Super Twisting Implementation for Quadcopter control

Réalisé par :
Kinani Abdelmoughit
Taher Anas

Encadré par :
Mr. Taleb Mohammed

Table des matières

Introduction :

Un quadrirotor est un giravion, un aéronef à voilure tournante comportant quatre rotors pour assurer sa portance. Les rotors sont généralement placés aux extrémités d'une croix. Afin d'éviter à l'appareil de tourner sur lui-même sur son axe de lacet, il est nécessaire que deux hélices tournent dans un sens et les deux autres dans l'autre sens. Pour pouvoir diriger l'appareil, il est nécessaire que chaque couple d'hélices tournant dans le même sens soit placé aux extrémités opposées d'une branche de la croix sur un châssis en « X » ou un châssis en « + ».

Les méthodes de contrôle nécessitent des informations précises à partir des mesures de position et d'attitude effectuées avec un gyroscope, un accéléromètre et d'autres appareils de mesure, tels que GPS, sonar et capteurs laser.

Ce rapport présente la conception et le contrôle de position d'une modélisation simplifié d'un Quadcopter comme une barre soutenue dont le mouvement angulaire est contrôlé par un couple des hélices à l'extrémité de la barre. Après la modélisation du système, des équations et des paramètres dynamiques sont étudiés, pour ce faire, nous utiliserons Simscape Multibody. L'algorithme de contrôle Super-twisting sera appliqué au modèle avec différents paramètres et les résultats obtenus sont comparés.

Modélisation du système :

Dynamique du système

Selon la 2ème loi de Newton de rotation :

$$J\ddot{\theta} = u + d(t)$$

Où u est le couple de contrôle

Et d(t) est la perturbation et les incertitudes sur le modèle.

Si on pose $x = (\theta, \dot{\theta})$:

On obtient l'équation suivante $\dot{x} = Ax + Bu + \begin{pmatrix} 0 \\ \frac{d(t)}{J} \end{pmatrix}$

Où $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ et $B = \begin{pmatrix} 0 \\ \frac{1}{J} \end{pmatrix}$

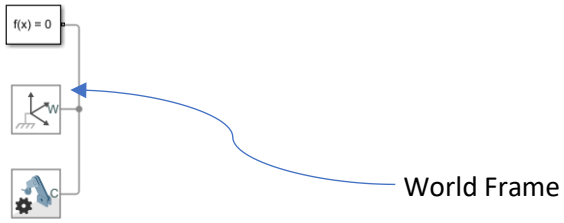
Alors le système peut être modéliser par un State-Space model mais une raison pratique, nous opterons pour la modélisation par Simscape MultiBody qui consiste un construire le système mécanique et grâce à solveur de Simscape on peut obtenir un modèle complet et graphiquement représentable.

Simscape Multibody

Dans cette section, nous présenterons l'outil Simscape Multibody et servira d'introduction pour modéliser notre système grâce à cet outil.

Frames :

Les frames sont les repères associés à chaque solide dans Simscape. Pour avoir un modèle Simscape Multibody fonctionnel, il est nécessaire de définir le repère fixe de l'univers Simscape. Il sera la référence.



En reliant deux frames, nous les superposons, ainsi tout mouvement dans l'un, implique un mouvement dans l'autre.

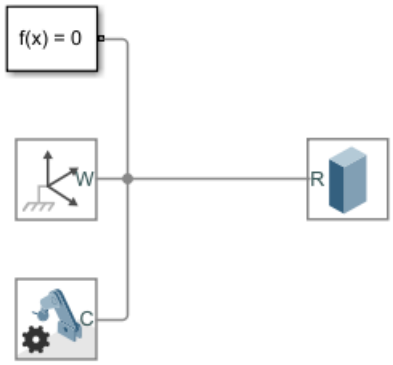
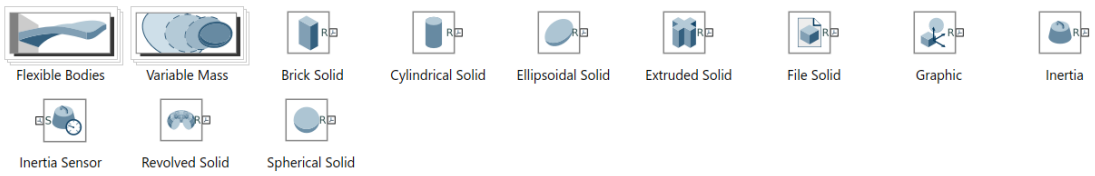


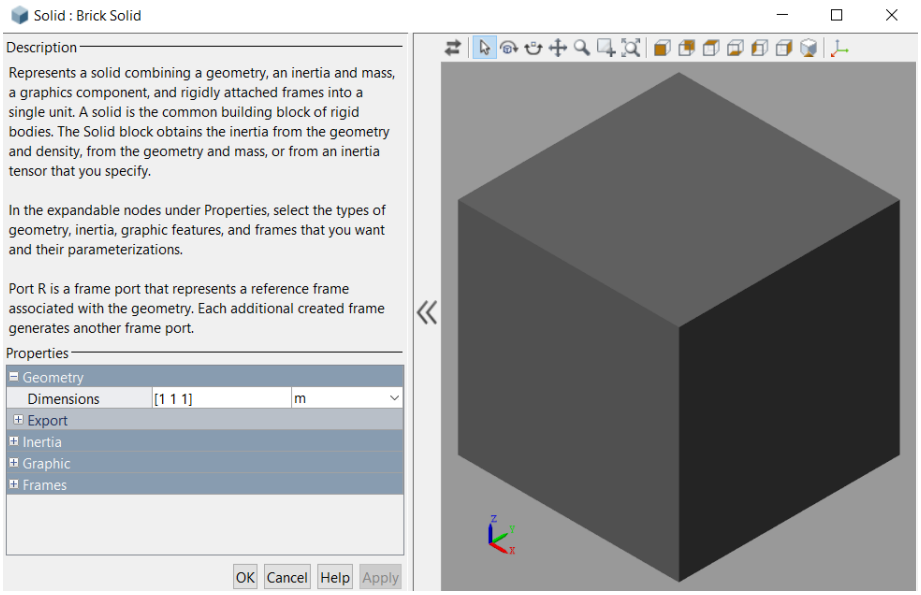
Figure 1: Le repère R, associé au Solide, est relié au Repère Univers. Ceci implique que le solide sera toujours fixe vu que le repère univers est toujours fixe

Les solides :

Dans Library Browser de Simulink (Simscape/Multibody), on retrouve la librairie Body Eléments. Elle comporte tous les composants géométriques de bases.

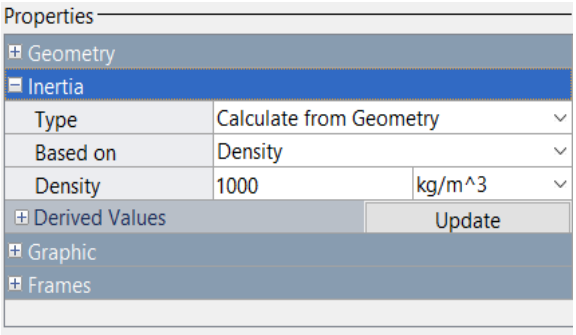


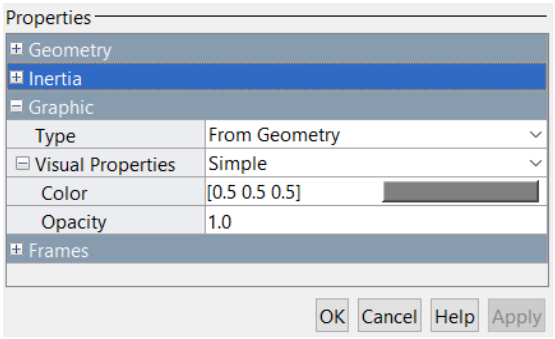
Par exemple, Le brick solide permet de créer un solide de forme prismatique. Pour le configurer, il suffit de double cliquer et on obtient une interface graphique de configuration.



