance Σ_p associée à la position p à l'instant t par la relation :

$$\Sigma_2 = J_p \Sigma_p J_p^T$$

la matrice J_p (matrice jacobienne) ayant pour expression :

$$J_p = \begin{bmatrix} 1 & 0 & -r\dfrac{\Delta\theta_D + \Delta\theta_G}{2} \sin\left(\theta(t) + r\dfrac{\Delta\theta_D - \Delta\theta_G}{2e}\right) \\ 0 & 1 & r\dfrac{\Delta\theta_D + \Delta\theta_G}{2} \cos\left(\theta(t) + r\dfrac{\Delta\theta_D - \Delta\theta_G}{2e}\right) \\ 0 & 0 & 1 \end{bmatrix}$$

2. Écrire une fonction `calcul_sigma_2(mat_sigmap,mat_jacob)` qui prend en arguments le tableau `mat_sigmap` de dimensions 3×3 associé à la matrice de variance-covariance Σ_p de la position p à l'instant t, le tableau `mat_jacob` de dimensions 3×3 associé à la matrice jacobienne J_p et renvoie la matrice `mat_sigma2` (Σ_2). On écrira la fonction la plus générique possible et on évaluera sa complexité.

Localisation du système par fusion de données

En pratique, avec une approche basée uniquement sur l'odométrie, l'incertitude associée à l'estimation de position ne fait que croître au cours du déplacement. Une telle approche se révèle donc inadaptée pour satisfaire l'exigence de précision associée à la localisation du système. Une solution classique en robotique mobile consiste à réaliser une estimation de la position à partir de la fusion de mesures obtenues par des capteurs indépendants :
— des capteurs proprioceptifs, qui ont une perception locale du déplacement de la plateforme, les codeurs de roues ici ;
— des capteurs extéroceptifs, qui ont une perception globale du déplacement de la plateforme, le capteur laser ici, qui renvoie les positions (r_i, φ_i) de $n \leq 12$ cibles réfléchissantes C_i dans un repère local attaché à la plateforme (figure 4).
La procédure de localisation dont le principe est présenté en figure 4 se compose de 4 étapes décrites ci-après en adoptant un formalisme discret.

Étape 1 : estimation
Partant d'un rang k associé à un instant t, cette étape consiste à *estimer la position* de la plateforme au rang $k + 1$ (associé à l'instant $t + \Delta t$). L'estimation, notée $\hat{p}(k + 1|k)$ est calculée à partir de $\hat{p}(k|k)$, « meilleure » estimation au rang k obtenue à l'issue de l' itération précédente et de l'intégration des variations angulaires des deux roues motrices $(\Delta\theta_G, \Delta\theta_D)$ mesurées par les codeurs (odométrie). Cette étape conduit également à la détermination de l'incertitude associée à $\hat{p}(k + 1|k)$ (représentée par une matrice $\Sigma_p(k + 1|k)$).

Les traitements numériques sont ici similaires à ceux abordés dans la portant sur la localisation par odométrie (l'unique différence réside sur le fait que l'estimation au rang $k+1$ fait à présent intervenir la « meilleure » estimation obtenue au rang k).

Étape 2 : prédiction
Cette étape consiste à effectuer une prédiction de l'observation associée au laser. Pour chaque cible C_i de coordonnées globales $(x_i\ ,\ y_i)$ répertoriée par la carte, les coordonnées $(r_i,\ \varphi_i)$ dans le repère local attaché à la plateforme, dont la position est celle estimée à l'étape précédente, sont calculées. Un vecteur

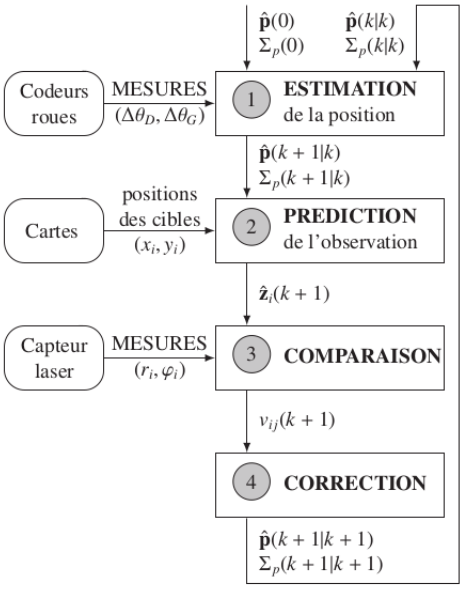


FIGURE 4 – Principe de localisation

$\hat{z}_i(k+1)$ est ainsi formé pour chaque cible, à partir des coordonnées locales calculées. Les coordonnées globales (x_i , y_i) des 12 cibles sont regroupées dans un tableau `cible_map` de dimensions 12×2.

3. On admet qu'une fonction `global_local(xi,yi,p1)` prenant en arguments les coordonnées globales `xi` (x_i), `yi` (y_i) d'une cible i, le tableau `p1` de dimension 3 associé à la position estimée (vecteur $\hat{p}(k+1|k)$) et renvoyant les coordonnées locales (`ri`,`phii`) (r_i, φ_i) de la cible i a déjà été écrite.
   Écrire une fonction `prediction(p1,cible_map)` qui prend en arguments le tableau `p1` de dimension 3, associé à la position estimée ($\hat{p}(k+1|k)$)), le tableau `cible_map` et renvoie le tableau `zpred` de dimensions 12×2, contenant les coordonnées locales (r_i, φ_i) des 12 cibles (12 vecteurs $\hat{z}_i(k+1)$).

Étape 3 : comparaison

Cette étape consiste à réaliser une comparaison entre la prédiction de l'observation et l'observation effective afin de former le vecteur innovation $v_{ij}(k+1) = z_j(k+1) - \hat{z}_i(k+1)$ qui permettra de déterminer la correction à appliquer. Dans un premier temps, lorsque le capteur laser détecte une cible C_j donnée (l'indice j est fixé), de coordonnées locales (r_j, φ_j), il est nécessaire d'établir une correspondance entre la cible C_j détectée et une cible C_{i0} parmi les 12 cibles C_i , dont les positions ont été prédites à l'issue de l'étape 2. Il s'agit, pour une mesure donnée (r_j, φ_j) de rechercher le plus proche voisin dans le tableau `zpred`. Pour cela, un balayage du tableau `zpred` est effectué, au cours duquel est calculée la distance d i j entre la cible C_i et la cible C_j détectée, permettant ainsi d'identifier la cible C_{i0} la plus proche de C_j . Par souci de simplification, la norme euclidienne sera utilisée ici pour le calcul de la distance :

$$d_{ij} = \sqrt{r_i^2 + r_j^2 - 2 r_i r_j \cos(\varphi_j - \varphi_i)}$$

4. Écrire la suite d'instructions de la fonction `comparaison(zj,zpred)` conduisant à identifier dans le tableau z`zpred` les coordonnées `(r_pp,phi_pp)` de la cible C_{i0} plus proche voisin d'une cible C_j détectée par le laser.

Étape 4 : correction

Cette étape consiste à déterminer une correction de l'estimation initiale de position $p(k+1|k)$ à partir des $v_{ij}(k+1)$ permettant le calcul de $p(k+1|k+1)$, « meilleure » estimation au rang $k+1$. L'incertitude associée à $p(k+1|k+1)$ (représentée par la matrice de covariance $\Sigma_p(k+1|k+1)$ est là aussi quantifiée. Cette dernière étape, qui repose sur la mise en œuvre d'un filtre de Kalman, n'est pas abordée ici.