PénELoP : Péniche d'Expédition Longue Portée

Auteur: 2LAT

Je tiens à remercier les viewers de ma chaîne Twitch (https://www.twitch.tv/le_labo_a_trucs) pour leur aide dans la réflexion sur le projet.

Ce document complète la présentation du projet réalisée sur YouTube :

https://www.youtube.com/@Lelabo%C3%A0trucs

Objectif :

Le projet est un robot péniche modèle réduit autonome pour le sondage du fond des canaux. L'intérêt est de créer un robot guidé par GPS qui peut mesurer la profondeur sur une portion donnée dans l'objectif de réaliser des cartographies du fond.

Table des matières

Objectif : 1
Réalisation : 3
Structure mécanique 3
Dimensionnement : 3
Electronique embarqué 4
Motorisation et commande 4
Mesures et analyse 12
Annexe 1 : Courbes de simulation du contrôle 13

Réalisation :

Structure mécanique

L'ensemble de la structure est réalisée en 3D sur FreeCAD et les fichiers sont disponibles sur le GitHub : https://github.com/Lelaboatrucs/PenELoP

Cette structure est ensuite imprimée en PLA classique via une imprimante 3D.

Pour des contraintes de volume d'impression, la coque est divisée en plusieurs modules collés avec une colle pour canalisation en PVC (résistante à l'eau).

Dimensionnement :

Le choix de la taille du robot a été déterminé de manière à être réalisé avec mon imprimante 3D, mais également en fonction de la motorisation que j'avais à ma disposition. En effet, je disposais de l'électronique d'une véhicule télécommandé à l'échelle 1/16 avec une vitesse de pointe à 24km/h annoncée.

Figure 1 : modèle du véhicule radiocommandé utilisé comme base pour la motorisation du robot péniche

Cette contrainte m'impose que le robot ne soit pas trop grand, afin que la motorisation soit suffisante.

Une petite péniche mesure environ 10 m de long pour 3 mètres de large. Au format 1/16 cela donnerait 62,5 cm de long pour 18 cm de large. Pour garder de la marge, notre version fait 50 cm de long pour 12 cm de large, ce qui conserve à peu près le ratio longueur/largeur. (grosse hypothèse ici : à même échelle, une voiture à la même puissance qu'une petite péniche)

Electronique embarqué

Motorisation et commande

Par simplicité, j'ai choisi d'utiliser une motorisation du véhicule télécommandée. Il me permet d'avoir un ensemble moteur (pour la propulsion), servomoteur (pour la direction), et l'électronique précablé et paramétré (pour l'alimentation électrique). Je précise que cette dernière partie me servira uniquement pour les tests, afin de valiser le bon fonctionnement de l'hélice, du gouvernail et du comportement sur l'eau du bateau avant l'automatisation de la péniche.

Figure 2 : image premier essai du design

Une fois la validation effectuée, il faut passer à l'automatisation. Coté motorisation, cela implique de changer le servo moteur par un nouveau plus facile à commander avec Arduino et la commande moteur pour la propulsion se fait avec un pont en H pour choisir une tension (qui agit sur la vitesse et le sens de rotation). Il reste à remplacer les fonctions relatives à la vision et à la prise de décision du pilote.

Autonomisation

Remplacement des fonctions du pilote :

Pour remplacer les fonctionnalités apportées par le pilote. J'utilise un capteur GPS pour vérifier que le robot est bien en permanence dans la portion à mesurer. CE module à également la possibilité d'être utilisé en compas, ce qui est utile pour les demi-tours qui seront effectués pour balayer la zone à sonder.

Figure 3 : module GPS type BN 880

Un capteur Lidar est employé pour mesurer la distance avec le bord et les éventuels obstacles à 360 degrés.

Figure 4 : capteur Lidar pour la mesure des distances à 360 degrés : RPLidar A1 Slamtec

La communication est assurée avec des modules LoRa Reyax RYLR890/ RYLR896 pour la transmission des données collectées à plusieurs kilomètres (10km à 15km en théorie).

