Projet Robotique Mobile

Sommaire :

I) Introduction :

Au cours de ce semestre, nous avons dû réaliser un projet de robotique mobile. Le but était d'implémenter un robot capable de se déplacer d'un point A à un point B en évitant des obstacles, cela se faisant sur le logiciel Matlab.

Tout d'abord, nous avons donc réaliser un simulateur : c'est la création d'un environnement défini et délimité par des bordures, il y a également des obstacles qui sont défini. Puis il a fallu créer le robot, nous avons choisi une forme simple pour celui-ci : un triangle. Il possède alors 3 roues : deux d'entre elles seront motorisées, et une autre permettra d'assurer son équilibre.

Enfin nous avons réalisé la fonction commande : c'est donc cette fonction qui va permettre au robot de se déplacer.

II) Mise en place du cahier des charges :

Le cahier des charges nous a donné quelques paramètres à respecter, mais aussi le prototype de la fonction commande, car lors de l'examen, le robot sera testé dans un simulateur qui lui est inconnu, il faudra donc être cohérent face à ce nouveau simulateur :

Voici une capture d'écran de notre cahier des charges :

```
function commande=controle(Etat, Cible, Capteurs)

Avec les vecteurs sous la forme :

commande=[Wd;Wg]
Etat=[x;y;theta,wd;wg]
Cible=[x;y;theta]
Capteurs=[d1 theta1;d2 theta2;d3 theta3]


Valeurs numériques :
Nmin = 0.05;            % Distance min / à l'arrivée
|theta-angle_cible| <0,03 % décalage angulaire à l'arrivée
|Wmax| = 10;              % Vitesse de rotation max des roues
R = 0.05;               % Rayon des roues
L = 0.4;                % 1/2 entre-axe roue
Dmax = 3 ;              % portée max laser/valeur si pas détection

Tau moteur = 200ms
Dt=0,1 s
theta1=-pi/6     theta2=0   theta3=pi/6
```

Figure 1 : copie d'écran d'une partie de notre cahier des charges.

III) Le simulateur :

a) Création de l'environnement :

Le but est ici de créer un espace délimité par des barrières, ou il y aura des obstacles.

Pour commencer nous avons déjà réinitialiser la figure 1, afin de ne pas être gêner s'il y avait une simulation en mémoire. Puis nous avons défini les coordonnées maximums de la figure 1 : nous avons choisi un carré de 50 unités de côté, afin d'avoir un assez grand environnement pour pouvoir tester le comportement de notre robot.

```
%réinitialisation de la figure 1
clc;
close all
clear
figure(1)
clf
hold on
%choix des coordonnées max:
axis([0 50 0 50])
axis square
```

Figure 2 : les premières lignes de code….

Maintenant il faut créer les obstacles : nous avons fait le choix de créer ces obstacles et ces barrières dans une fonction, ce qui va permettre de simplifier l'écriture de notre fichier principal :

```
function obstacles = obs()
%je définis les bordures
obstacles(1).A = [5 45]; obstacles(1).B = [5 5];
obstacles(2).A = [45 5]; obstacles(2).B = [45 45];
obstacles(3).A = [5 45]; obstacles(3).B = [45 45];
obstacles(4).A = [5 5]; obstacles(4).B = [45 5];

%Je définis les obstacles:
obstacles(5).A = [8 20]; obstacles(5).B = [15 30];
obstacles(6).A = [8 20]; obstacles(6).B = [30 15];
obstacles(7).A = [25 25]; obstacles(7).B = [15 5];
obstacles(8).A = [40 30]; obstacles(8).B = [30 30];
obstacles(9).A = [40 40]; obstacles(9).B = [40 15];
obstacles(10).A = [10 35]; obstacles(10).B = [20 40];
```

Figure 3 : créations des bordures et des barrières

b) Choix de départ et d'arrivée :

Nous devons alors définir deux points : le départ et l'arrivée. Deux choix sont possible : soit nous définissons deux points qui seront fixes dans l'environnement, soit nous laissons l'utilisateur choisir quels seront ces points, il nous faut également définir l'angle de départ…

Nous allons donc réaliser le code de la seconde possibilité : nous allons utiliser la fonction ginput().

```
%Affichage d'obstacles
obstacle=obs();
%choix du point de départ
Dep= ginput(1);
plot(Dep(1), Dep(2), 'go');
%choix angle de départ:
b = ginput(1);
c = b-Dep;
theta=atan2(c(2),c(1));
%Choix de l'arrivée
arrivee = ginput(1);
plot(arrivee(1),arrivee(2),'ro');
```

Figure 4 : code pour laisser le choix à l'utilisateur des points de départ, d'arrivée et de l'angle.

