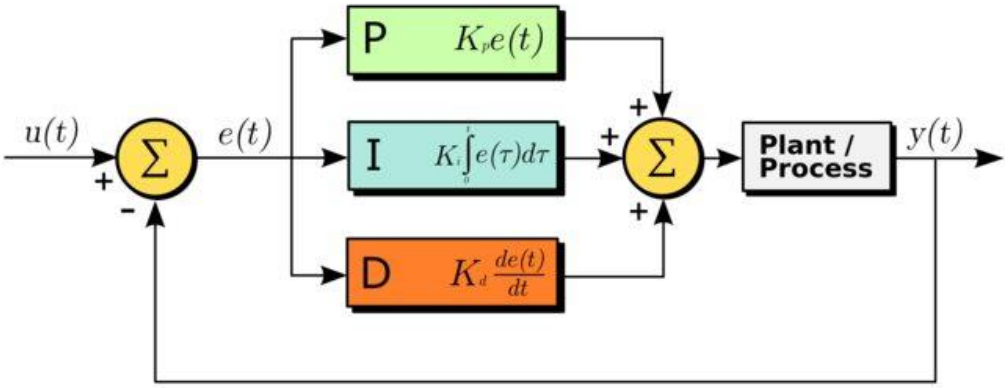
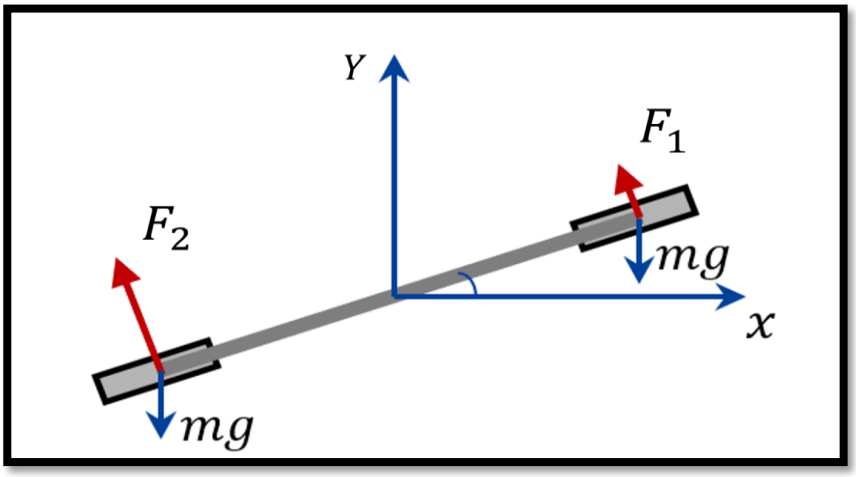
Module BAu211
AUTOMATIQUE
2024-2025

Etudiants :
Madi ZABIKH
Franck-wilfrid KODJO

Projet-Drone





Introduction

Dans ce projet nous allons modéliser un drone à partir du logiciel Matlab en nous appuyant sur les connaissances acquis au cours du module « Automatique » de la formation bachelor2 aéronautique à l'IPSA. Il s'agira pour nous de contrôler le comportement d'un drone dans un système à boucle fermé et à retour unitaire avec des consignes de position sur l'axe x et l'axe y. Notre système est linéaire et à temps continu.

Linéarité :

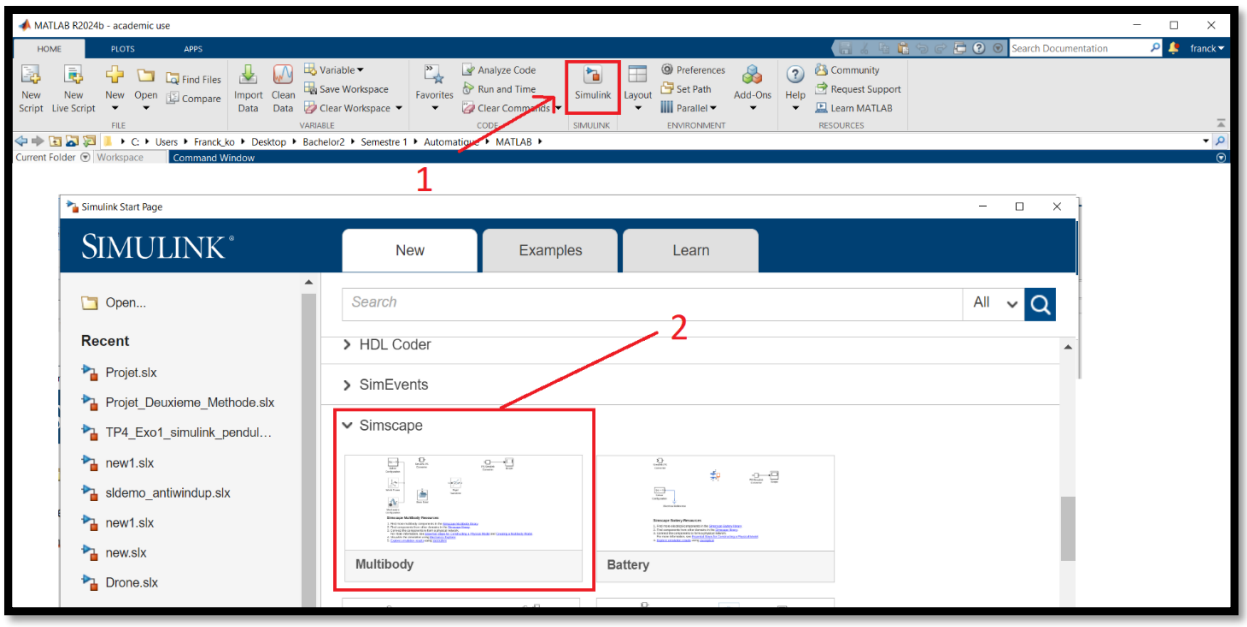
Pour notre drone, il sera nécessaire de contrôler la position, la vitesse et l'accélération. Le modèle peut ainsi être décrit par une équation différentielle de la forme suivante :

$$m \cdot \ddot{y}(t) + k \cdot \dot{y}(t) + b \cdot y(t) = u(t)$$

Pour résoudre cette équation différentielle, nous pouvons utiliser la transformée de Laplace.