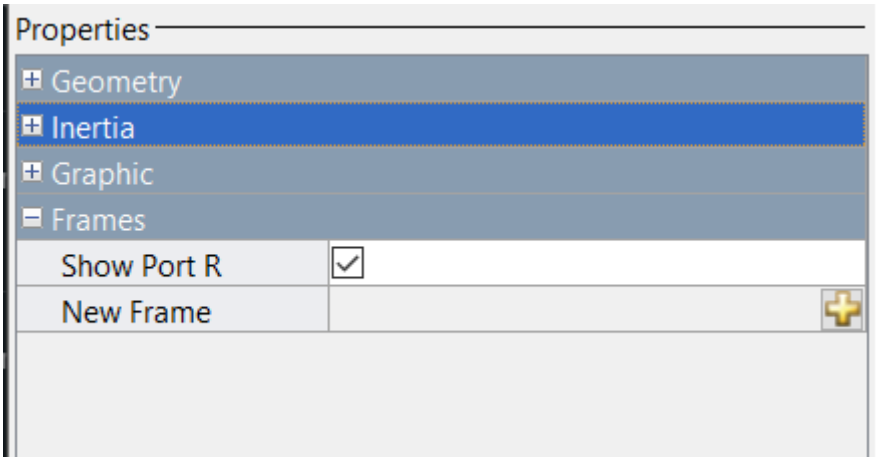
On peut voir le modèle 3D ainsi que des volets ou on peut définir les paramètres géométriques (longueur, largeur et hauteur).