La fonction «ginput()» permet à l'utilisateur de cliquer sur un endroit spécifique de l'environnement, et de mémoriser ces informations.

c) Création du robot :

Il faut alors initialiser le robot. Tout d'abord dans le cahier des charges, on nous a donné quelques informations :

Le robot sera sous la forme unicycle, voici ci-dessous la forme du robot :

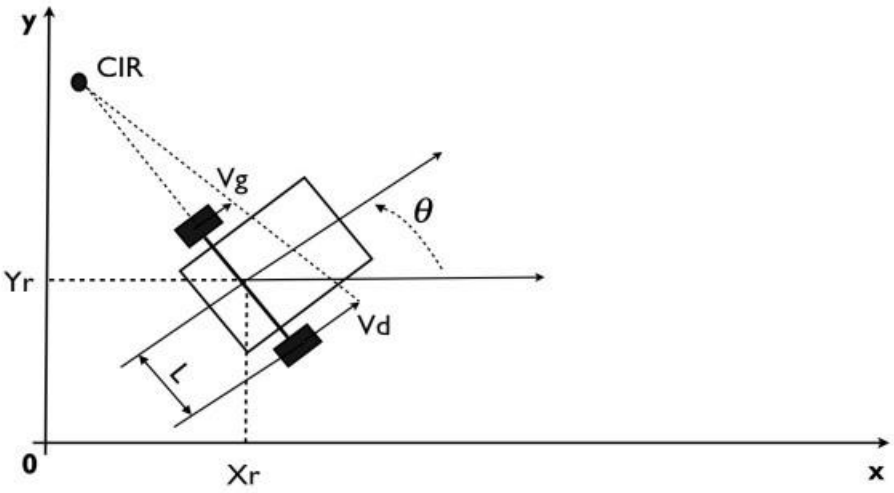

Figure 5 : Robot mobile unicycle

Nous allons définir les coordonnées de trois points qui seront les extrémités du robot, puis nous utiliserons la matrice de transposition pour repasser ces cordonnées dans le repère de l'environnement.

```
%trois points dans le repère du robot
Rob.a=[1;0;1]; Rob.b=[0;-0.5;1]; Rob.c=[0;0.5;1];
%maintenant il faut passer dans le repère absolue:
%définition de la matrice de passage
mat_passage = [cos(theta) -sin(theta) Dep(1) ; sin(theta) cos(theta) Dep(2) ; 0 0 1];
A=mat_passage*Rob.a;
B=mat_passage*Rob.b;
C=mat_passage*Rob.c;
%On trace le robot
rob_points=[A B C]';
faces=[1 2 3];
Rob.ptr=patch('vertices',rob_points,'faces',faces,'facecolor','r');
```

Figure 6 : code permettant de tracer le robot dans l'environnement

d) Les capteurs :

Pour que le robot puisse se déplacer tout en évitant les obstacles, il lui faut alors des capteurs. Le cahier des charges définit alors ces capteurs comme trois segments fixes, de longueur 3 unités, espacés de $\frac{\pi}{6}$.

Nous avons fait le choix de mettre ce code dans une fonction, ce qui a pour but de simplifier l'écriture du fichier principale.

```
function cap = def_capteur(Etat, Rob)
% on va définir les extremités des capteurs
%on va donc aditionner la projection de l'extremité sur l'axe des absice ou
%ordonnée à l'absice et à l'ordonnée du point central du robot
    cap.extrem = [Etat(1)+Rob.Dmax*cos(Etat(3)+Rob.theta1)  Etat(2)+Rob.Dmax*sin(Etat(3)+Rob.theta1);
                  Etat(1)+Rob.Dmax*cos(Etat(3))  Etat(2)+Rob.Dmax*sin(Etat(3));
                  Etat(1)+Rob.Dmax*cos(Etat(3)+Rob.theta3)  Etat(2)+Rob.Dmax*sin(Etat(3)+Rob.theta3)];
    cap.ptr1 = line([2 0], [0 0]);
    cap.ptr2 = line([2 0], [0 0]);
    cap.ptr3 = line([2 0], [0 0]);


end
```

Figure 7 : code permettant de créer nos capteurs.

IV) Le robot :

a) Détection des obstacles :

L'objectif de ce projet est d'arriver au niveau du point d'arrivée en évitant tous les obstacles qui se trouvent sur son chemin.