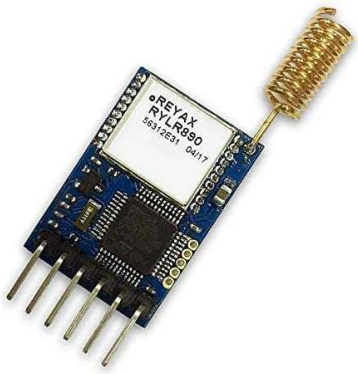


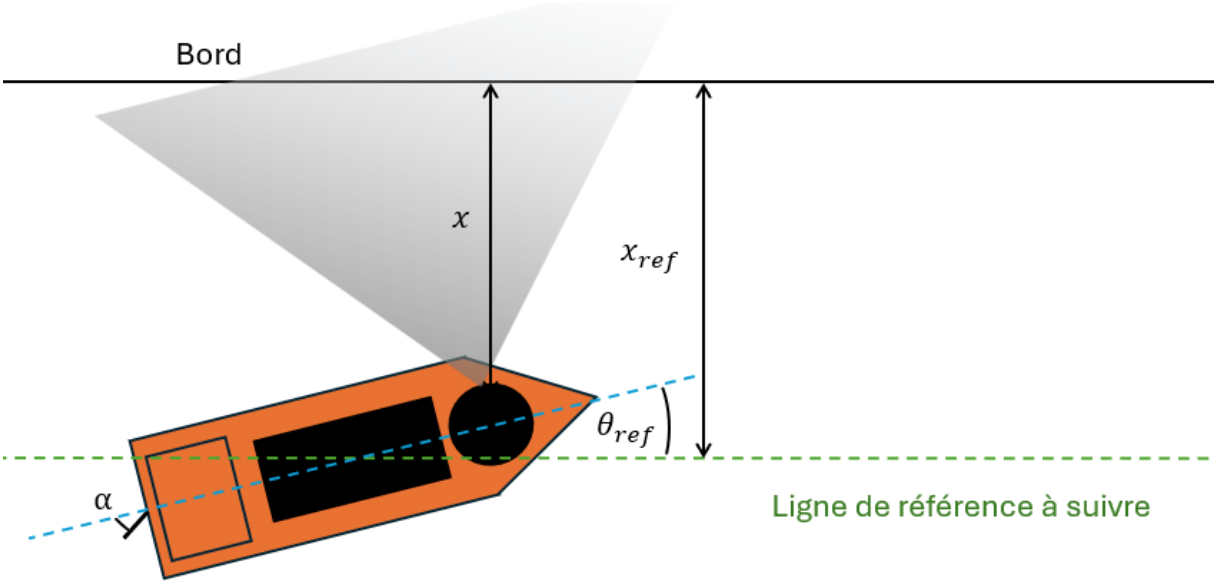
Figure 5 : Module de transmission Lora Reyax 890

L'ensemble est piloté via une carte programmable Arduino Mega. Programme également disponible sur GitHub

Principe de fonctionnement de la commande de pilotage (⚠ trigger warning Maths) :

Le fonctionnement est découpé en 4 modes. Ces 4 modes se baseront sur un principe de suivi de bord comme détaillé ci-dessous :

Le GPS n'est pas suffisamment précis pour servir au guidage. Il faut donc un repère plus fiable.



Pour assurer que le robot reste bien à une distance d du bord, on applique la loi de commande qui garantit la stabilité vis-à-vis des perturbations, on applique donc la théorie du contrôle (sur un modèle physique linéaire simplifié) :

Construction du model physique :

On part du principe que le robot fonctionne à vitesse constante prédéfinie à l'avance.

A vitesse constante, on a :

$$m\frac{dv_{croisière}}{dt} = 0 = F + \gamma v_{croisière}$$

$$F = \gamma v_{croisière}$$

On a donc la dynamique suivante :

$$(1)\begin{cases} \dot{x} = v_{croisière} \times \sin(\theta) \\ \dot{\theta} = k \times v_{croisière} \times sin(2\alpha) \end{cases}$$

k est un coefficient qui lie la vitesse d'avance à la vitesse de rotation du bateau. Il prend également en compte la part de la propulsion qui permet la rotation du bateau. Etant donné la configuration du gouvernail, on sait que moins de 50% de la propulsion est utilisée pour la rotation. On peut calculer γ en mesurant la force nécessaire pour maintenir le monocoque à une position dans un courant constant à vitesse connue. La force peut être obtenue en mesurant l'élongation d'un ressort connu.

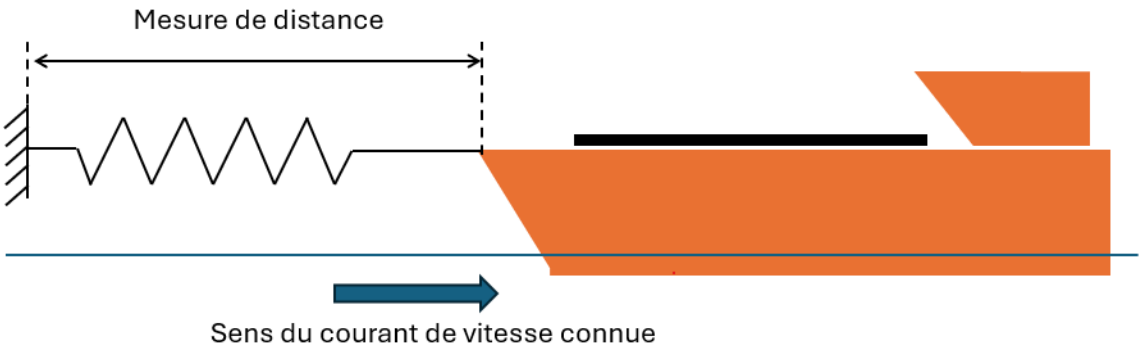


Figure 6 : schéma explétif de l'essais d'identification du coefficient de trainé dans l'eau

Pour notre système, on obtient pour une vitesse de courant de 1.25 m/s une force de 1.63N soit γ=1.3 N/(m/s).