Création d'un nouveau projet Simscape sous Matlab:

1. Accédez à l'environnement de schéma bloc « **Simulink** », dont l'icône se trouve dans la barre d'outils.
2. Choisissez ensuite les outils de simulation multi-domaine **Simscape** et **Multibody**.



Ce projet sera développé de manière progressive en plusieurs étapes, avec des ajouts et des modifications du système au fur et à mesure de l'avancement.

Partie 0 : Modélisation physique du drone

Objectif :

Dans cette première partie, nous établissons la structure de base du drone en utilisant **Simscape Multibody**. L'objectif est de créer une représentation physique et modulaire qui servira de fondation pour les étapes suivantes. De plus, les paramètres physiques spécifiques au drone, comme sa masse, ses dimensions et sa puissance maximale, sont définis conformément au cahier des charges.

Cahier des charges

1. **Caractéristiques physiques du drone** :
   - **Masse :** 200 g
   - **Dimensions** : [44, 4, 2] (unités arbitraires, probablement en cm)

2. **Puissance maximale F**:

- La puissance maximale est déterminée en fonction de la masse du drone. À l'équilibre, la force est donnée par la relation suivante :

$$\vec{F} = m \cdot \vec{g}$$

où $\vec{g}$ est l'accélération gravitationnelle.

- Pour faire monter le drone, il est nécessaire que le coefficient α soit supérieur à 1, avec $\alpha = \frac{\vec{F}}{m \cdot \vec{g}}$
  - Si $\alpha = 1$, le drone reste en équilibre.
  - Si $\alpha = 2$, la force appliquée pour la montée est équivalente à une chute libre inversée, ce qui est trop brusque.
- Pour des raisons de sécurité et de contrôle, nous avons choisi arbitrairement $\alpha_{max} = \frac{\vec{F}}{m \cdot \vec{g}}$
- Ainsi, la force maximale choisie $\frac{3}{2} m \cdot \vec{g}$ est donnée par la relation :

$$F_{max} = F_{1\,max} + F_{1\,max} = \frac{3}{4} m \cdot \vec{g} + \frac{3}{4} m \cdot \vec{g}$$

Étapes de modélisation

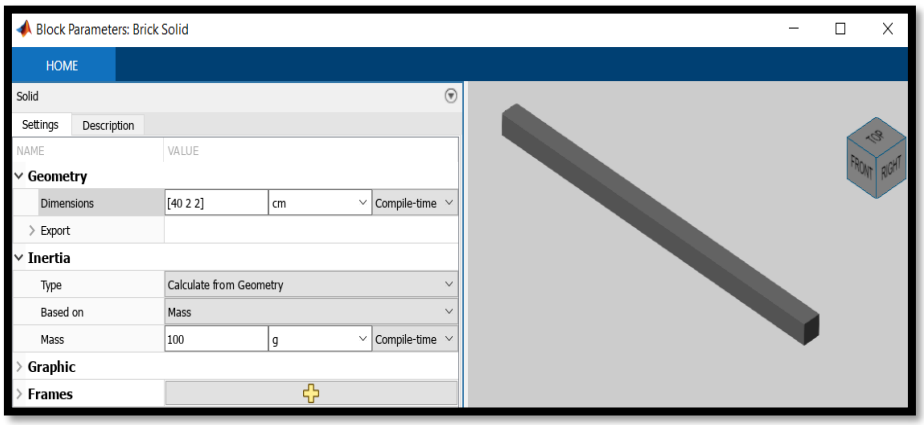
1. **Création de la structure principale** :

- Construction du drone :

  Nous utilisons un composant **Brick Solid** pour représenter le corps principal du drone.

  Nous aurons besoin de trois fois l'objet « Brick solid » disponible dans la librairie (Accès : simscape -> Multibody -> Body éléments -> Brick solid).

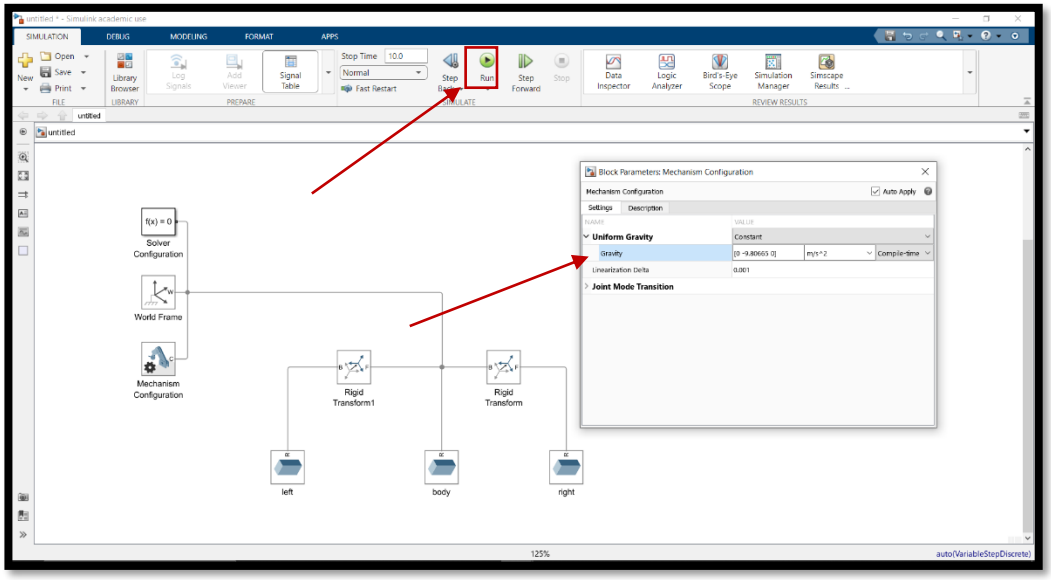
- Deux de ces objets représenterons les propulseurs de masse 50g chacun et de dimension [2 ; 4 ; 2]. Le troisième objet « brick solid » aura une masse de 100g et de dimension [40 ; 2 ; 2]