Nous pouvons aussi modifier les paramètres du matériau (Densité) et graphique

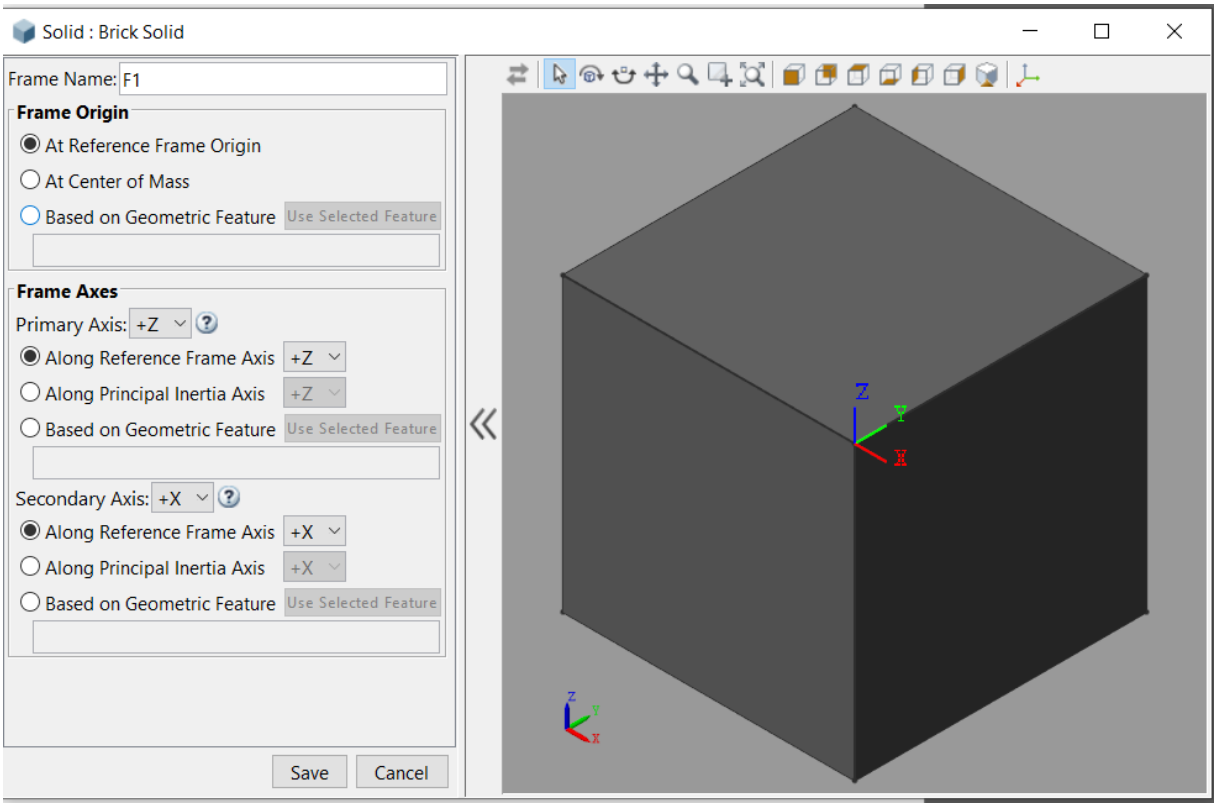


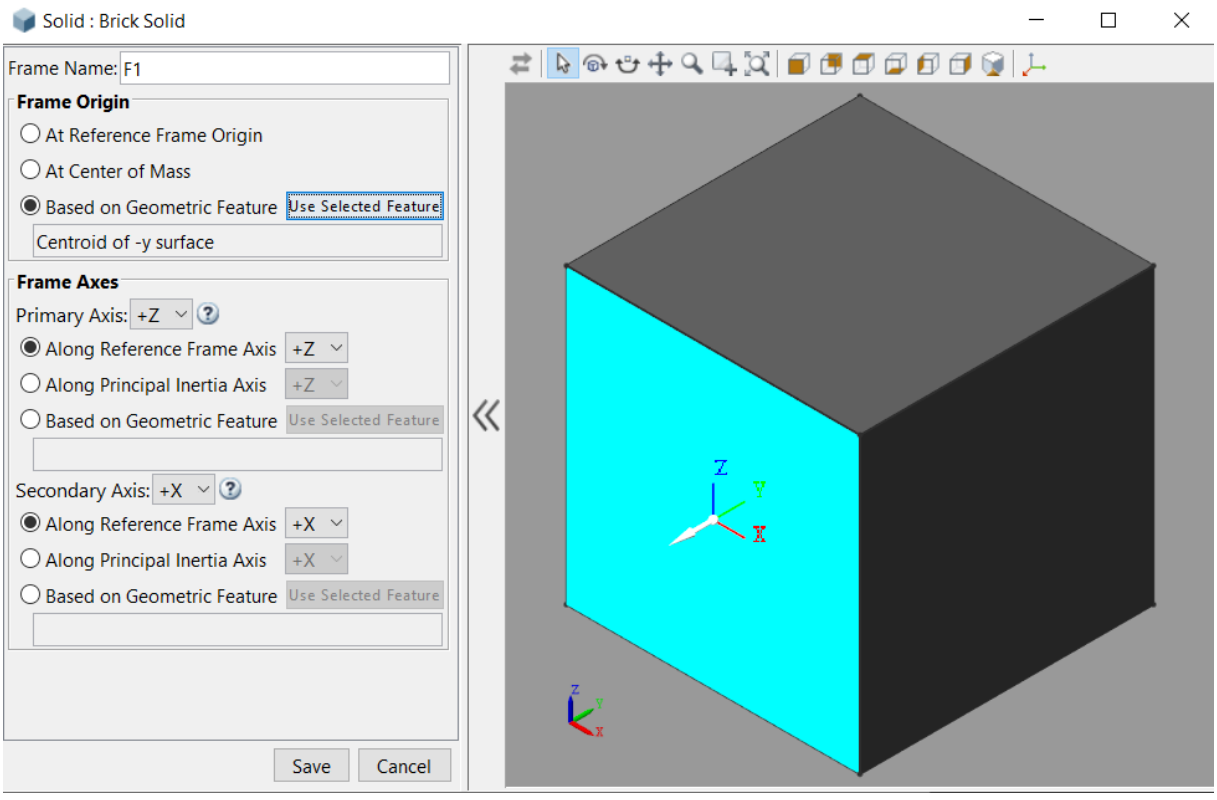


Il est aussi possible de créer d'autres frames (repères) au niveau du solide dans le volet 'Frames'.

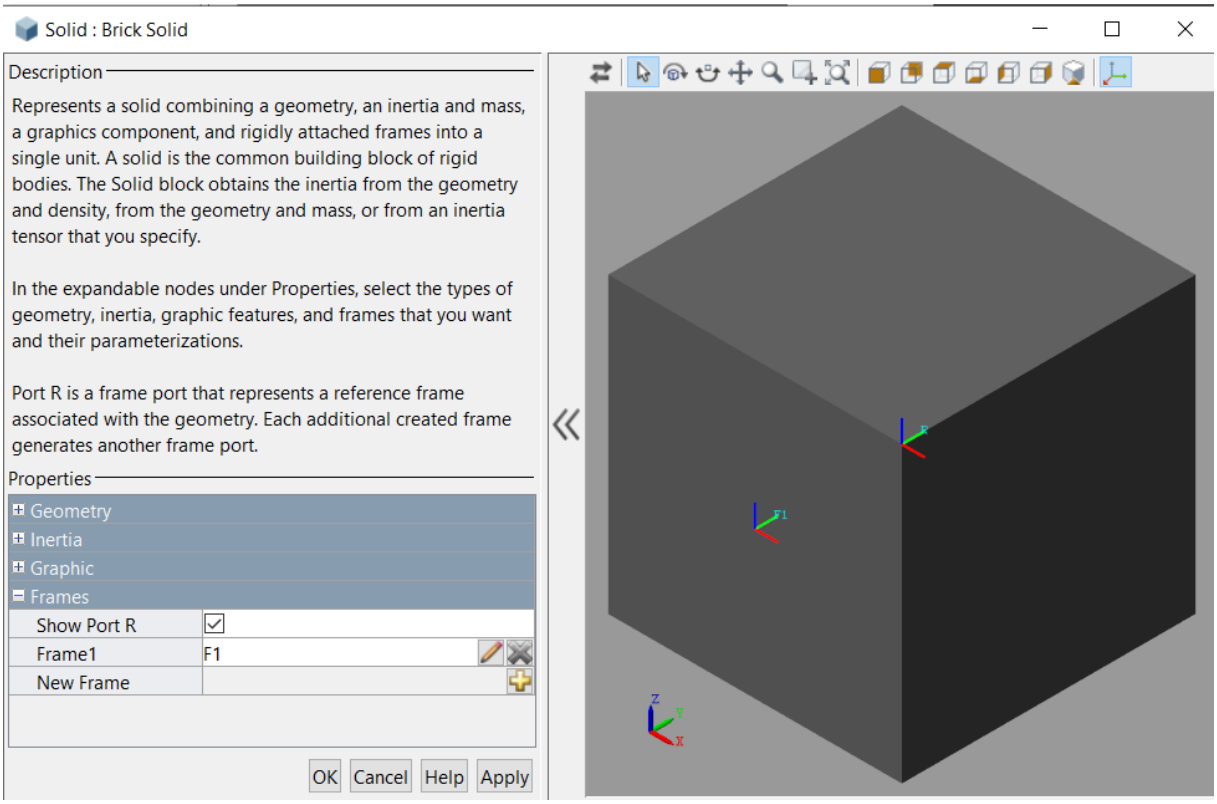


En appuyant sur le bouton '+', une interface s'ouvre. Celle-ci permet de définir la position du repère ainsi que son orientation.

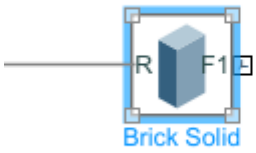




Dans cet exemple, on obtient un second frame F1, sur la face du brick avant solide, une fois le frame enregistrer, on peut le voir sur le modèle 3D du solide.

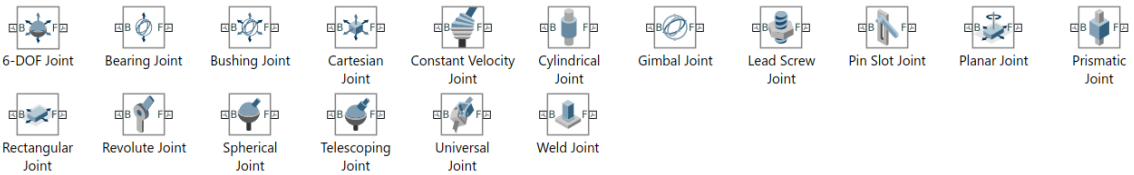


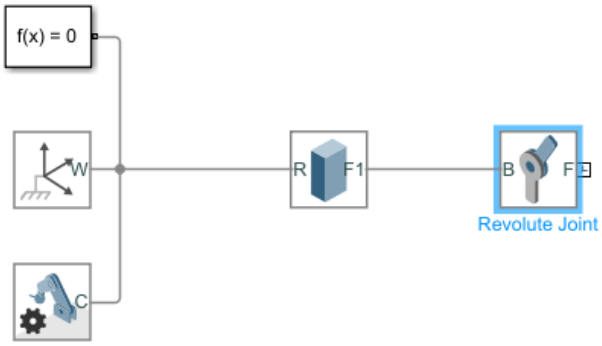
De plus, un nouveau frame apparait sur le block simulink du Brick Solid. Ainsi nous pourons relié F1 à d'autres frames dans notre modèle.

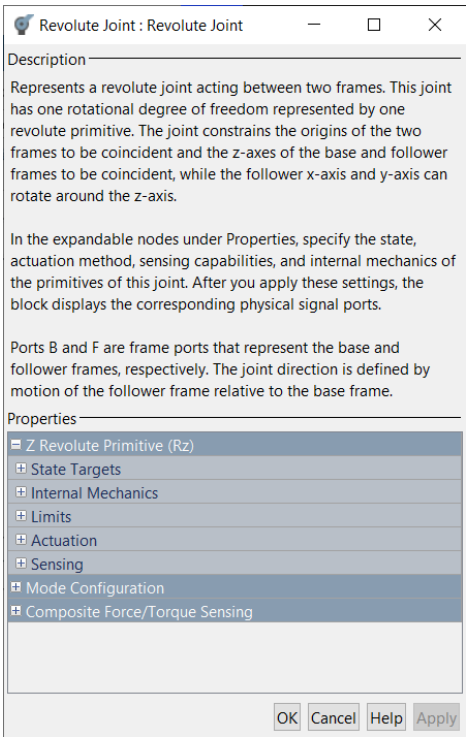


Les liaisons :

Les liaisons sont des composants Simscape Multibody, qui permette d'effectuer des contraintes de mouvements. On les retrouve dans la librairie Simscape/Multibody/Joints.