Figure 7 : image de l'essai d'identification du coefficient de trainé dans l'eau

On peut donc en déduire que le bateau d'une masse en charge de 2kg atteindra sa vitesse de croisière de 0.5m/s suivant la courbe comme suit :

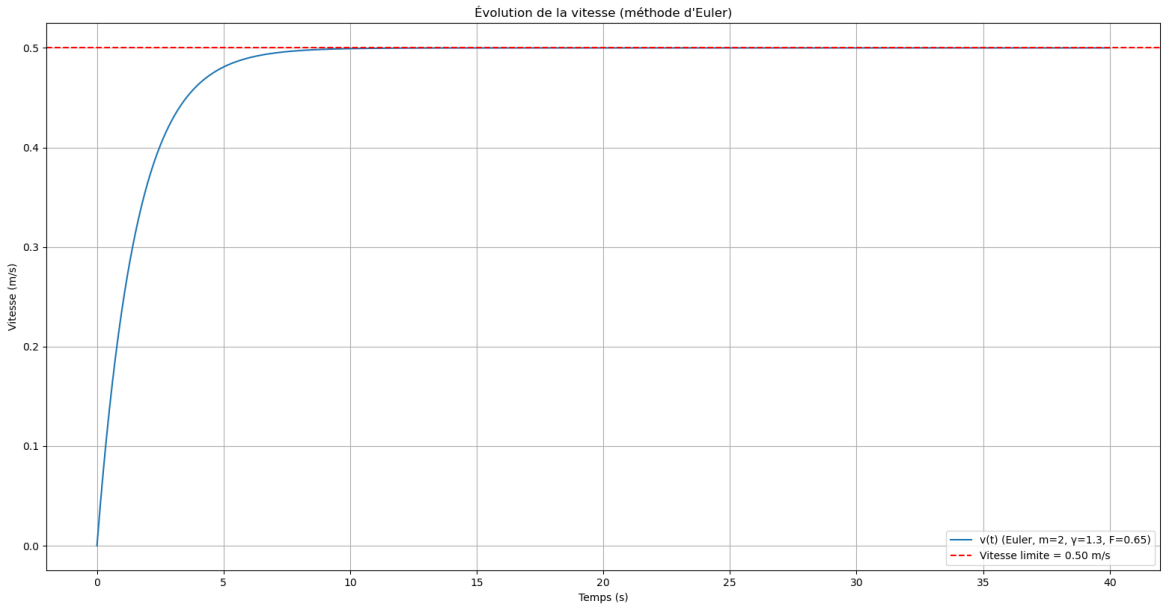


Figure 8 : courbe de la vitesse en fonction du temps (théorique)

C'est courbe est le tracé de la vitesse en fonction du temps solution de :

$$m\frac{dv}{dt} = \gamma v_{croisière} - \gamma v$$

On en déduit le temps nécessaire d'environ 10s avant de lancer le calcul de la commande en condition optimale. (temps pour que la vitesse atteigne la vitesse de croisière.

On linéarise le modèle (1) autour de $\theta = 0$ et $2\alpha = 0$ et on le présente sous forma matricielle.

$$\begin{bmatrix}\dot{x}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}0 & V\\ 0 & 0\end{bmatrix}\begin{bmatrix}x\\ \theta\end{bmatrix} + \begin{bmatrix}0\\ 2kV\end{bmatrix}\alpha$$

$$\begin{bmatrix}0 & V\\ 0 & 0\end{bmatrix} = A$$

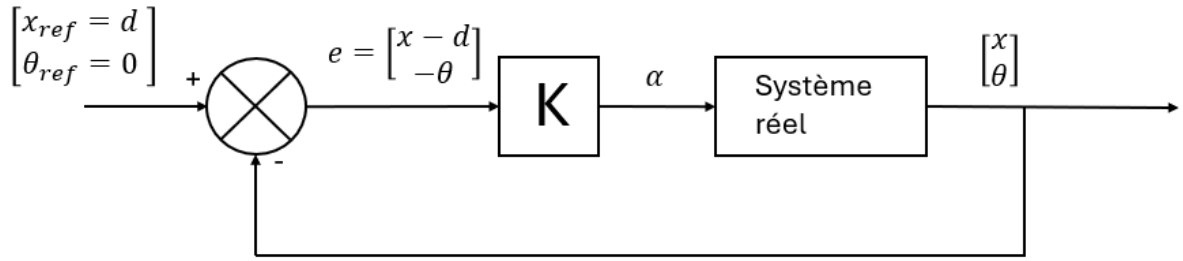
$$\begin{bmatrix}0\\ 2kV\end{bmatrix} = B$$

Ce modèle est très approximatif mais il donne une bonne idée de la structure du model et de l'ordre de grandeur du comportement du système et de la commande à appliquer dans un premier temps avant de corriger sur le tas.

