École Universitaire Polytechnique de Montpellier
Microélectronique et Automatique - Électronique et Informatique Industrielle

6 juin 2015

Rapport de robotique mobile Hovercraft

TERRIE Corentin - BELLOC Thomas
CROS Bastien - ABLAJ Yassine
SZAFIR Michael - BOUKHALFA Yannis
MONNIER Matthias

MEA

Table des matières

1 Introduction

1.1 Répartition des tâches

2 Étude de fonctionnement général d'un aéroglisseur

3 Simulateur

3.1 Modèle

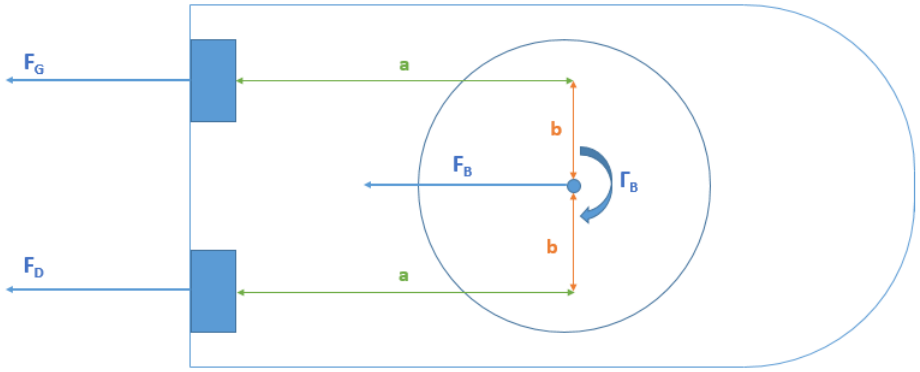

FIGURE 1 – Schéma de l'hovercraft.

3.2 Implémentation

3.2.1 Schéma de commande

Le schéma global utilisé pour le simulateur est le suivant :

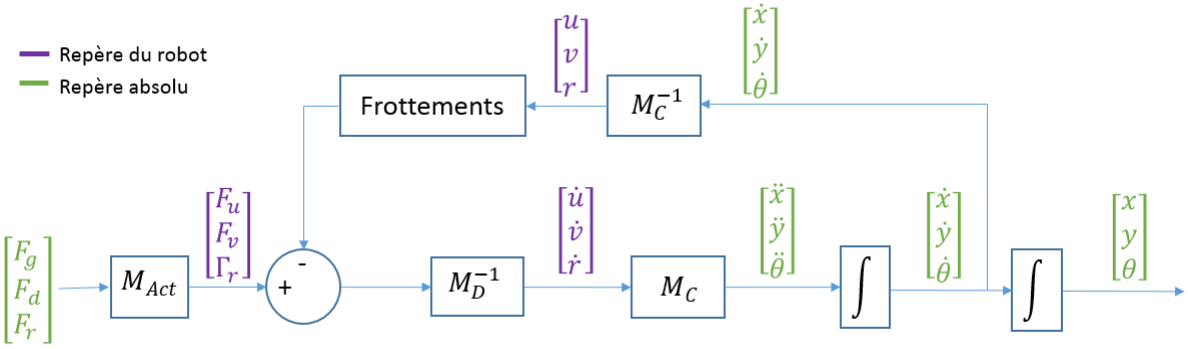

FIGURE 2 – Schéma général de notre simulateur.

On commande notre système en choisissant la force appliquée par chaque moteur. Fr est la force de sustentation. F_g et F_d sont les moteurs droit et gauche.

On obtient en sortie de notre simulateur la position et l'orientation de notre aéroglisseur dans le plan (x, y).