Prenons par exemple, le revolute joint, cette liaison permet de bloquer 5 degrés de liberté (Tx, Ty, Tz, Rx, Ry) et laisser Rz. Il faut noter qu'il s'agit de Rz du repère propre de la liaison. C'est-à-dire il va falloir transformer le repère de la liaison en utilisant des transformations de repère (Ceci sera traiter ultérieurement).

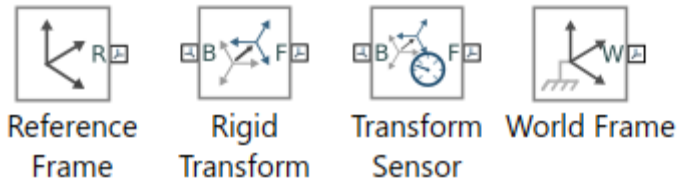
En double-cliquant sur le revolute joint, une interface s'ouvre qui permet de configurer la liaison.

Il est ainsi possible de définir un état initial, ainsi que de définir les paramètres mécaniques de la liaison.

Il permet aussi d'actionner les liaisons par des couples, et de mesurer les variables mécaniques de la liaison (Position, Vitesse, couple, …).

Les transformations :

Les transformations sont des composants Simscape Multibody, qui permette d'effectuer des transformations géométriques sur repères. On les retrouve dans la librairie Simscape/Multibody/Frames and Transform.

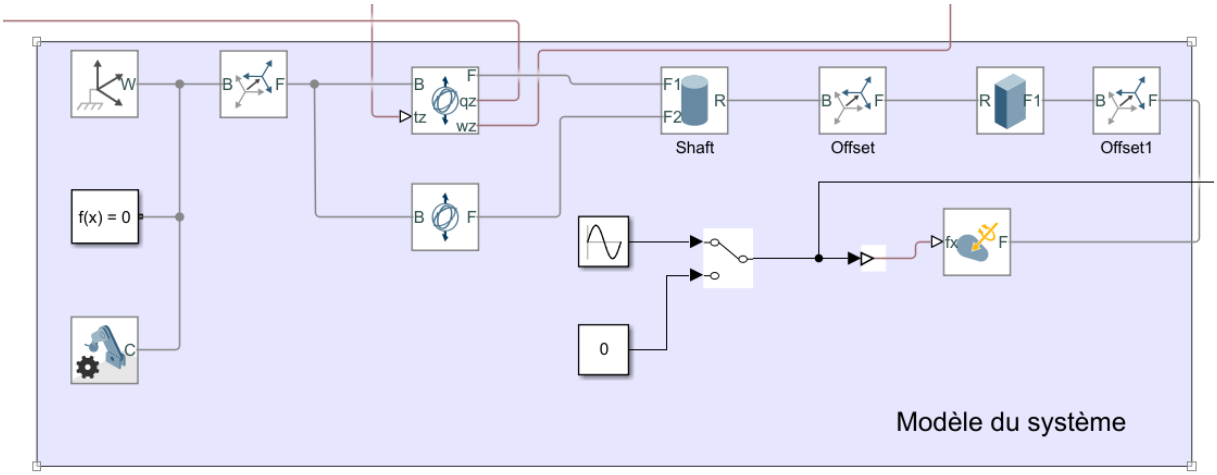


Ce sont des composants essentiels puisqu'elles permettent de gérer le positionnement et l'orientation de chaque composants solide et liaison dans l'espace par rapport à leurs repères respectifs.

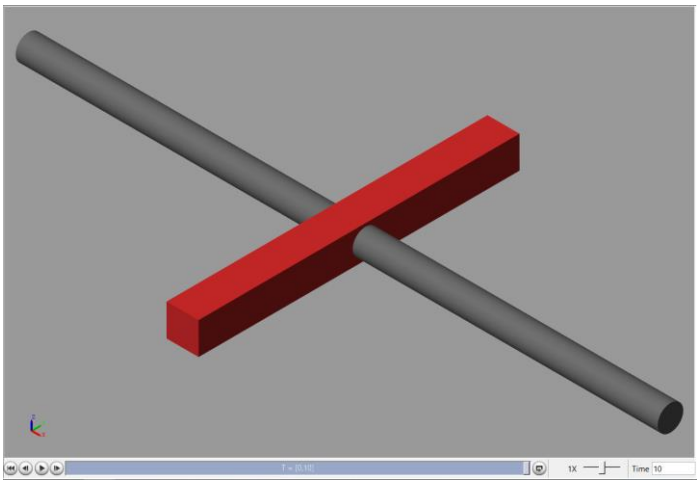
Modélisation du système :

Notre système sera modélisé par un arbre et une barre. L'objectif est d'actionner des moteurs pour positionner la barre à la position désirée. Vu que les moteurs ne sont pas modélisés, on va actionner le système à partir des liaisons, ici il s'agit d'une liaison roulement.

Il faut noter que nous avons ajouté une perturbation sinusoïdale, celle-ci nous permettra de tester la robustesse de notre algorithme de contrôle.



Voici la vue 3D du système :



Les systèmes non linéaires :

Un système non linéaire est un système dont l'équation dynamique s'écrit sous la forme :

$$\dot{x} = f(x, u)$$

Où x est vecteur d'état et u l'entrée de contrôle et f une fonction non linéaire

Critère de Lyapunov :

Considérons un système dynamique et son modèle donné : $\dot{x} = f(x)$ et x=0 est le point d'équilibre du système.

Le système est stable s'il est déplacé de son point d'équilibre à une position donnée, il évoluera vers le point d'équilibre et y restera.

Nous devrions souligner qu'un système donné peut avoir plusieurs points d'équilibre, de sorte que le système peut être stable pour certains points d'équilibre et peut ne pas l'être pour d'autres. Ensuite, la stabilité n'est pas la propriété du système, mais est plutôt la propriété intrinsèque du point d'équilibre.

Le critère de stabilité de Lyapunov est un critère qui permet d'effectuer une analyse de stabilité d'un point d'équilibre donné.

Le critère consiste en le choix d'une fonction appelée fonction Lyapunov ou fonction « énergie ».

UNE fonction V doit vérifier les conditions suivantes pour être comme considérer comme fonction Lyapunov :

1. V est une fonction de classe C^1
2. V doit être définie positive (i.e., V>0 and V(0)=0)
3. $\dot{V} < 0 \; and \; \dot{V}(0) = 0$

Théorème :

S'il existe une fonction Lyapunov pour x=0, alors x=0 (le point d'équilibre) est stable.

Example :

Soit un système linéaire $\dot{x} = -x^3$ and x=0 son point d'équilibre.

Choisissons une fonction Lyapunov pour effectuer l'analyse. La fonction choisie doit vérifier les critères ci-dessus.