Pour vérifier si le modèle est commandable (c'est-à-dire si l'on peut définir une commande α qui permet d'atteindre $\begin{bmatrix}x\\ \theta\end{bmatrix}$ que l'on souhaite en un temps finit) on calcul $C = [B\ AB] = \begin{bmatrix}0 & 2kV^2\\ 2kV & 0\end{bmatrix}$ et l'on vérifie que les deux lignes soient linéairement indépendantes (donc que l'on ne peut pas écrire la première en fonction de le deuxième).

On peut maintenant s'attaquer à la méthode de calcul de α, pour ça on commence avec une partie proportionnelle.

On a donc le schéma de commande suivant :



Il reste à calculer K pour stabiliser le système et imposer la vitesse de convergence vers la référence (dans la limite du maintien de la stabilité).

Pour cela on cherche K qui permet d'imposer les pôles de (A-BK) à partie réelle strictement négative. Les pôles donnent une information sur la vitesse de convergence du système (dans la limite des possibilités physiques). La partie réelle des pôles est inversement proportionnelle à la constante de temps du système τ et donc au temps de réponse t_r car pour un premier ordre, on a $t_r \approx 5\tau$

Des pôles à parties réelles égales à 0.5 donne un temps de réponse autour de 10s, ce qui semble raisonnable pour définir la valeur initiale du gain K. Les essais réels permettrons d'améliorer les performances dynamiques. Une faible partie imaginaire (autour de ± 0.1 de chaque pole permet de limiter les oscillations autour de la référence.

On peut simuler notre commande dans les conditions idéals pour vérifier le bon fonctionnement. On sature tout de même l'angle du safran pour que la linéarisation soit toujours valable avec moins de 10% d'écart.

Courbe(sinus theta=thetha)

Comme on voit, on doit limiter θ et 2α à $\pm 45°$. Ici, le plus contraignant est surtout α qui doit être compris entre -22.5° et 22.5°

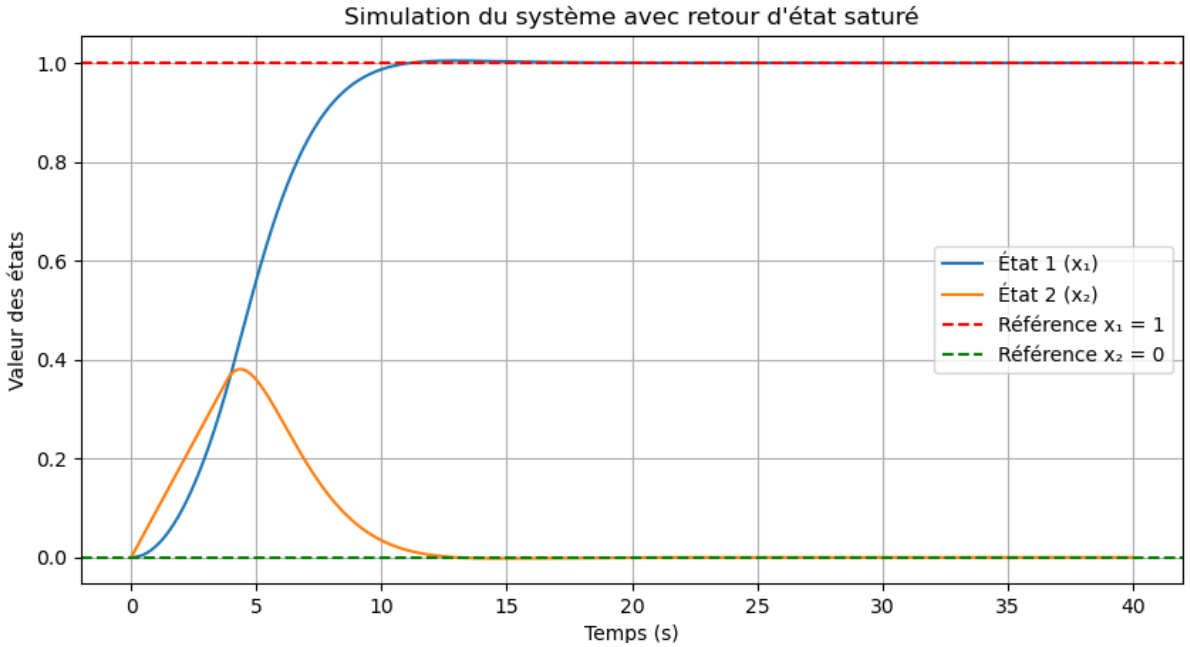


Figure 9 : graphique des états dans le cadre d'une réponse à l'échelon d'amplitude 1 sur la distance (état 1) et l'angle du bateau (état 2) garde une consigne à 0