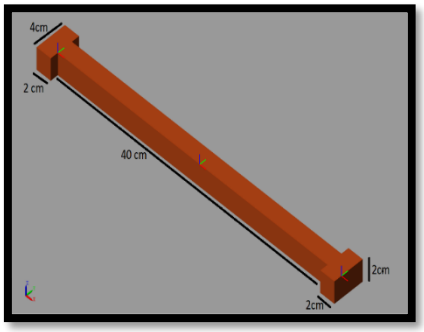


- Deux blocs **Rigid Transform** sont ajoutés de part et d'autre du **Brick Solid** pour établir des points de connexion avec les autres parties, notamment les moteurs, représentés par deux autres **Brick Solid**.
- Enfin pour relier entre eux ces trois objets nous aurons recours à l'outil « Rigid transform » disponible dans la librairie (Accès : simscape -> Multibody -> Frames

and transform -> Rigid transform). Dont les parmètres presentés sur la figure ci-dessous.

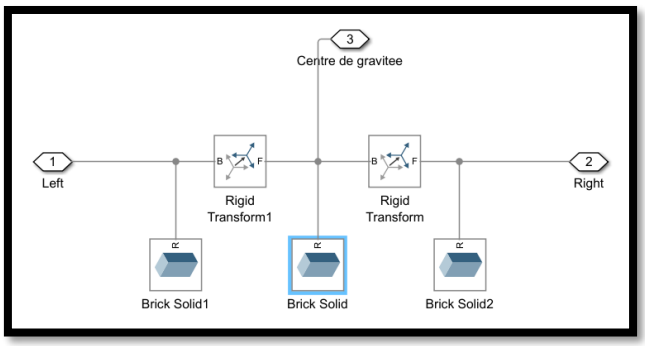
- Pour une première simulation, nous relierons nos objets entre eux avant de lancer la simulation. Sans oublier de fixer la force de pesanteur $\vec{g} = -9{,}8\,\vec{j}$ sur l'axe Y :

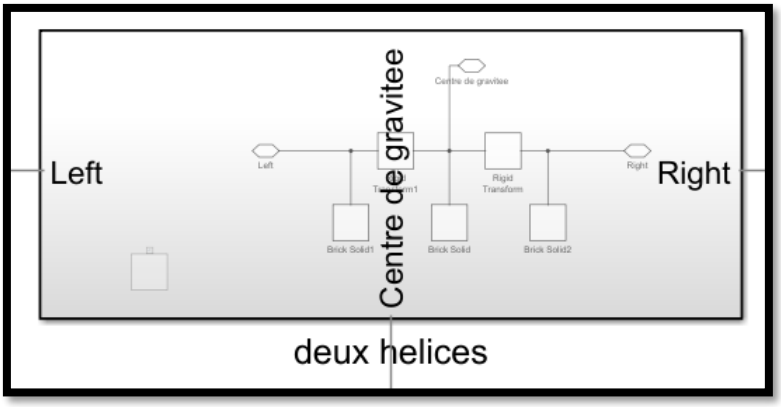




2. **Organisation dans un sous-système** :

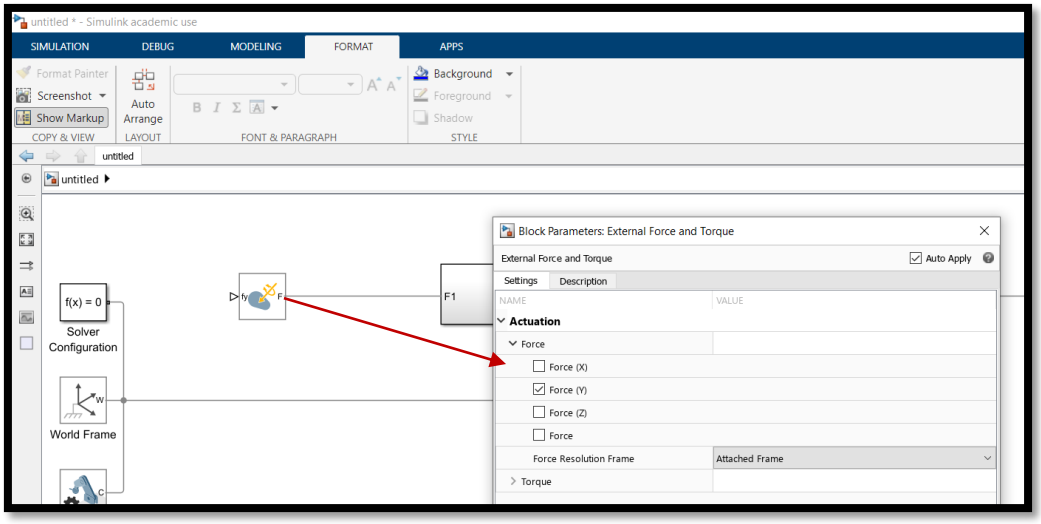
- Pour simplifier la gestion et la visualisation, l'ensemble du modèle du drone est encapsulé dans un **Subsystem**.



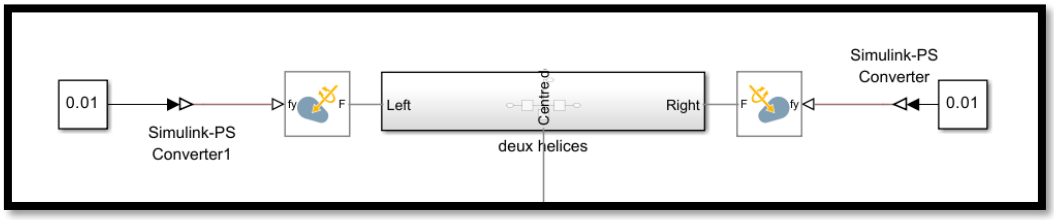


3. **Ajout des forces externes** :

- Deux blocs **External Force and Torque** sont placés à gauche et à droite du sous-système pour permettre la manipulation des forces appliquées sur le drone (Left/Right).
- Ont rajoutera deux forces extérieur représentant la portance créée par les propulseurs, et orienté sur l'axe Y. Pour cela utilisons l'outil « external force and torque » de la librairie (Accès : simscape -> Multibody -> Force and torque -> External force and torque).