3.2.2 Matrice d'actionnement

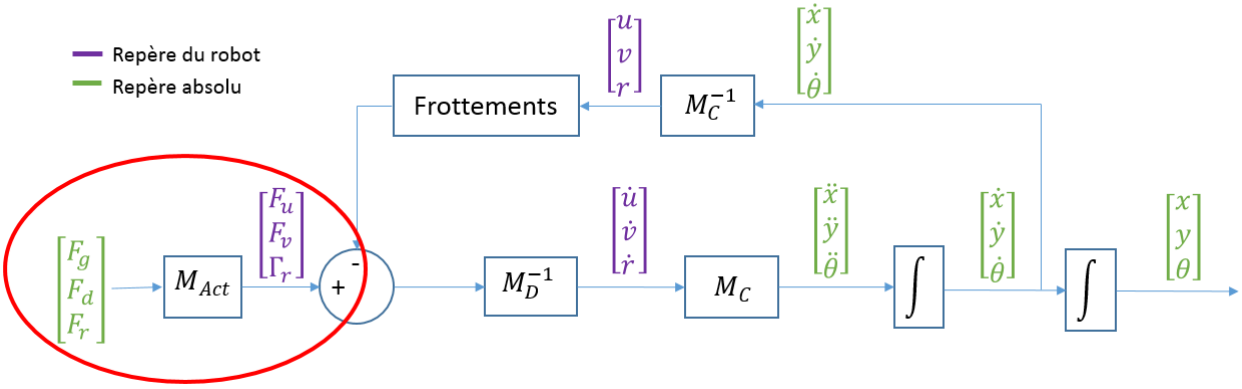

FIGURE 3 – Matrice d'actionnement.

Pour passer des forces dans le repère absolu à celles dans le repère du robot, on passe par la matrice d'actionnement que l'on a tiré des équations du modèle. En sortie de notre matrice d'actionnement on récupère F_u et F_v qui sont respectivement les forces d'avance et de dérive, et Γ_r qui est le couple induit par la force de sustentation (par l'hélice) et les forces des deux moteurs (voir. Notre simulateur travaille donc dans le plan, on ne prend pas en compte la légère élévation de l'aéroglisseur qui permet de s'affranchir des frottements.

Voici le code de notre matrice d'actionnement :

```
function MatActionnement = MAct(Vect_F)
global d k
Ma = [  1    1    0;
        0    0    0;
        -d   d    k];

MatActionnement = (Ma * Vect_F);
end
```

FIGURE 4 – Fonction de notre matrice d'actionnement.

On remarque que l'on n'a pas de contrôle sur la force de dérive au vu de la disposition des actionneurs sur notre aéroglisseur. L'équation que l'on a utilisée pour faire cette matrice est la suivante :

$$\begin{cases} F_B = F_G + F_D \\ F_v = 0 \\ \Gamma_B = F_D * b - F_G * d + k * \omega_r \end{cases} \quad (1)$$

3.2.3 Modèle dynamique inverse

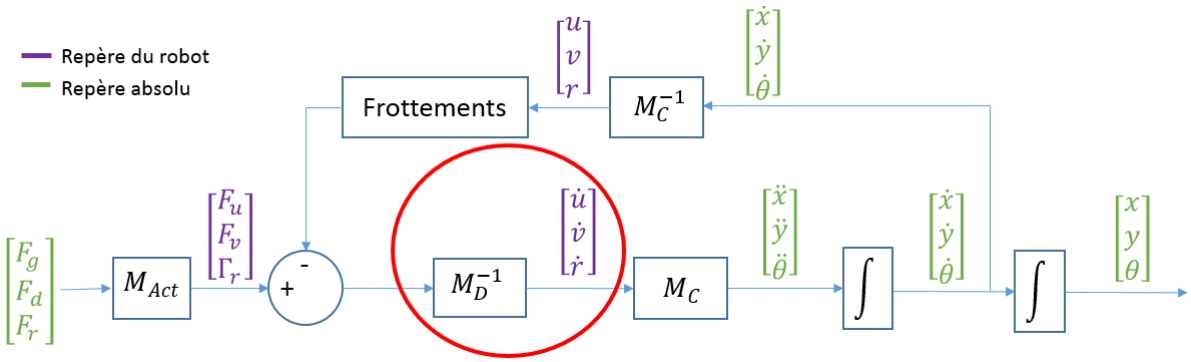

FIGURE 5 – Matrice dynamique inverse.