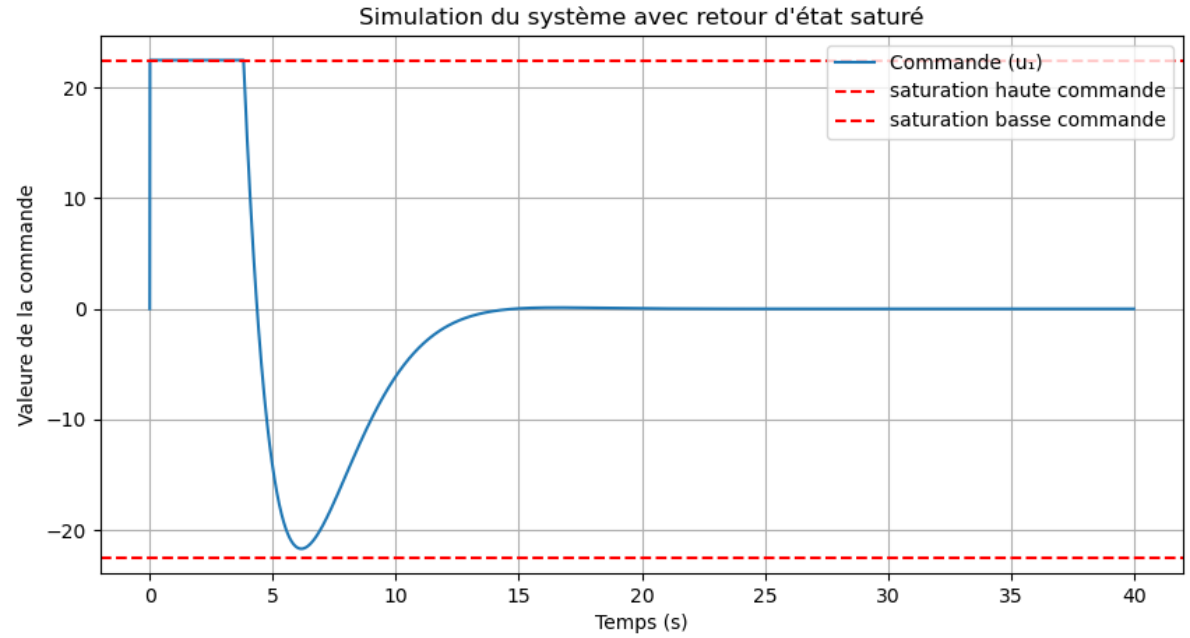


Figure 10 : graphique de la commande saturée en fonction du temps

On voit que les saturations des angles nous empêchent d'atteindre l'objectif en 10s, mais plutôt en 12s

En réalité, il y a de l'imprécision sur chacun des coefficients du model linéaire, et la commande pourra s'actualiser toutes des 0.2s environ. En prenant en compte cela, on vérifie si la commande est robuste (reste stable même si les paramètres changent par rapport à ceux qui servent à calculer K. On essaie alors :