Nous allons donc créer une fonction qui va tester si les capteurs ont croisé un obstacle.

Pour ce faire, nous allons réaliser le calcul suivant :

Un segment dans le plan a deux extrémités : R et P, celui-ci est donc l'ensemble des points M décrit par :

$$\begin{cases} x_M = x_R + (x_P - x_R)s \\ y_M = y_R + (y_P - y_R)s \end{cases}$$

Ou s est un paramètre compris entre 0 et 1 inclus.

S'il y a une intersection entre le segment AB et le segment RP, cela va se traduire par cette équation matricielle :

$$[P-R \quad A-B]\begin{bmatrix} s \\ t \end{bmatrix} = A - R$$

Il faudra ensuite vérifier l'inversibilité de M = [P-R A-B] : si M n'est pas inversible, il n'y a pas d'intersection… Puis il faut calculer u = inverse(M) x (A -R), vérifier si u(1) et u(2) sont compris entre 0 et 1. Une fois tout ceci vérifier, notre point d'intersection sera donnée par R+s(P-R).

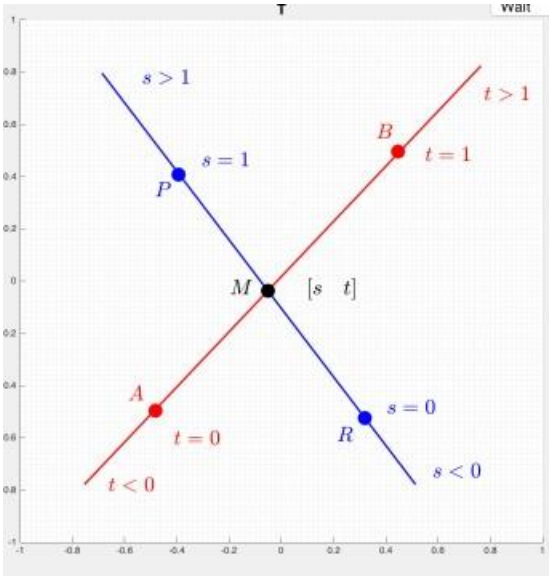

Figure 7 : intersection entre PR et AB

Voici ci-dessous, le code de notre fonction intersect(), qui a pour but de déterminer si il y a oui ou non une intersection entre les capteurs et un obstacle quelconque :

```
function [distance_capteur_obs, point_intersec] = intersect(obstacles,capteur,Etat,Rob)
% on initialise la distance par defaut à 10
distance_capteur_obs = 10; point_intersec=[0 0];

% boucle pour de 1 à 10 pour chaque obstacle
for i=1:10
    %on cherche ici à écrire M = A + t x (B -A)
    %matriciellement cela s'écrit
    %[P ? R A ? B]x [s ; t] = A ? R
    %P et R désignent les extremités de l'obstacle
    %A et B c'est les extremité du capteur
    % le but ici est de connaitre les coordonnées du point d'intersection
    % entre le laser et l'obstacle
    P = obstacles(i).B;
    R = obstacles(i).A;
    A = [Etat(1) Etat(2)];
    B = capteur;
    Mobstacle = [P(1)-R(1) ; P(2)-R(2)];
    Mcapteur = [A(1)-B(1) ; A(2)-B(2)];
    M = [Mobstacle Mcapteur];
    %tout d'abord il faut vérifier l'inversibiité de M (via le déterminant)
    if abs(det(M)) < 0.0001
        %si on est dans ce cas, matrice M non inversible => pas d'intersection
    else
        %cas d'existence d'intersection
        % il faut résoudre U = inverse(M) x (A -R)
        u = M\[A(1)-R(1);A(2)-R(2)];
        % on a s = u(1) et t = u(2)
        %il faut vérifier que s et t sont dans [0 1]
        if (u(1)>0 && u(1)<1) && (u(2)>0 && u(2)<1)
            %cas OK
            s = u(1);
            t = u(2);
            distance_capteur_obs = Rob.Dmax*t;
            %les coordonnées du point d'intersec sont donnée par R+s(P-R)
            point_intersec = [R(1)+s*(P(1)-R(1)) R(2)+s*(P(2)-R(2))];
        end
    end
end
```

Figure 8 : code de la fonction intersect

b) L'asservissement du robot :

La fonction Ode45() est une fonction du langage Matlab qui permet de résoudre des équations différentielles, cette fonction va donc nous être utile afin d'ajuster les coordonnées du robot et l'état de celui-ci.