Pour obtenir les accélérations dans le repère du robot, on utilise le modèle dynamique inverse qui est implémenté comme ceci :

```
function MDynaInv = MdInv(Vect)
global m_u m_v m_r

Md = [m_u   0     0;
      0     m_v   0;
      0     0     m_r];

Md_inv = inv(Md);

MDynaInv = Md_inv * Vect ;

end
```

FIGURE 6 – Fonction de notre modèle dynamique inverse.

3.2.4 Modèle cinématique

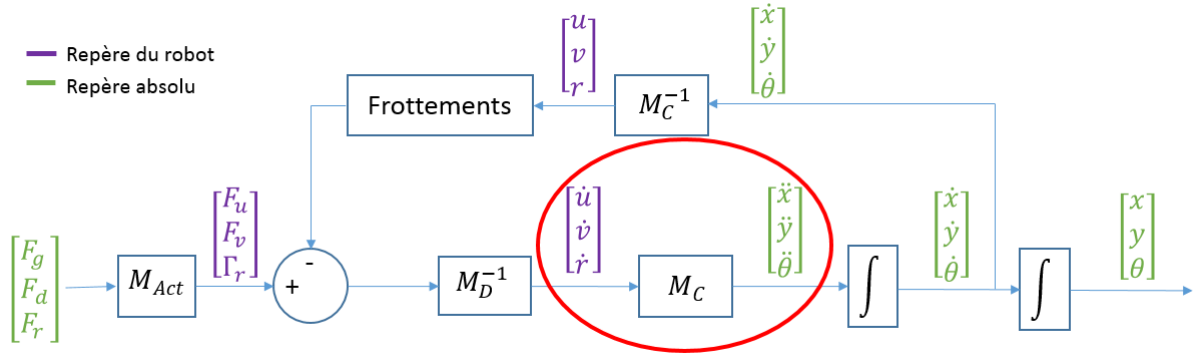

FIGURE 7 – Matrice dynamique inverse.

On repasse aux accélérations dans le repère absolu grâce au modèle cinématique.

```
function ModCine = Mc(Acc_rob,angle)

Mc = [cos(angle) -sin(angle) 0;
      sin(angle) cos(angle) 0;
      0 0 1];

ModCine = Mc * Acc_rob;

end
```

FIGURE 8 – Fonction de notre modèle cinématique.

Cette fonction prend l'accélération dans le repère du robot et l'angle θ pour avoir la matrice de rotation actualisée.

3.2.5 Fonction d'intégration

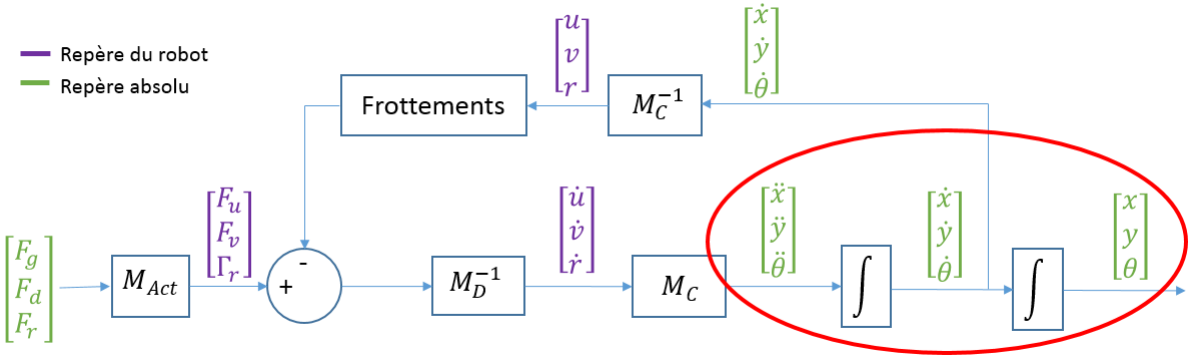

FIGURE 9 – Double intégration.