$$A_{reel} = A \times 2 \; et \; B_{reel} = 2 \times B$$

$$A_{reel} = A \times 0.5 \; et \; B_{reel} = 0.5 \times B$$

$$A_{reel} = A \times 2 \; et \; B_{reel} = 0.5 \times B$$

$$A_{reel} = A \times 0.5 \; et \; B_{reel} = 2 \times B$$

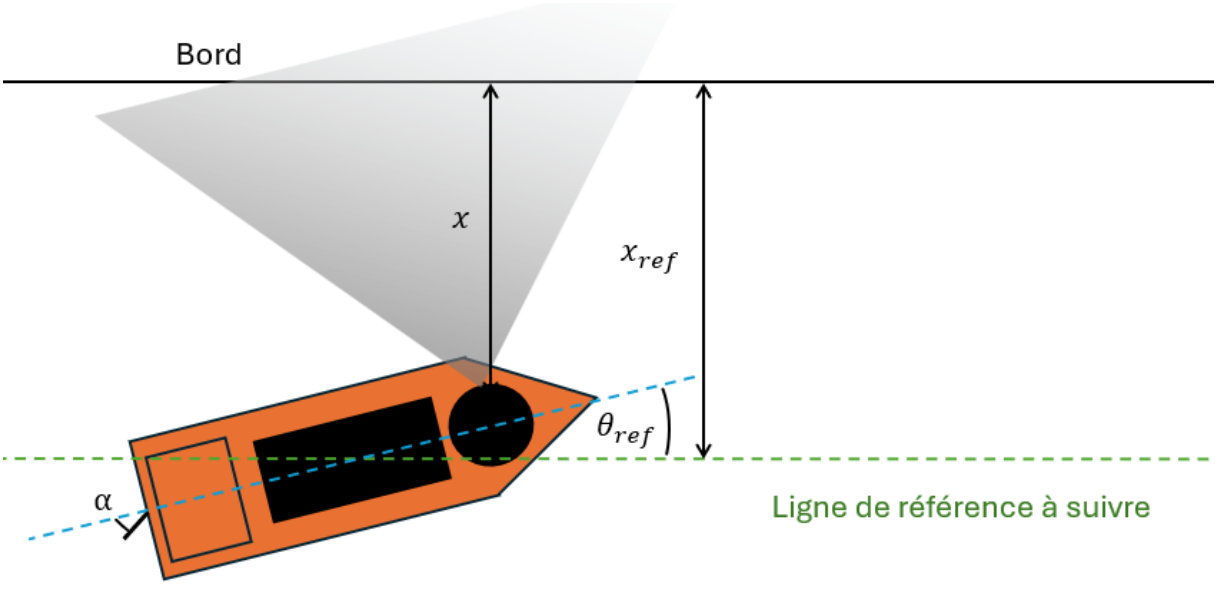
Les courbes sont visibles Annexe 1 : Courbes de simulation du contrôle

En voyant que ça marche dans ces 4 configurations, on peut être plutôt serein sur la robustesse du gain et donc sur la stabilité de la commande avec notre proposition de K pour les premiers essais.

Maintenant que l'on a un control dans lequel on peu avoir confiance, il est temps de détailler les différents modes de fonctionnement. Chaque mode s'appuiera sur les référence en angle et en distance (θ_{ref} et x_{ref}) pour définir les modes de fonctionnement. Puisque notre contrôle semble fonctionner, chacun des modes ne devrait poser aucun problème.

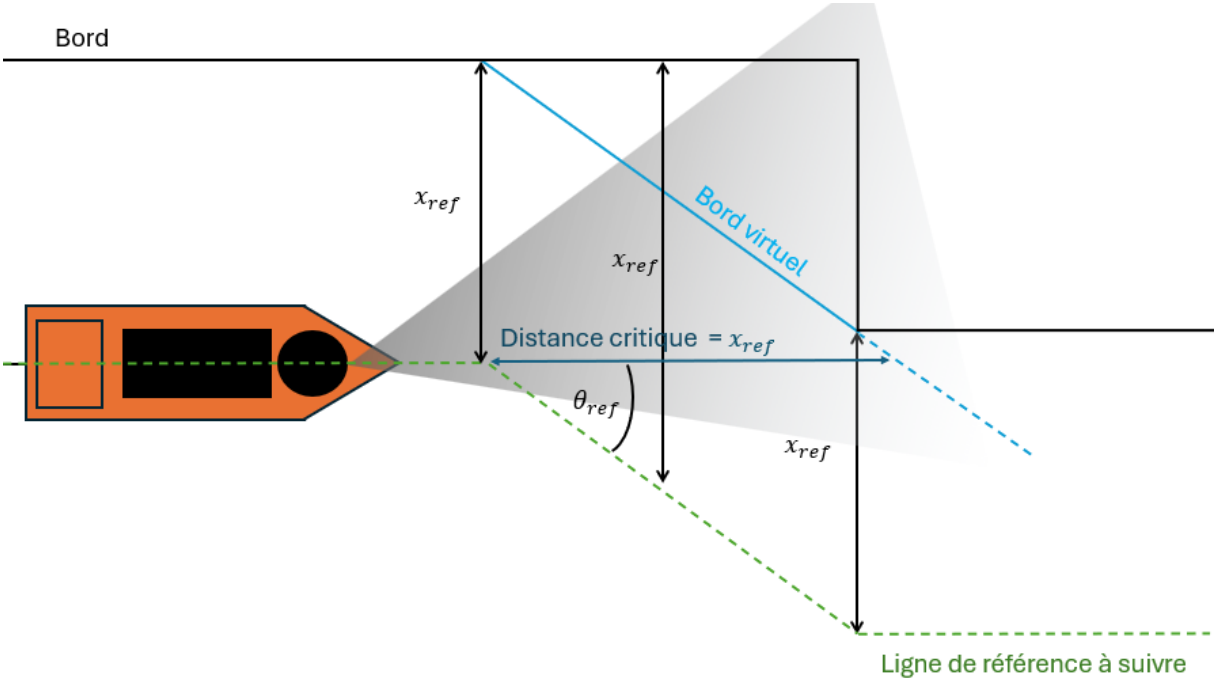
Mode 1 : on est dans la zone GPS prédéfinie et il n'y a pas d'obstacle

Le premiers cas correspond au fonctionnement nominal qui devrait représenter la majorité du fonctionnement du système. Il existe sous deux forme : Le suivi à droite et le suivi à gauche du bord. Seul le coté pour la mesure de x et θ diffère entre ces deux cas. Le reste est inchangé. Il correspond exactement au simulation précédentes ou x_{ref} peu varier pour balayer toutes la largeur du canal.