Voici le code qui nous a permis de faire l'asservissement du robot :

```
while commande(1)~=0 && commande(2)~=0
    %tant que la vitesse du robot n'est pas nulle...
    %intégration de l'état actuel
    [T,X]= ode45(@(t, Etat)modele(Etat,commande,Rob)', [0 0.1], Etat);
    L45 = length(T);  %il n'y a que la dernière ligne qui nous interressent
    Etat = X(L45, :);
    %mide à jour de la figure 1:
    capteurs= mise_a_jour(Etat, Rob, capteurs);
    %detection d'obstacles
    for i=1:3 %pour chaque capteurs
        [dist_capteur,pt_inter] = intersect(obstacle,capteurs.extrem(i,:),Etat,Rob);
        if dist_capteur>0 && dist_capteur<Rob.Dmax
            %cas ou le laser ne permet pas de détecter d'obstacle
            capteurs.extrem(i,1)=pt_inter(1);
            capteurs.extrem(i,2)=pt_inter(2);
        end
        detection_capteur(i,1)=dist_capteur;
    end
    %controle des commandes en fonction de la cible, l'état et les obstacles
    commande = controle(Etat,Cible,detection_capteur);

end
```

Figure 10 : boucle d'asservissement du robot

c) La partie commande :

Cette fonction commande va traiter les informations venant de l'environnement et des capteurs afin de commander le robot. Dans notre cahier des charges, le prototype de cette fonction nous était donné par :

```
function commande=controle(Etat, Cible, Capteurs)
```

Tout d'abord, lorsque le robot s'approche de la cible d'arrivée, la vitesse de celui-ci va ralentir : on va considérer que l'on est proche de la cible si la distance entre le robot et la cible est de l'ordre de dmax.

La sensibilité des capteurs va être diminué si l'arrivée est trop proche d'un obstacle ou d'un bordure car sinon il lui sera impossible de s'approcher de l'arrivée.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% réglage vitesse/ sensibilité suivant l'approche de la cible %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cas 1: cible très proche: la cible est dans le rayon du capteur
if distance_cible < dmax
    wmax = 0.8*Robwmax; %on ralentit
    if capteurs(3,1) >= distance_cible*1.5 || distance(2)>=distance_cible*1.5 || distance(1) >= distance_cible*1.5
    %si la cible est proche d'un obstacle
        %il faut réduire la sensibilité des capteurs
        sensibilite = 0.2*sensibilite;
    end
else
    %pas de changement au niveau de la vitesse ni de la sensibilité
    wmax = Robwmax;
end
```

Figure 11 : réglage de la vitesse et de la sensibilité à l'approche de la cible

Une fois le robot sur le point d'arrivée, il faut que celui-ci tourne jusqu'à ce qu'il est l'angle d'arrivée π définit dans le code (dans notre code, nous avions choisi un angle de $\frac{\pi}{2}$, que l'on peut modifier facilement en allant dans la fonction principale « robot.m »).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               Rotation pour angle point d'arrivée               %%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% test: a t'on atteint la cible (prise en compte de l'erreur de position)
if (Etat(1) > Cible(1)-position_err && Etat(1) < Cible(1)+position_err) && (Etat(2) > Cible(2)-position_err && Etat(2) < Cible(2) + position_err)
    % delta_angle est l'angle au quel le robot doit tourner
    delta_angle = Etat(3)-Cible(3);
    %cas 1: delta_angle proche de 0
    if abs(delta_angle) < angle_err
        % on arrete nos deux moteurs
        wd = 0;
        wg = 0;
    % sinon il faut tourner pour l'atteindre
    elseif delta_angle > 0
        wd = -0.5*wmax;
        wg = 0.5*wmax;
    else
        wd = 0.5*wmax;
        wg = -0.5*wmax;
    end
```

Figure 12 : code du réglage de l'angle à l'arrivée

Lors du trajet, les capteurs ne détectent pas d'obstacles, le robot se dirige directement sur la cible. Sinon le robot va alors modifier sa trajectoire en effectuant des virages pour éviter les obstacles qui posent problèmes.

V) Conclusion :

Lors de ce projet, nous avons appris à utiliser le logiciel et langage Matlab afin de créer un simulateur et un robot virtuel capable de se déplacer en évitant certains obstacles. Nous avons par ailleurs découverts plusieurs fonctions très intéressantes et relativement pratique tel que ginput() et Ode45(). La construction d'un projet complet avec des regroupements de fonctions sous des scripts différents était tout nouveau pour nous et a bien enrichi notre panel d'outils sur Matlab.