Dans cette partie, il y a une double intégration. On est donc en présence d'un système d'équations différentielles du deuxième ordre. En utilisant ode45 on peut la résoudre en récupérant les vitesses et les positions dans le repère absolu. Voici comment nous avons scindé nos équations dans la fonction appelée par ode45 :

```
% On scinde l'equa diff du second ordre :
dq = zeros(6,1);
dq(1) = Vect_acc_abs(1);     % x_pp = Vect_acc_abs(1)
dq(2) = q(1);                % x_p = q(1)
dq(3) = Vect_acc_abs(2);     % y_pp = Vect_acc_abs(2)
dq(4) = q(3);                % y_p = q(3)
dq(5) = Vect_acc_abs(3);     % theta_pp = Vect_acc_abs(3)
dq(6) = q(5);                % theta_p = q(5)
```

FIGURE 10 – Code où l'on scinde l'équation différentielle du second ordre.

Ceci n'est pas toute la fonction d'intégration, la fonction entière sera détaillée plus loin.

3.2.6 Modèle cinématique inverse

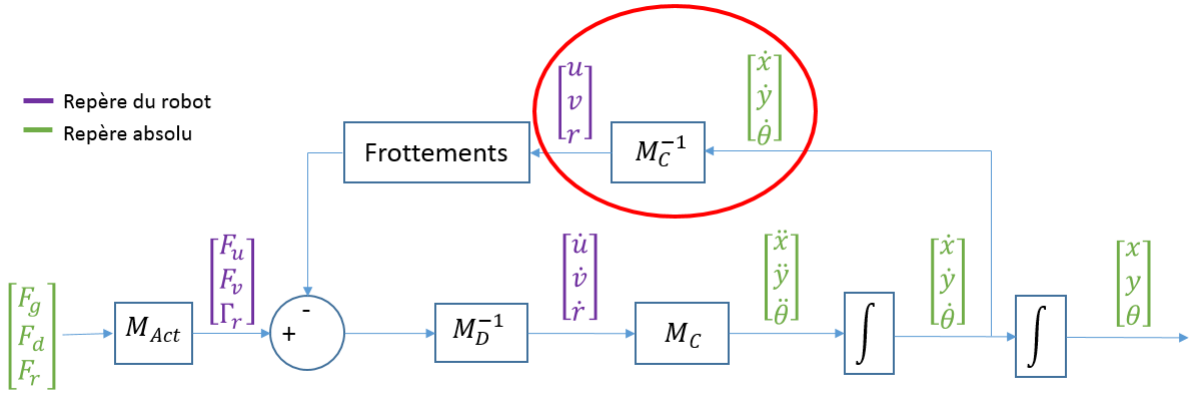

FIGURE 11 – Matrice cinématique inverse.

Nous avons choisi de passer par le modèle cinématique inverse pour revenir à nos vitesses dans le plan du robot. Celles-ci sont utilisées pour calculer les frottements subis par notre robot. Voici le code correspondant :

```
function ModCineInv = McInv(V_abs, angle)

Mc = [cos(angle)     -sin(angle)      0;
      sin(angle)     cos(angle)       0;
      0              0                1];

ModCineInv = Mc\V_abs;

end
```

FIGURE 12 – Fonction de notre modèle cinématique inverse.