Mode 2 : on est dans la zone GPS prédéfinie et en présence d'obstacle

Si l'obstacle est entre le bateau et le bord, la consigne reste inchangée par rapport au **Mode 1**. En revanche dans le cas où l'obstacle est devant, on change θ_{ref} et x_{ref} comme suit :



Si l'obstacle est de l'autre côté du bateau, on garde la même référence que celle initiale dans la limite d'une distance avec l'obstacle de 1 m qui prime par rapport à la distance du bord.

Mode 3 : On est en dehors de la zone GPS prédéfinie et il n'y a pas d'obstacle

Ce mode correspond à un demi-tours complet, cela correspond à imposer θ_{ref}montant de 0 à 180°.

Mode 4 : on est en dehors de la zone GPS prédéfinie en présence d'obstacle

En cas d'obstacle durant le demi-tour : aucune idée de comment faire yet XD

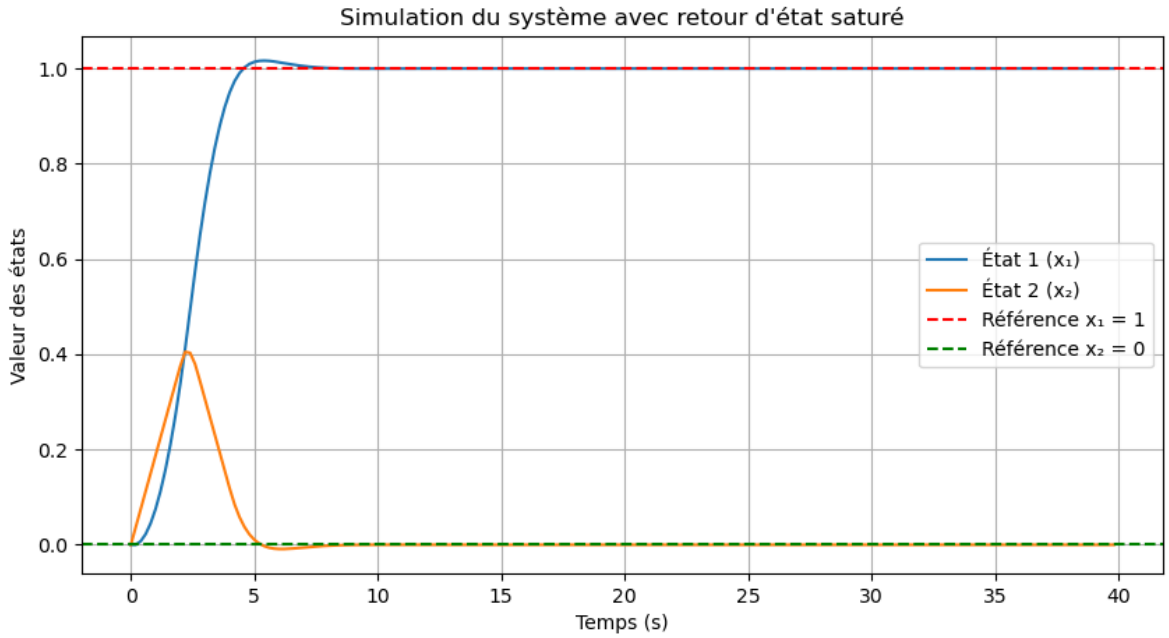
Un fois les 4 modes définis, on ajoute la mesure par GPS et la transmission des infos pour compléter le tout.