- Ces blocs sont connectés à des blocs **Constant** via des **Simulink-PS Converters**, permettant d'entrer des valeurs spécifiques pour manipuler les forces.

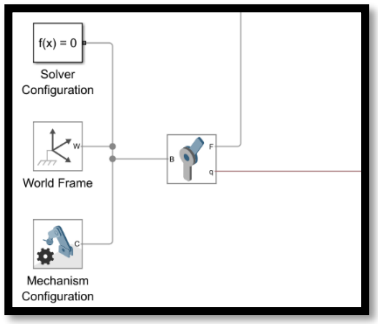


Partie 1 : Roulis sur l'axe Z

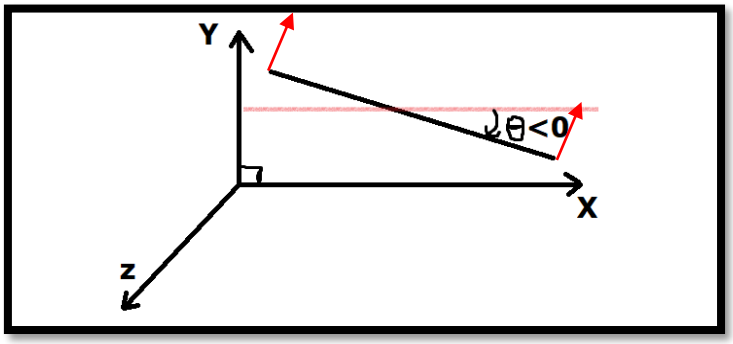
1. **Connexion au Revolute Joint(du point de pivot)** :

- Ont rajoutera alors on axe de rotation pour le roulis avec l'outil « Revolut joint » de la librairie (Accès : simscape -> Multibody -> Joints -> Planar joint). Qui nous permet une translation sur l'axe X, sur l'axe Y et seulement une rotation sur l'axe Z. Et l'axe de rotation sera fixé au centre de gravité de notre drône. Le paramètre « sensing -> position » pour récupérer les valeurs de la position sur X et Y puis de l'angle d'inclinaison sur Z.

- Le centre de gravité du drone a été connecté à un **Revolute Joint**, permettant de simuler une rotation autour de l'axe de roulis.
- Le **Revolute Joint** est relié à :
  - **Solver Configuration** pour gérer la résolution numérique,
  - **World Frame** pour définir le cadre de référence global,
  - **Mechanism Configuration** pour paramétrer les propriétés mécaniques du modèle.



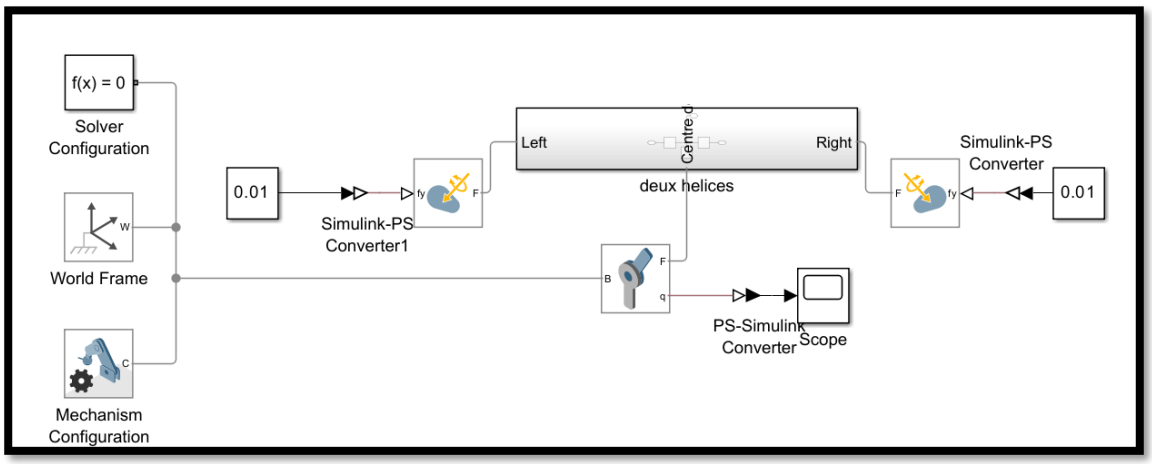
2. **Control de l'angle d'inclinaison (Boucle fermé)** :



- Lorsqu'on commande un angle positif θ au drone, l'angle lu par notre capteur est à valeur négative car incliné dans le sens horaire. Pour calculer donc l'erreur, il ne s'agira pas de faire une soustraction mais plutôt la somme de la consigne avec la valeur retournée par le capteur. Lorsque le drone est incliné, les forces F1 et F2 sont à norme inégale (Projeté sur *Y*) cette différence noté $u = cos(\theta)F1 - \cos(\theta)F2$. Et pour que le drone soit équilibré, dans son état d'inclinaison, il faut $\cos(\theta)F1 - cos(\theta)F2 = mg$. De ces deux équations, on déduit $\begin{cases} F1 = (u + mg)/2\cos(\theta) \\ F2 = (u - mg)/2\cos(\theta) \end{cases}$

Résultat final

Le modèle final de cette étape, incluant la structure physique et les paramètres du cahier des charges :



Partie 2.1 : Ajouter le déplacement selon l'axe horizontal X

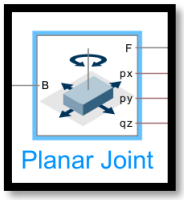
Objectif

Dans cette partie, nous ajoutons la capacité au drone de se déplacer selon l'axe ***X*** en intégrant un **Planar Joint** à la place du **Revolute Joint** utilisé précédemment. Nous implémentons également un contrôle PID pour gérer les mouvements et les positions du drone.