3.2.7 Boucle globale

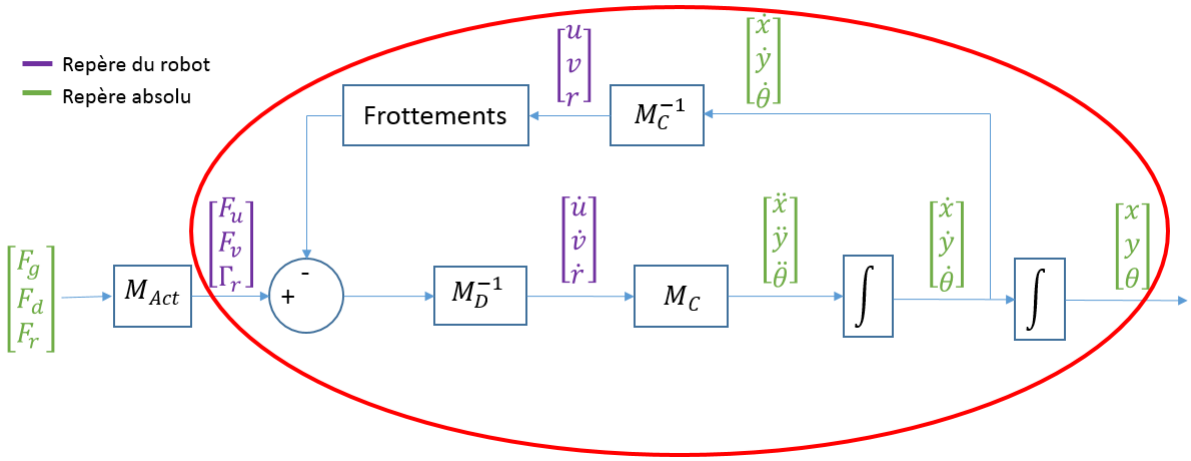

FIGURE 13 – Boucle globale.

Notre programme est constitué d'une boucle for qui représente la mise à jour de l'état de notre robot. Elle se compose des étapes classiques d'un simulateur :

- État de départ (x, y, theta, dx, dy, dtheta)
- Dessin
- Calcul de la commande
- ODE (résolution du système d'équations différentielles du second ordre)
- Extraction de l'état suivant (qui sera l'état de départ de la prochaine itération de la boucle)

```
for T=0:dt:tmax*5
    count = count+1;
    %--------- Etat x y t xd yd td ----------%
    q = [   Vect_v_abs(1);      % x_p
            Vect_pos_abs(1);    % x
            Vect_v_abs(2);      % y_p
            Vect_pos_abs(2);    % y
            Vect_v_abs(3);      % theta_p
            Vect_pos_abs(3)     % theta
            ];
    plot_hover(q, Vect_F, 0)

    % dessin
    %----------------- Dessin ----------------%
    set(ptr,'xdata',Vect_pos_abs(1),'ydata',Vect_pos_abs(2))
    plot(x,y);
    drawnow

    %------------ Calcul commande -----------%

    % ex: test de la fronde

    %----------------- ODE  ----------------%
    % On resout l'equa diff du second ordre
    % pour obtenir la vitesse du robot dans le plan absolu
    [t45, q45] = ode45(@(t,q) Acc2Vit2Pos(t, q, Fb),[0 dt], q);

    % On stock la trajectoire dans q_stock
    q_stock(:, count)=q;

    %------------- Etat = q45 --------------%
    Vect_v_abs(1) = q45(length(q45),1); % x_p
    Vect_v_abs(2) = q45(length(q45),3); % y_p
    Vect_v_abs(3) = q45(length(q45),5); % theta_p

    Vect_pos_abs(1) = q45(length(q45),2); % x
    Vect_pos_abs(2) = q45(length(q45),4); % y
    Vect_pos_abs(3) = q45(length(q45),6); % theta

    %------ x = etat(1)... td=etat(6) ------%
    x = Vect_pos_abs(1);
    y = Vect_pos_abs(2);
    theta = Vect_pos_abs(3);
end
```

FIGURE 14 – Code de notre boucle globale.