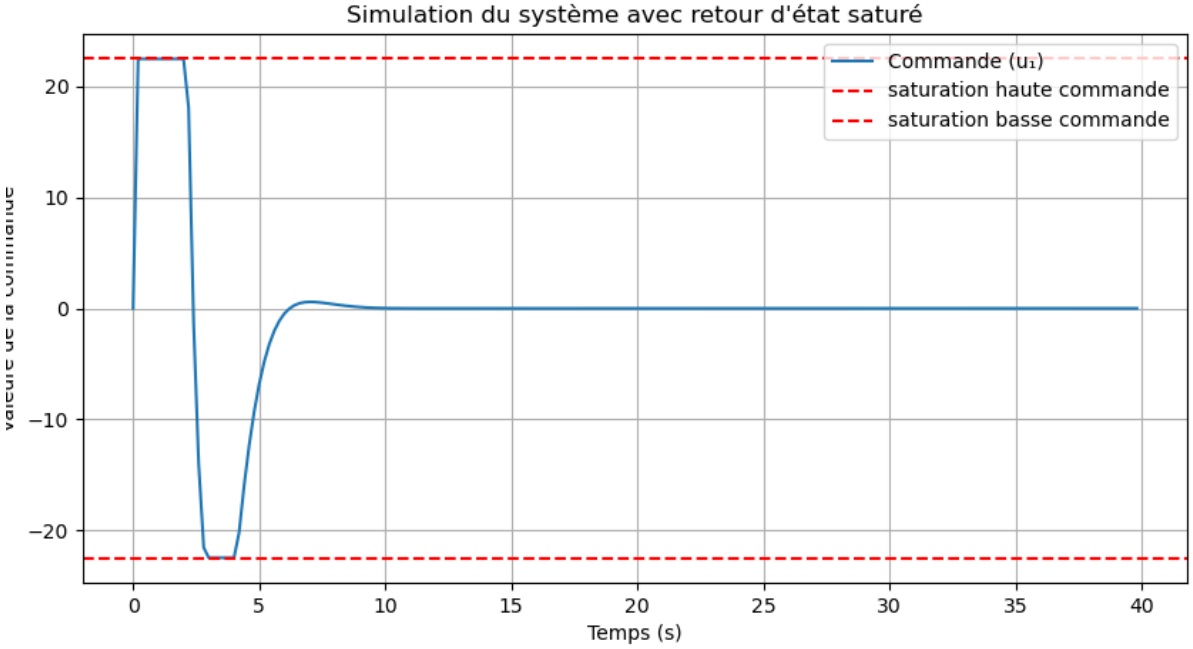
Le programme complet (ou la version actuelle à minima) est disponible en Annexe 2 : Code C++

Mesures et analyse

Annexe 1 : Courbes de simulation du contrôle

$$A_{reel} = A \times 2 \; et \; B_{reel} = 2 \times B$$





Annexe 2 : Code C++

```
#include <RPLidar.h>

#define RPLIDAR_MOTOR 3 // The PWM pin for control the speed of RPLIDAR's motor (MOTOCTRL).

RPLidar lidar;

float minDistance = 100000;
```

```
float angleAtMinDist = 0;
float xRef=1000;
float thetaRef=270;
float Kex=0.18;
float Ketheta=4;
float ex= 0;
float etheta= 0;
float alpha= 0;
float prevalpha= 0;
// sercomoteur
#include <Servo.h>
Servo myServo;
int entalpha = (int)alpha+90;

void setup() {
 Serial.begin(115200);
 Serial1.begin(115200);  // For RPLidar
 lidar.begin(Serial1);
 pinMode(RPLIDAR_MOTOR, OUTPUT);  // set pin modes

 myServo.attach(9);
myServo.write(entalpha);
}

void loop() {
 if (IS_OK(lidar.waitPoint())) {
   //perform data processing here...
   float distance = lidar.getCurrentPoint().distance;
   float angle = lidar.getCurrentPoint().angle;  // 0-360 deg

   if (lidar.getCurrentPoint().startBit) {
     // a new scan, display the previous data...
     printData2(angleAtMinDist, minDistance);
     minDistance = 100000;
     angleAtMinDist = 0;
```

```
   } else {
     if ( distance > 0 &&  distance < minDistance) {
      if ( angle > 225 &&  angle < 315 ){
       minDistance = distance;
       angleAtMinDist = angle;
       }
     }
   }
  }
  else {
   analogWrite(RPLIDAR_MOTOR, 255); //stop the rplidar motor
   // Try to detect RPLIDAR
   rplidar_response_device_info_t info;
   if (IS_OK(lidar.getDeviceInfo(info, 100))) {
     // Detected
     lidar.startScan();
     analogWrite(RPLIDAR_MOTOR, 255);
     delay(1000);
   }
  }
}
void printData(float angle, float distance)
{
  Serial.print("dist: ");
  Serial.print(distance);
  Serial.print("  angle: ");
  Serial.println(angle);
}
void printData2(float angle, float distance)
{
  // calcul erreurs
ex= xRef-distance;
etheta=thetaRef-angle;
  // calcul alpha
alpha=Kex*ex-Ketheta*etheta;
```

```
//saturation alpha
if (alpha>=90){
  alpha=90;
}
if (alpha<=-90){
  alpha=-90;
}
//filtrage alpha
alpha=0.2*alpha+0.8*prevalpha;
prevalpha=alpha;
entalpha = ((int)alpha)+90;
  Serial.print("dist: ");
  Serial.print(distance);
  Serial.print("  angle: ");
  Serial.print(angle);
  Serial.print("  ex: ");
  Serial.print(ex);
  Serial.print("  etheta: ");
  Serial.print(etheta);
  Serial.print("  alpha: ");
  Serial.println(alpha);
  myServo.write(entalpha);
}
```