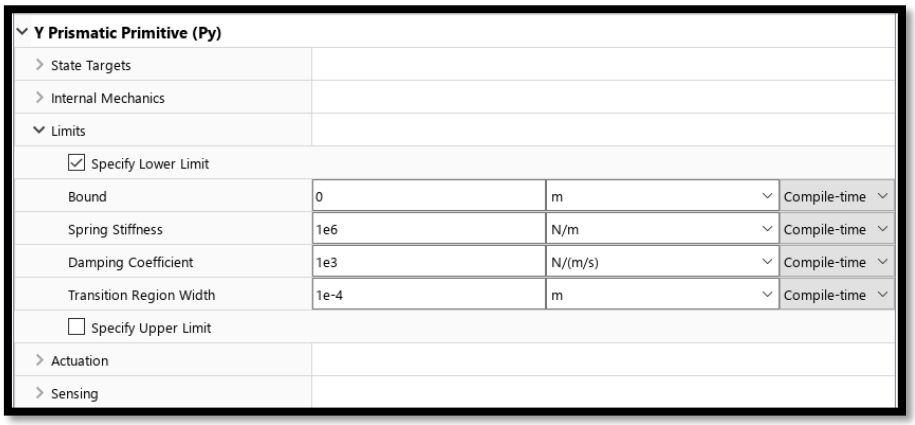
Étapes de mise en œuvre

1. **Remplacement par un Planar Joint** :

   - Le **Revolute Joint** est remplacé par un **Planar Joint** :

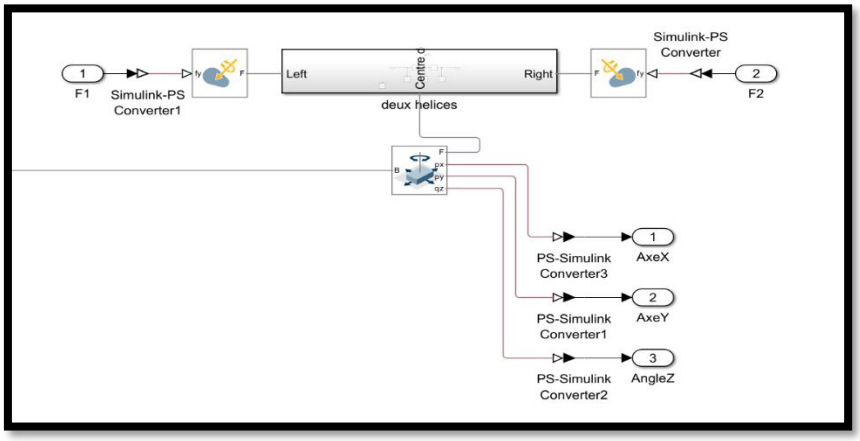


   - Dans les paramètres du **Planar Joint**, nous activons l'option **Specify Lower Limit** dans la section **Y Prismatic Primitive (Py)** et définissons la limite inférieure à **0** dans le champ **Bound** :



   Raison : Cela garantit que le drone ne peut pas se déplacer en dessous du niveau 0 sur l'axe vertical, simulant ainsi une contrainte physique réaliste.

   - Ensuite, pour chaque axe *X,Y,Z* nous activons la détection des positions dans la section **Sensing** pour afficher les positions de l'axe *X, Y et Z*. Ces sorties sont connectées aux blocs **Scope1-3** pour visualiser les graphiques des positions en fonction du temps :

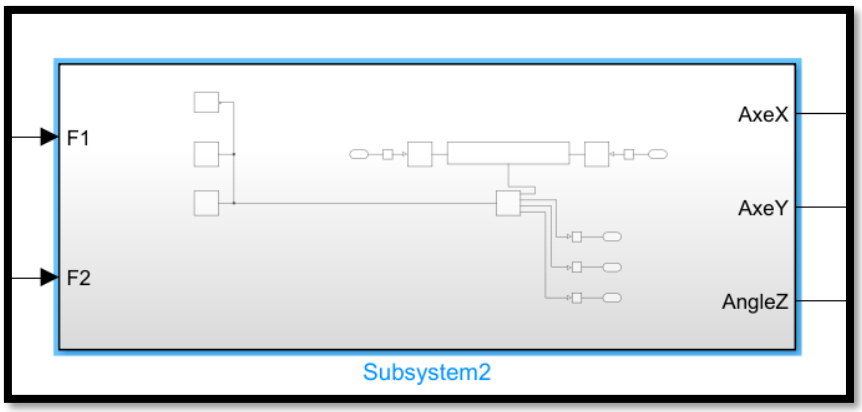


2. **Création du sous-système (Subsystem 2)** :

   - Un second sous-système est créé pour organiser les connexions internes.
   - Les trois sorties des axes (*AxeX,AxeY,AngleZ*) sont reliées à des ports pour les afficher au niveau de la structure principale.