Prenons, par exemple, la fonction $V(x) = \frac{1}{2}x^2$

1. Cette fonction est certainement une fonction de classe C^1
2. V(x)>0 et V (0) =0
3. $\dot{V}(x) = x\dot{x} = -x^4 < 0 \; et \; \dot{V}(0) = 0$

Alors, la fonction Lyapunov existent pour ce point d'équilibre, et selon le théorème ci-dessus, ce point d'équilibre est stable.

Sliding Mode Control :

The Sliding mode Controller (SMC) peut être considéré comme le résultat naturel de la méthode de stabilité Lyapunov. Cet algorithme consiste à choisir une fonction S(x) appelée “Sliding Surface". L’objectif de l’algorithme est de conduire le système de sorte que la « Sliding Surface » devienne zéro et en même temps conduire le vecteur de l’espace d’état à l’état désiré.

Donc, nous voulons en fait avoir un état stable à $S(x_{desired}) = 0$

Pour y parvenir, nous devons définir un signal de contrôle qui peut vérifier le critère de stabilité de Lyapunov $S(x_{desired}) = 0$.

Définissons d’abord la fonction Lyapunov :

$$V(S) = \frac{1}{2}S^2$$

La fonction Lyapunov choisie est fonction de la classe 1. Et *V(S)>0 et V (0) =0*

Donc, la condition restante est d’avoir $\dot{V}(S) < 0 \; and \; \dot{V}(0) = 0$.

Pour parvenir à la stabilité, nous devons définir l’entrée de contrôle « u » afin que nous puissions avoir $\dot{V} < 0$:

$$\dot{V} = S\dot{S}$$

Considérons un système défini par : $\dot{x} = f(x) + U$

On définit $e = x - x_{desired}$

Et sa « Sliding surface » est $S = \dot{e} + e$

$$\dot{V} = S(\dot{x}) = S(f(x) + U)$$

Donc, nous devons choisir U d’une manière que nous pouvons avoir $\dot{V} < 0$

Par exemple, $U = -f(x) - \eta \; sign(S)$

$$\dot{V} = -\eta . sign(S) . S = -\eta |S| < 0$$

Cette dernière condition est appelée ***Reachability Condition.***

En effet lorsque le système atteint le mode de glissant càd S=0, le système est alors déterminé par l’équation dynamique

$$\dot{e} + e = 0$$

Alors, le système converge exponentiellement.

Super Twisting Sliding Mode Control :

Bien que le « Sliding Mode Control » classique est un algorithme robuste, il peut présenter certains comportements indésirables, tels que le « Chattering ». « Chattering » est un phénomène où des oscillations à hautes fréquences se produisent lorsque le système atteint la « Sliding Surface ». La prévention de ce phénomène nécessite des composants de commutation infiniment rapides. Ce qui est impossible à réaliser puisque tout système physique à l'inertie. Alors on utilise un algorithme Sliding Mode Control de 2ème ordre. Dans cet algorithme, le *Sliding Mode* est défini par $S = 0 \; et \; \dot{S} = 0$.

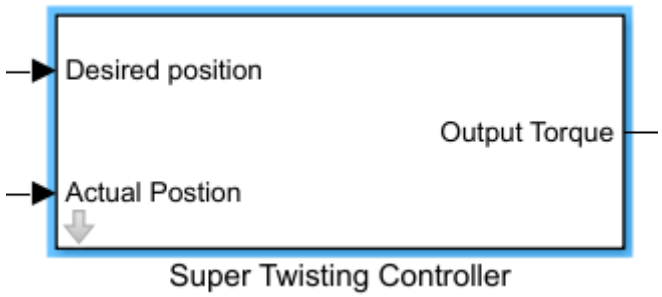
L'entrée de contrôle de l'algorithme du « Super Twisting » est définie par :

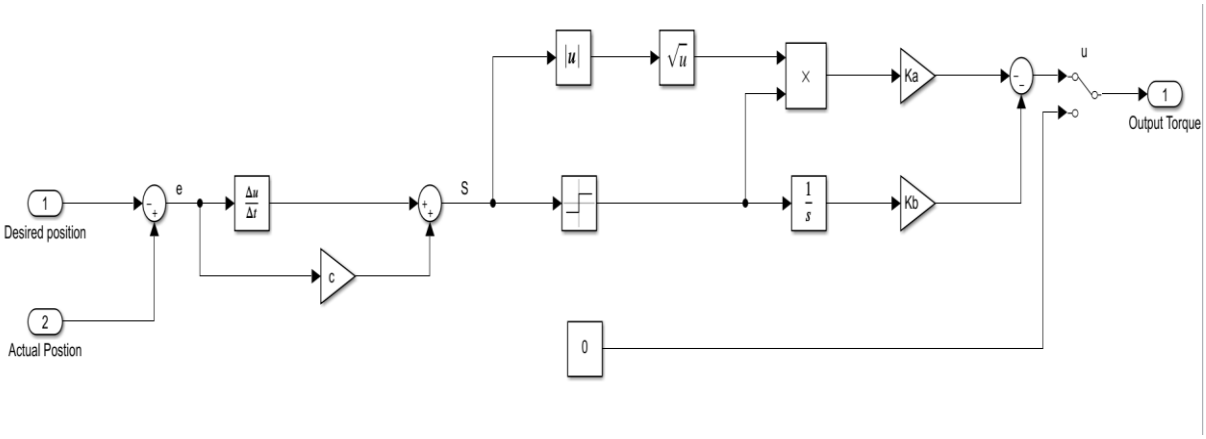
$$u = -f(x) - \mu . |S|^{1/2} . sign(S) - \beta . \int_0^t sign(S) d\tau$$

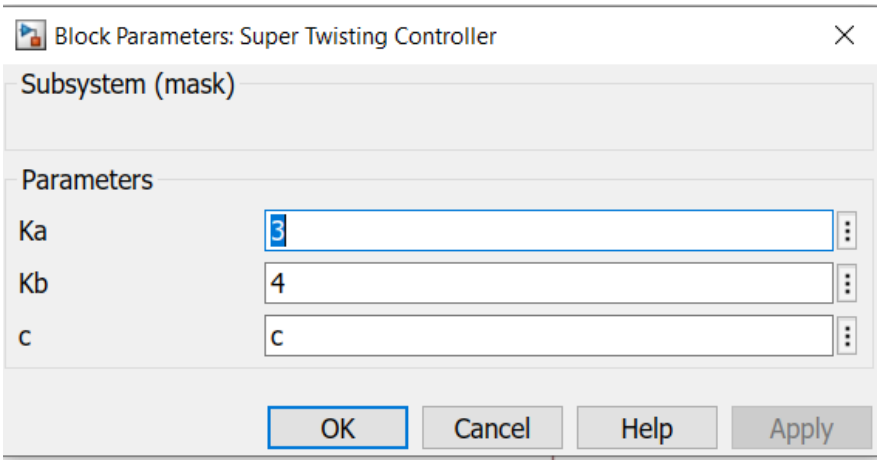
Comme nous pouvons l'observer, cette technique de contrôle n'est plus discontinue. En effet, $sign(S) . |S|^{1/2}$ et $\int_0^t sign(S) d\tau$ sont continus.

Implémentation Simulink :

La figure ci-dessous est l'implémentation Simulink de l'algorithme Super-Twisting. L'objectif est de définir les paramètres Ka et Kb et c, afin d'avoir le comportement voulu au niveau du système.