Voici la fonction appelée par ode45 :

```
function [dq] = Acc2Vit2Pos(t, q, Fb)
    % q(1) = x_p      q(2) = x
    % q(3) = y_p      q(4) = y
    % q(5) = theta_p  q(6) = theta

    Vect_v_abs = zeros(3,1);
    Vect_v_abs(1) = q(1); % x_p = q(1)
    Vect_v_abs(2) = q(3); % y_p = q(3)
    Vect_v_abs(3) = q(5); % theta_p = q(5)

    %----- CHAINE DE RETOUR -----%
    % On réactualise le modele cinematique et on calcul (u, v, r):
    Vect_v_rob = McInv(Vect_v_abs,q(6));

    Frottements = Frottement(Vect_v_rob);
    % Frottements = Frottement(Vect_v_abs); % on aurait également pu
    % appliquer les frottement dans le repère absolu

    %------- SOUSTRACTEUR -------%
    % YUR=[0 Yur*Vect_v_rob(1)*Vect_v_rob(3) 0]';
    Diff = (Fb - Frottements);%+YUR;

    % Calcul de l'accélération dans le plan du robot (u_p, v_p, r_p)
    Vect_acc_rob = MdInv(Diff);

    % Calcul de l'accélation dans le repère absolu (x_pp, y_pp, theta_pp)
    Vect_acc_abs = Mc(Vect_acc_rob, q(6));

    % On scinde l'equa diff du second ordre :
    dq = zeros(6,1);
    dq(1) = Vect_acc_abs(1);    % x_pp = Vect_acc_abs(1)
    dq(2) = q(1);               % x_p = q(1)
    dq(3) = Vect_acc_abs(2);    % y_pp = Vect_acc_abs(2)
    dq(4) = q(3);               % y_p = q(3)
    dq(5) = Vect_acc_abs(3);    % theta_pp = Vect_acc_abs(3)
    dq(6) = q(5);               % theta_p = q(5)
```

FIGURE 15 – Code de notre boucle globale.

On voit qu'à chaque appel de cette fonction par ode on met à jour les variables d'états nécessaires à l'intégration (frottements, orientation, etc.).

3.2.8 Fonction d'affichage de l'hovercraft

Nous avons créé une fonction pour afficher l'hovercraft :

```
function [  ] = plot_hover(q, Vect_F, mode)
```

Elle prend le vecteur d'état, le vecteur des forces (forces de chaque moteur) ainsi que le mode. Le mode est égal à 1 si c'est le premier appel de la fonction plot_hover. Cela permet de créer et d'initialiser les objets graphiques. Lorsque le mode est différent de 1, ces objets sont simplement actualisés avec les bonnes valeurs.

Voici le résultat d'affichage :

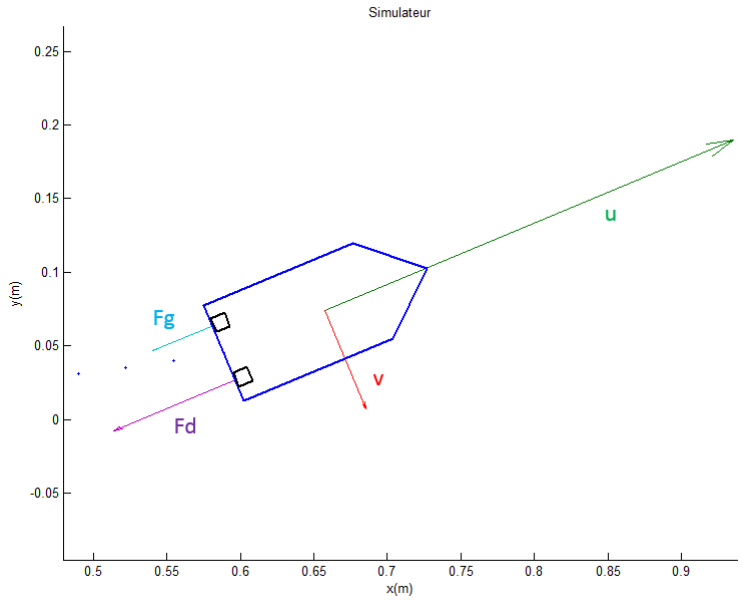

FIGURE 16 – Affichage de l'hovercraft.

F_g et F_d sont les forces de chaque moteurs, u est la force d'avance et v et la force de dérive.