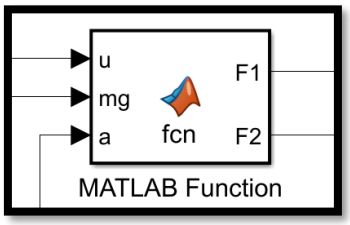
   Gestion des forces externes (F1 et F2) :

   - Les blocs **Constant**, qui étaient connectés directement aux **External Force and Torque**, sont supprimés.
   - Les forces *F1* et *F2* sont extraites du sous-système Subsystem2 à travers des ports nommés *F1* et *F2* :



3. **Ajout d'une fonction MATLAB pour calculer F1 et F2** :

   - Une fonction **MATLAB Function** est ajoutée au modèle principal :



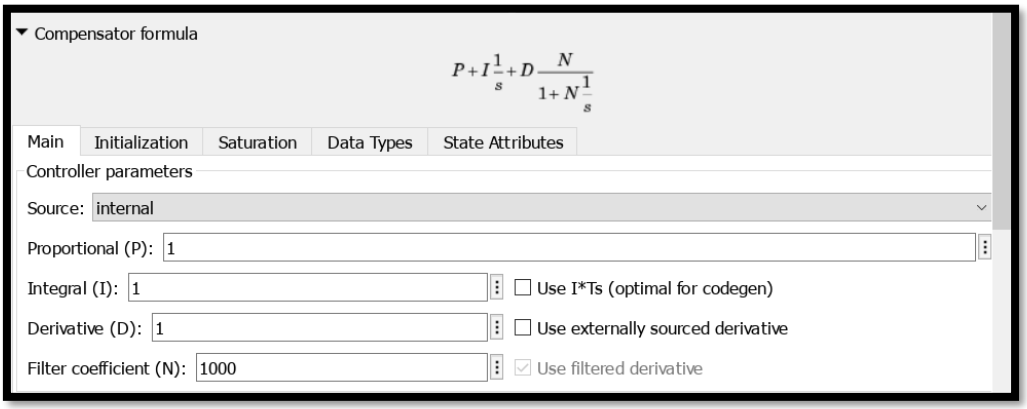
   - Le code de la fonction est le suivant :

```
function [F1,F2] = fcn(u,mg,a)
%#codegen
F2 = (u + mg) / (2 * cos(a));
F1 = (mg - u) / (2 * cos(a));
u = F1 - F2;
mg = F1 + F2;
```

   - Cette fonction prend en entrée uuu, mgmgmg, et aaa (angle d'inclinaison) et retourne *F1* et *F2*. Les sorties *F1* et *F2* sont reliées aux ports correspondants du sous-système.

4. **Implémentation du régulateur PID** :

   - Un régulateur PID est ajouté pour contrôler les mouvements en fonction des consignes.
   - Les gains *kP*,*kI*, et *kD* du régulateur sont définis à **1** chacun dans cette configuration initiale :



5. **Ajout d'une constante pour la masse gravitationnelle** :

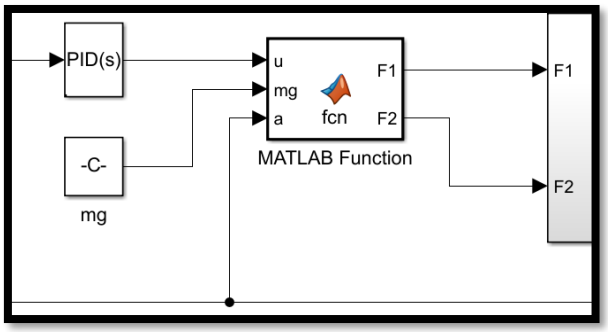
   - Une **Constant Block** est ajoutée avec une valeur correspondant au poids du drone (mg=0.2×9.80665).

6. **Bloc Consigne pour le contrôle de l'angle** :

   - Une autre **Constant Block** est ajoutée sous le nom de **Consigne**, pour permettre de définir l'angle souhaité pour le déplacement du drone. Cette consigne est connectée au régulateur PID.
   - Elle servira de référence pour ajuster l'inclinaison du drone.

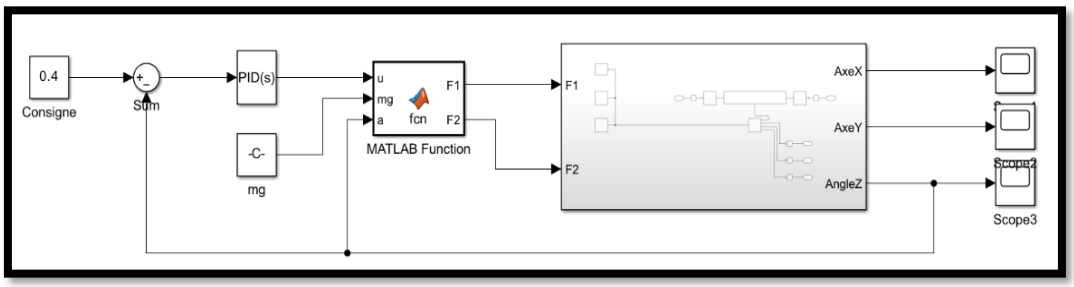
7. **Connexion de l'angle (a) à *AngleZ*** :

   - Le port *AngleZ* du sous-système est connecté à l'entrée aaa de la fonction MATLAB. Cela garantit que lorsque l'angle *AngleZ*=0, le cosinus dans le code est égal à 1 :



8. **Ajout d'un bloc Somme pour l'erreur** :

   - Un bloc **Sum** est ajouté avec une configuration de signe (+ et −).
   - Le port **AngleZ** est connecté à l'entrée **moins (-)**, tandis que la **Consigne** est connectée à l'entrée **plus (+)**.
   - La sortie du bloc somme (erreur) est reliée à l'entrée du PID, permettant de calculer la différence entre la consigne et l'angle actuel:



Résultat attendu

Cette configuration permet de simuler le mouvement du drone selon l'axe xxx, avec un contrôle précis de son angle d'inclinaison via le régulateur PID. Les graphiques des positions AxeX, AxeY et de l'angle AngleZ peuvent être visualisés pour analyser les performances du système.

Partie 2.2 : Intégration du deuxième PID régulateur pour le déplacement sur l'axe X

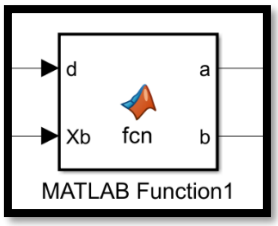
Objectif

Cette étape vise à ajouter un second régulateur PID pour contrôler la distance parcourue par le drone sur l'axe *X*, ainsi qu'à intégrer une deuxième fonction MATLAB pour calculer les ajustements nécessaires pour atteindre une consigne donnée.