Voici les résultats obtenus pour les valeurs ci-dessus :

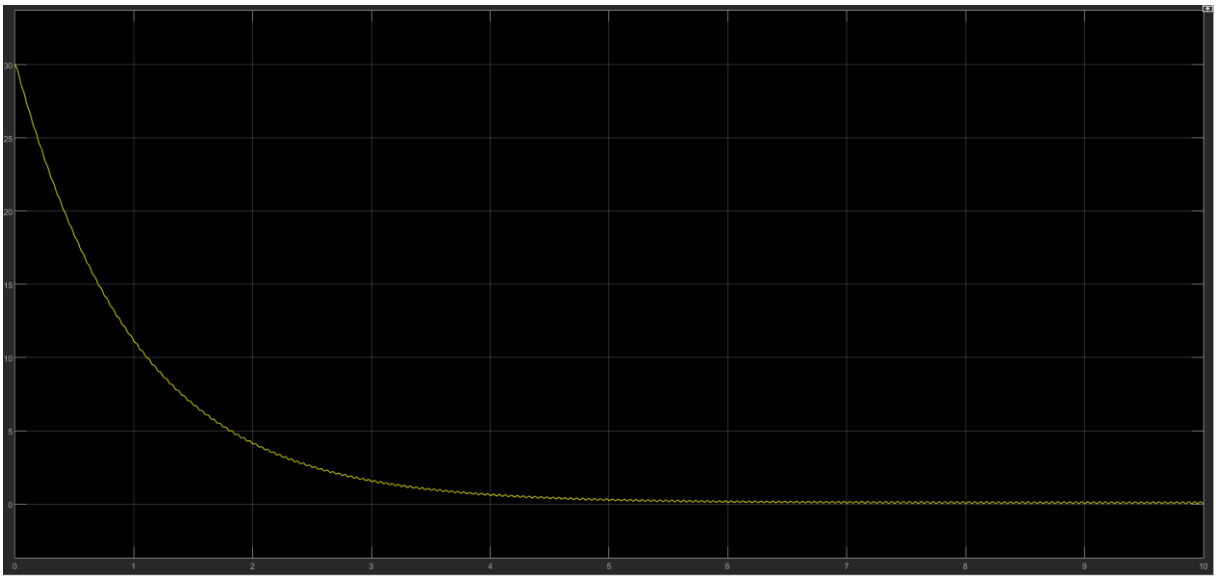


Figure 2: La position angulaire de la barre

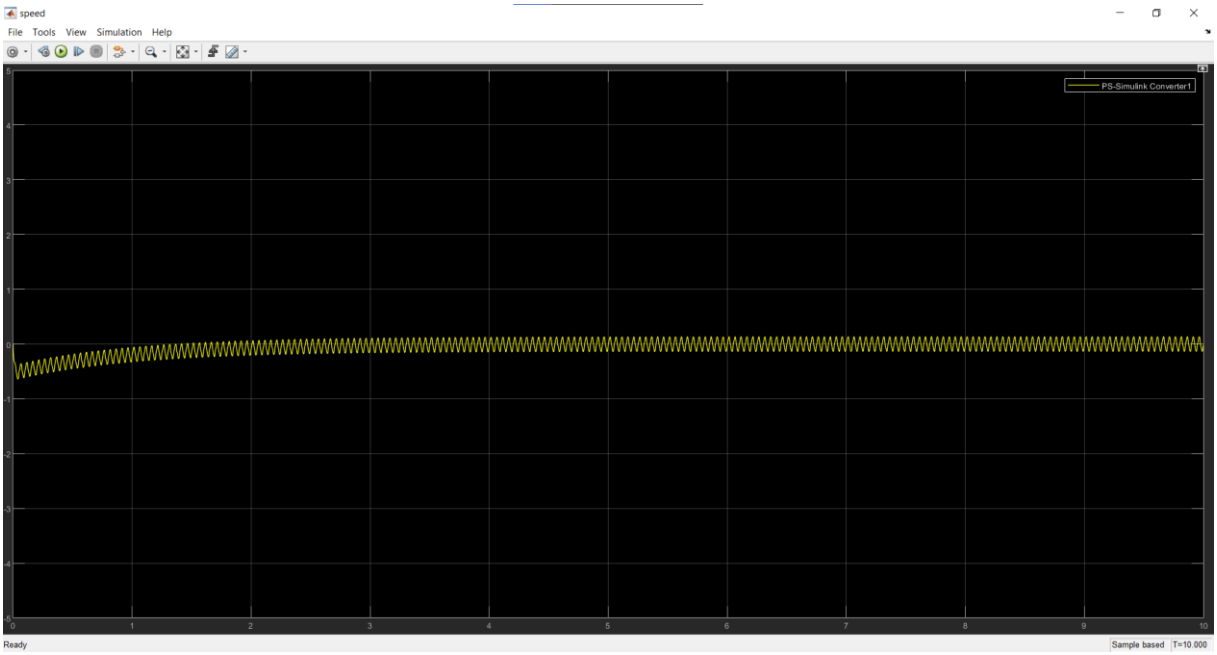
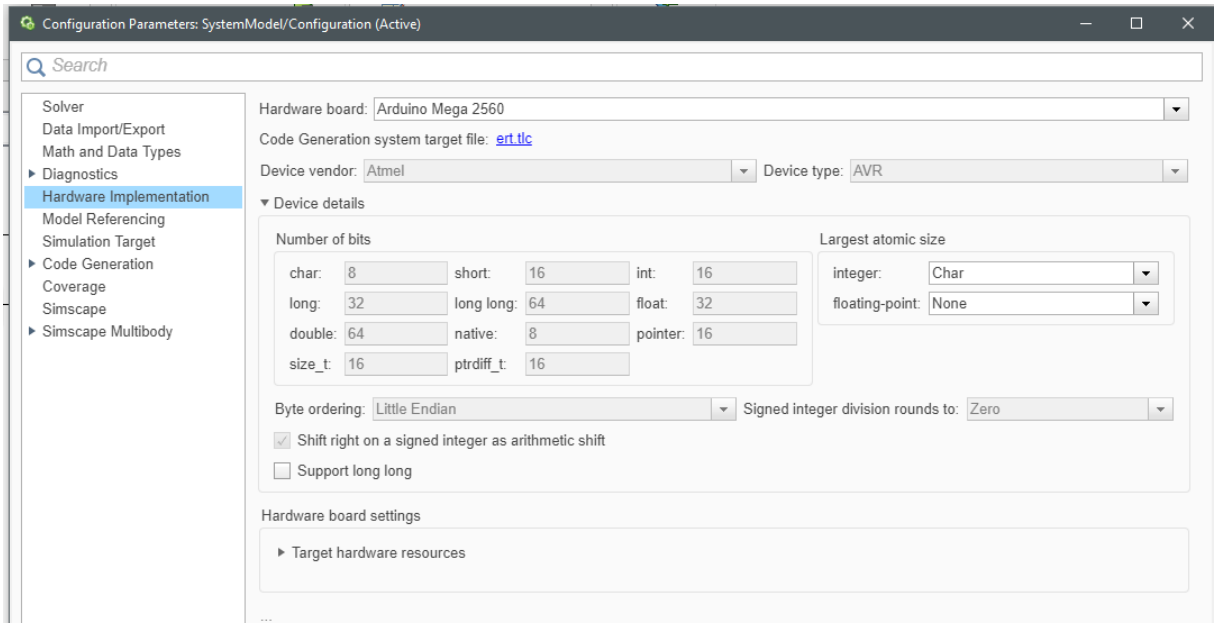