3.2.9 Résultats

Voici le résultat obtenu pour une force du moteur gauche égale à la moitié de celle du moteur droit :

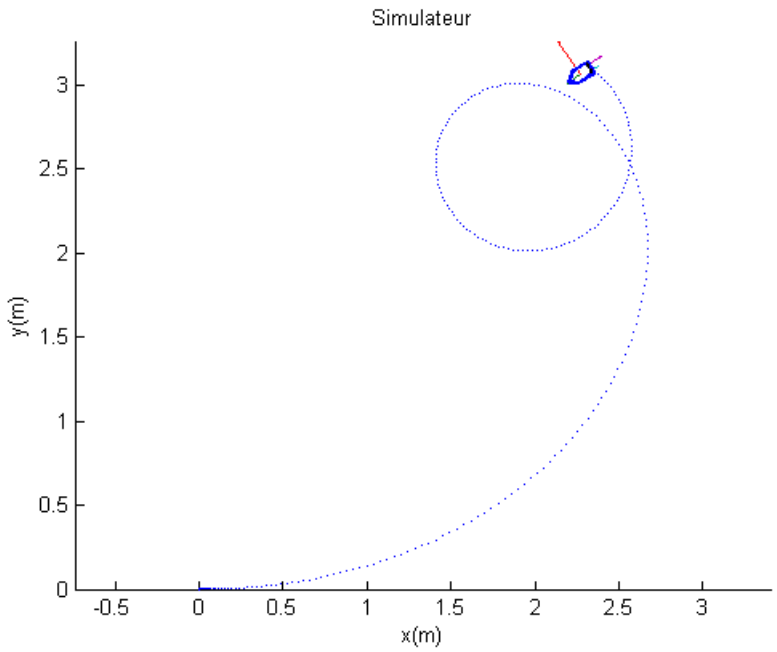

FIGURE 17 – Résulat pour $F_g = 0.5 * F_d$.

3.3 Validation (test de la fronde)

Pour valider notre simulateur, nous avons réalisé le test de la fronde. Pour vérifier que notre système est valide, il faut vérifier que lorsque l'on coupe les moteurs au cours d'un mouvement circulaire, le système doit suivre une trajectoire rectiligne. Voici comment nous l'avons implémenté (dans l'étape du calcul de la commande de notre boucle principale) :

```
% Test de la fronde :
if(T>10 && T<10.2)
    x3 = q(2);
    y3 = q(4);
end
if(T>=10.2)
    Vect_F =[   -0.05;
                0.05;
                0];
    Fb = MAct(Vect_F);
end
```

FIGURE 18 – Code du test de la fronde.

Et voici le résultat obtenu :

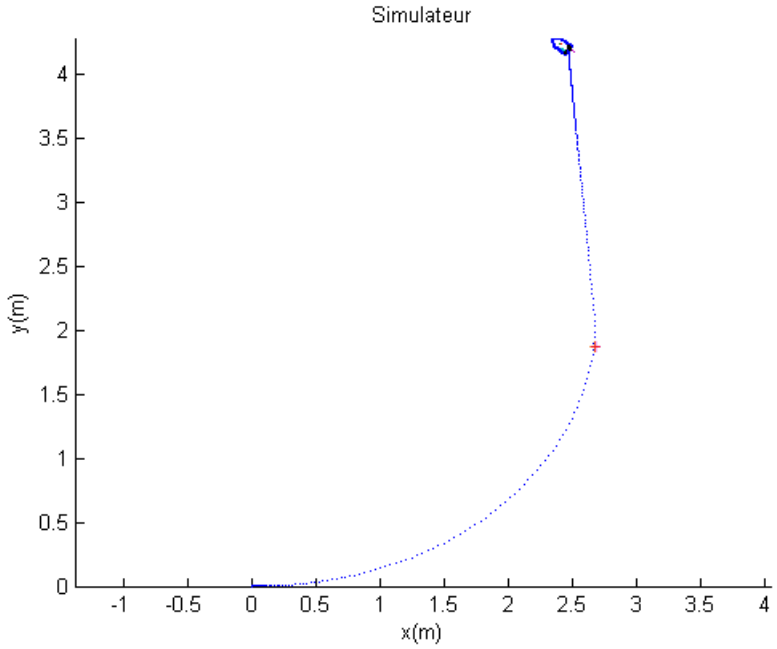

FIGURE 19 – Résultat du test de la fronde.

Le résultat du test valide le simulateur.

4 Mécanique

5 Programmation

6 Conclusion