Étapes de mise en œuvre

1. **Création d'une deuxième fonction MATLAB** :

- Une fonction **MATLAB Function1** est ajoutée dans la configuration principale :



- Voici le code de la fonction :

```
function [a, b] = fcn(d , Xb)
D = Xb - d;
a = 0;
if D == Xb
   a = 0;
end
if D < 0
   a = -0.00001*(-D);
   if  a < -0.25
      a = -0.25;
   end
end
if D > 0
   a = 0.00001*(D);
   if  a > 0.25
      a = 0.25;
   end
end
b = a;
%mg=F1+F2
```

Explication du code :

- ***D=Xb−d*** : Calcul de l'écart entre la position actuelle (*d*) et la position cible (*Xb*).

- ***a* = 0** : Si la différence entre *d* et *Xb* est nulle, aucune correction n'est nécessaire.
- **Cas où *D<0*** :

  Si la position actuelle d est supérieure à la position cible *Xb*, une valeur négative est attribuée à aaa, ajustée proportionnellement à *D*.

  Une limite minimale est fixée à −0.25 pour éviter des ajustements trop brusques.
- **Cas où *D>0*** :

  Si la position actuelle *d* est inférieure à *Xb*, une valeur positive est attribuée à aaa, également ajustée proportionnellement à *D*.

  Une limite maximale est fixée à 0.25.
- **Sorties** :

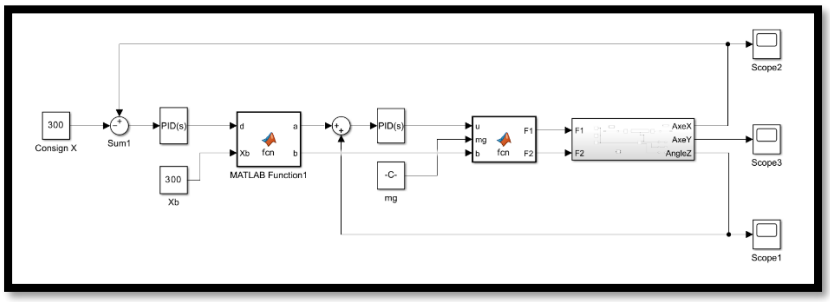
  a est utilisé pour ajuster la commande actuelle en fonction de l'écart *D*.

  b, qui est égal à aaa, est transmis à la première fonction MATLAB pour inclure cet ajustement dans les calculs des forces *F1* et *F2*.

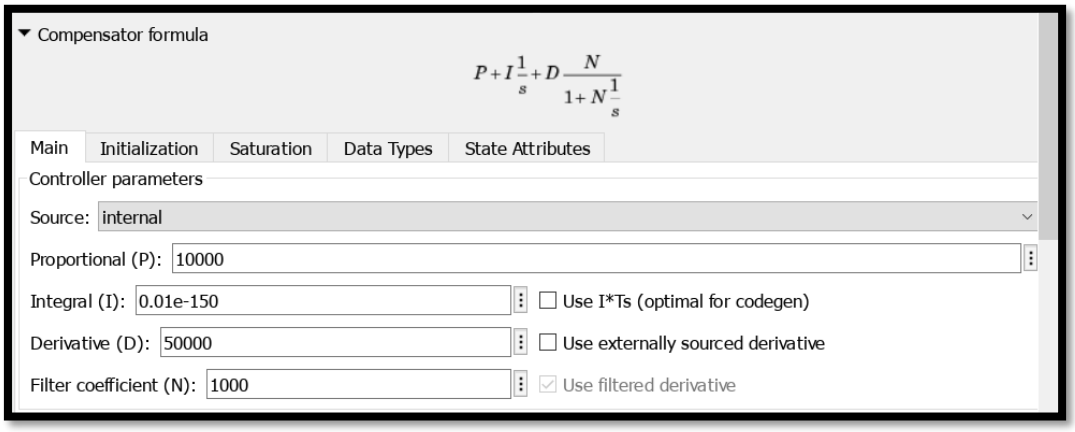
Modification de la première fonction MATLAB :

- Le paramètre *b* est ajouté en tant que nouvelle entrée :

  function [F1, F2] = fcn(u, mg , b)
- La sortie *b* de la deuxième fonction est connectée en tant qu'entrée à la première fonction :

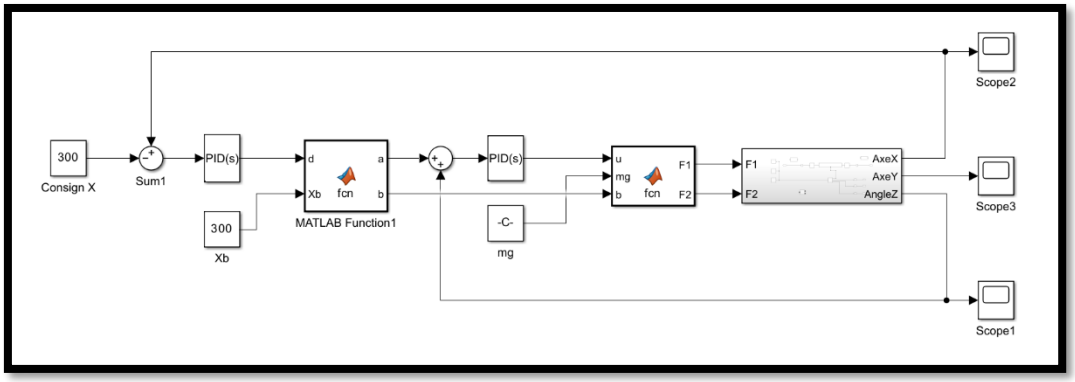


2. **Ajout du deuxième régulateur PID** :
   - Un second régulateur PID est ajouté, et ses sorties sont connectées à l'entrée *d* de la deuxième fonction MATLAB.
   - Les gains sont configurés comme suit :