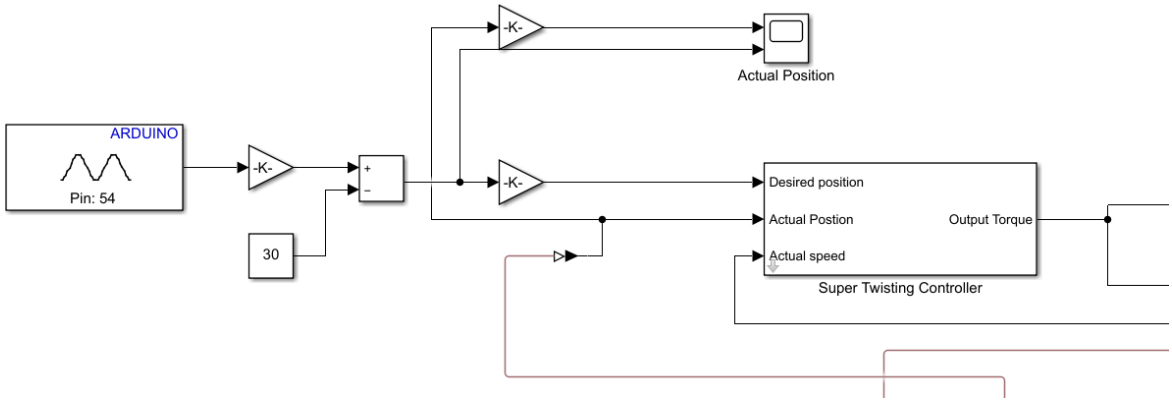
Figure 3: La vitesse angulaire de la barre

Déploiement de l'algorithme dans une carte microcontrôleur :

Pour ce faire, nous utiliserons Arduino Support Package for Simulink. Il nous permettra de contrôler notre modèle Simscape Multibody à partir de la carte de développement Arduino.



On ajoute à l'entrée de la commande un bloc permettant de lire la valeur depuis le pin analogique A0 ensuite on effectue un simple mapping linéaire pour avoir une gamme de commande de 30 dégrées dans les deux sens.

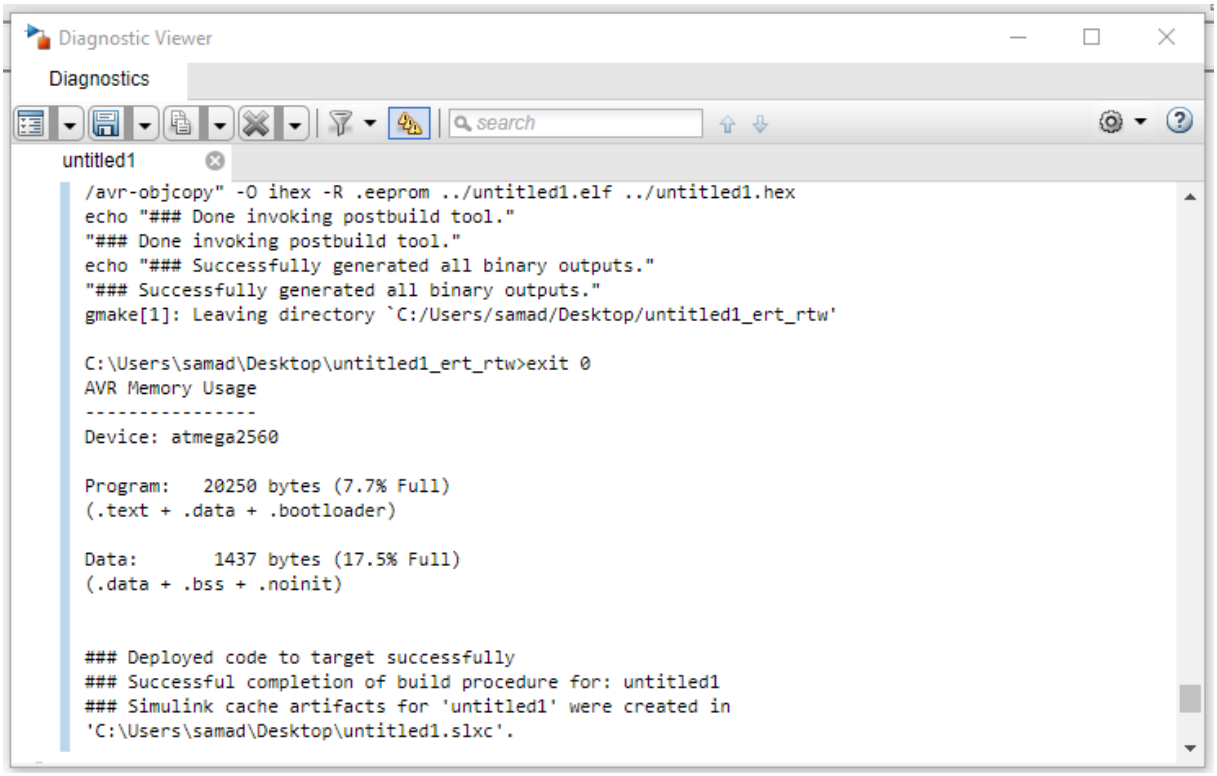


Vu la complexité du model Simscape, il n'est pas possible de réaliser la simulation en mode directe sur la carte Arduino, dans ce cas on opte pour une simulation de type HIL.

La simulation Hardware-in-the-Loop (HIL), ou HWIL, est une technique utilisée dans le développement et le test de systèmes embarqués en temps réel complexes. La simulation HIL fournit une plate-forme efficace en ajoutant la complexité de l'usine sous contrôle à la plate-forme de test. La complexité de

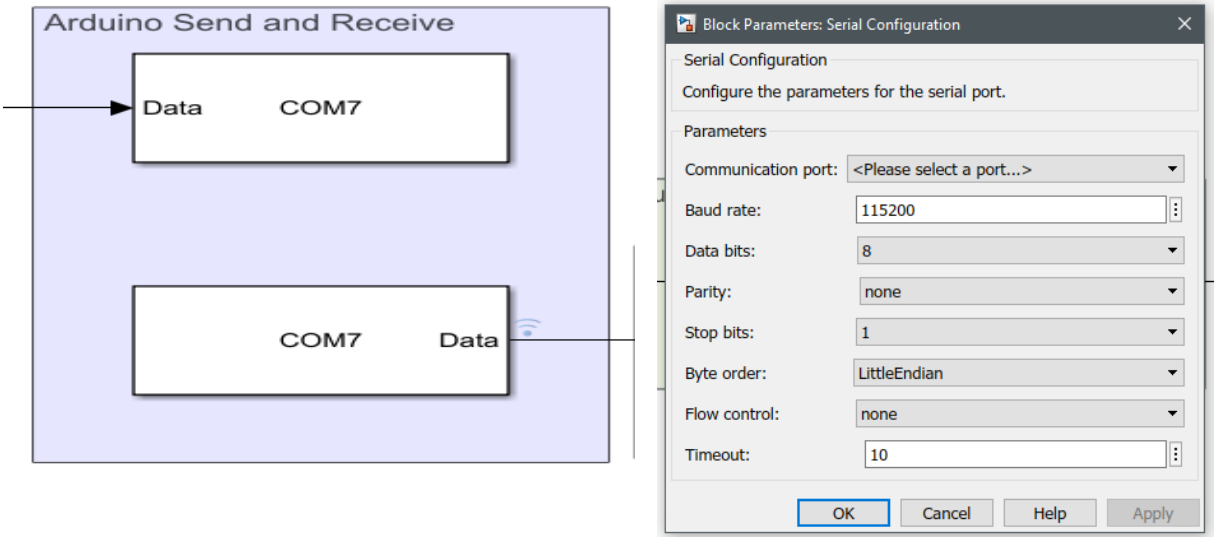
l'installation sous contrôle est incluse dans le test et le développement en ajoutant une représentation mathématique de tous les systèmes dynamiques associés. Le système embarqué à tester interagit avec cette simulation d'usine.

En ce moment, il n'existe des solutions de simulation HIL temps réel que pour les contrôleurs avancés en termes de puissance de calcul et de IO, on peut cependant vérifier la possibilité du déploiement du bloc contrôleur :



Pour réaliser cette architecture de simulation, il faut donc implémenter le contrôleur sur carte et ensuite interfacée cette dernière avec le model sur Simulink, et ce à travers le protocole de communication série.

On doit donc faire appelle au blocs '***Serial Transmit'*** et '***Recieve'*** de la librairie '***Instrument control' :***



La configuration du protocole série se fait selon le mode par default 8-N-1 selon la documentation du module Serial sur Arduino :

Parameters

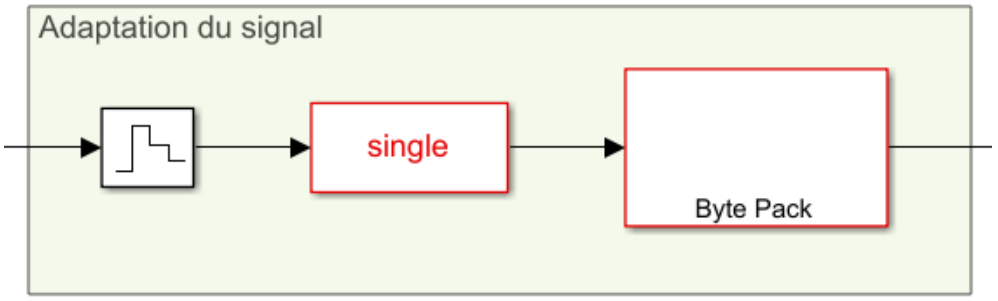
Serial: serial port object. See the list of available serial ports for each board on the Serial main page.

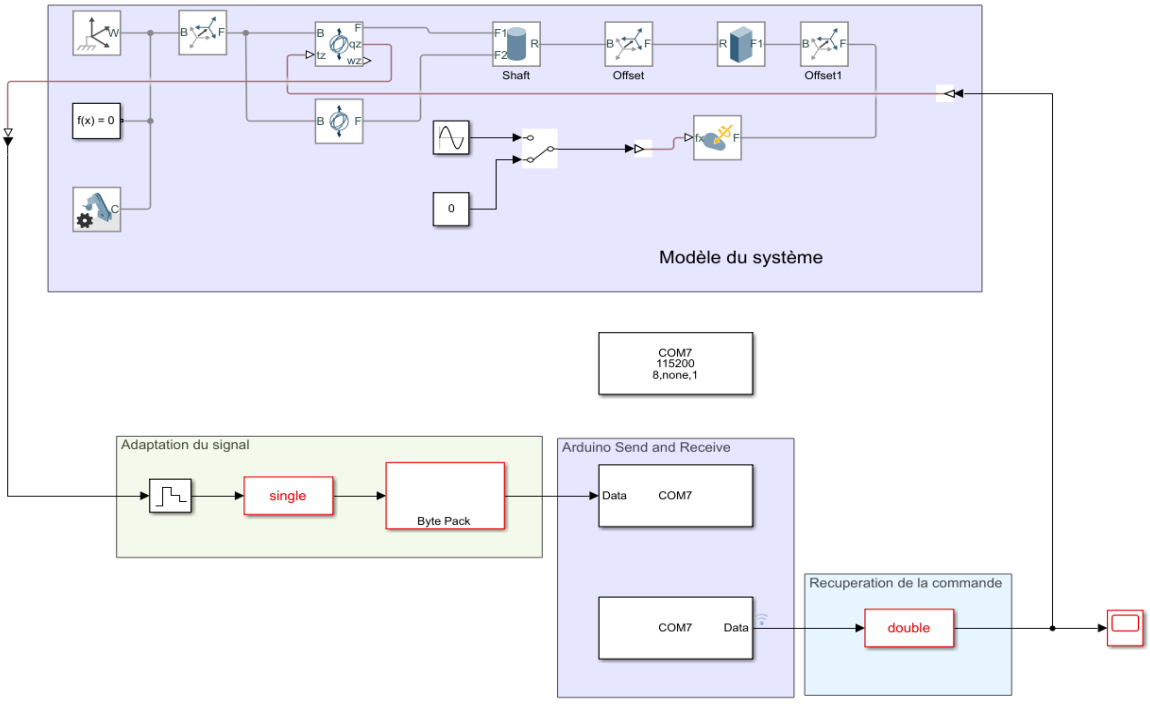
speed: in bits per second (baud) - **long**

config: sets data, parity, and stop bits. Valid values are

SERIAL_5N1
SERIAL_6N1
SERIAL_7N1
SERIAL_8N1 (the default)
SERIAL_5N2
SERIAL_6N2

Pour préparer le signal, on effectue d'abord un échantillonnage grâce au bloc échantillonneur bloqueur ensuite on effectue les conversions depuis le format double circulant dans le model vers le format float que va traiter Arduino :





Du coté Arduino, il faut donc écrire un code qui va recevoir 4 bytes, les transformer en float et ensuite calculer la commande puis l'envoyer sous la même forme. Pour cela on utilise une structure union qui est une valeur qui possède plusieurs représentation ou format en la même case mémoire, cette structure est définie comme suit :

```
typedef union
{
    float number;
    uint8_t bytes[4];
} FLOATUNION_t;
```

Le contrôleur est aussi traduit par la fonction suivante :

```
void super_t()
{
    ActualPosition = var.number;
    error = ActualPosition - DesiredPosition;
    S = (error - error_temp) / deltatime + c * error;
    error_temp = error;
    ua = sqrt(abs(S)) * sgn(S);
    ub = sgn(S) * deltatime + ub;
    u = -Ka * ua - Kb * ub;
    u = constrain(u, -6, 6);
    var.number = u;
}
```

Conclusion :

La réalisation de ce projet nous a donner l'opportunité d'avancer nos connaissances en termes de contrôle avancée des systèmes. Le contexte de la pandémie Covid ainsi que les problèmes au niveau du banc de test nous ont malheureusement limités en termes d'avancement de l'opération du déploiement et de la vérification.

Néanmoins cet obstacle nous a motivé pour retrouver des solutions alternatives et ainsi découvrir un nouvel aspect étant la simulation des systèmes physiques.

La nature du sujet qui couvre les drones nous a aussi pousser à explorer les différents aspects théoriques liée à ce domaine notamment l'acquisition et le traitement des données d'un capteur de vol non couvert sur ce rapport.