- $kP = 10000$
- $kI = 0.01e{-}150$
- $kD = 50000$

3. **Création de la constante Xb** :
   - Une **Constant Block** est ajoutée avec une valeur de $Xb = 300$, représentant la distance cible que le drone doit atteindre.
   - Cette constante est connectée à l'entrée *Xb* de la deuxième fonction MATLAB.
4. **Ajout d'un bloc Consigne X** :
   - Une **Constant Block** nommée "Consigne X" est ajoutée dans la configuration principale.
   - Elle est utilisée pour spécifier la consigne de distance que le drone doit atteindre sur l'axe *X*.
   - La sortie de "Consigne X" est reliée à l'entrée du second PID.
5. **Ajout d'un second bloc Sum** :
   - Un bloc **Sum** est configuré avec les signes $-+$.
   - Les connexions sont réalisées comme suit :



   - L'entrée + est connectée au port *AxeX* du sous-système pour récupérer la position actuelle sur l'axe *X*.
   - L'entrée − est connectée à la sortie de "Consigne X".
   - La sortie du bloc **Sum**, correspondant à l'erreur entre la consigne et la position actuelle, est connectée au second PID.

Résultat attendu

Avec cette configuration, le drone peut maintenant ajuster sa position sur l'axe ***X*** en fonction d'une consigne donnée. Le régulateur PID et la fonction MATLAB garantissent que les ajustements sont proportionnels et limités pour maintenir un contrôle stable. Les graphiques des positions (***AxeX,AxeY***) et de l'angle (***AngleZ***) permettent d'évaluer les performances du système.

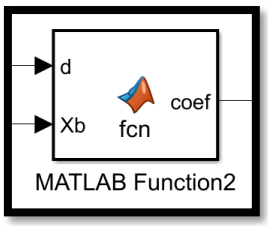
Partie 3 : Ajouter un degré de liberté sur l'axe vertical (axe Y) avec contrôle PID

Objectif

Dans cette partie, nous intégrons un troisième régulateur PID pour contrôler la position verticale du drone. Ce régulateur permettra au drone de maintenir ou d'atteindre une certaine hauteur définie par une consigne donnée.

Étapes de mise en œuvre

1. **Création d'une deuxième fonction MATLAB** :
   - Une fonction **MATLAB Function2** est ajoutée dans la configuration principale :



   - Voici le code de la fonction :

```
function coef = fcn(d, Xb)
D = Xb - d;
coef = 1;
if D == Xb
   coef = 1;
end
if D < 0
   coef = 1 - 0.00001 * D;
   if coef < (1/4)
      coef = (1/4);
   end
end
if D > 0
   coef = 1 + 0.00001 * D;
   if coef > (3/4)
      coef = (3/4);
   end
end
```

Explication du code :

- ***D=Xb−d*** : Cette ligne calcule la différence entre la position actuelle *d* et la position cible *Xb*.
- ***coef* = 1** : Si la position actuelle correspond à la consigne, le coefficient reste à **1** pour indiquer un état neutre.
- **Si *D*<0** :
  - *coef* diminue proportionnellement à *D*, indiquant que le drone doit ajuster sa hauteur en descendant. Une limite minimale est fixée à 1/4 pour éviter des ajustements excessifs.

- **Si *D*>0** :
  - *coef* augmente proportionnellement à *D*, signalant que le drone doit monter. Une limite maximale est fixée à 3/4.

Sorties :

- *coef* est un multiplicateur utilisé pour ajuster les forces nécessaires au maintien ou au déplacement vertical.

2. **Modification de la première fonction MATLAB** :
   - Le paramètre *coef* est ajouté en tant que nouvelle entrée. Voici le nouveau code :

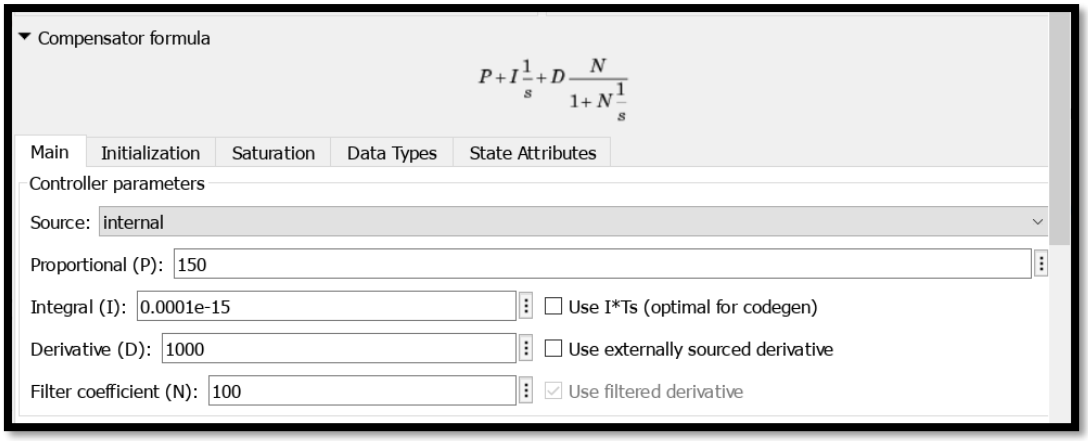
```
function [F1, F2] = fcn(u, mg, b, coef)
F2 = coef * ((mg - u) / (2 * cos(b)));
F1 = coef * ((mg + u) / (2 * cos(b)));

if coef == 3/4 || coef == 1/4
   F2 = coef * ((mg - u) / (2 * cos(0.2)));
   F1 = coef * ((mg + u) / (2 * cos(0.2)));
  % avec cos(b) = cos(-b)
End
```

Explication du code :

- coef ajuste les forces *F1* et *F2* en fonction de l'écart vertical.
- Les conditions supplémentaires garantissent que si *coef* atteint ses limites (1/4 ou 3/4), un angle constant de 0.2 est utilisé pour éviter des calculs instables.

3. **Ajout du deuxième régulateur PID** :
   - Un troisième régulateur PID est ajouté, et ses sorties sont connectées à l'entrée *d* de la troisième fonction MATLAB :

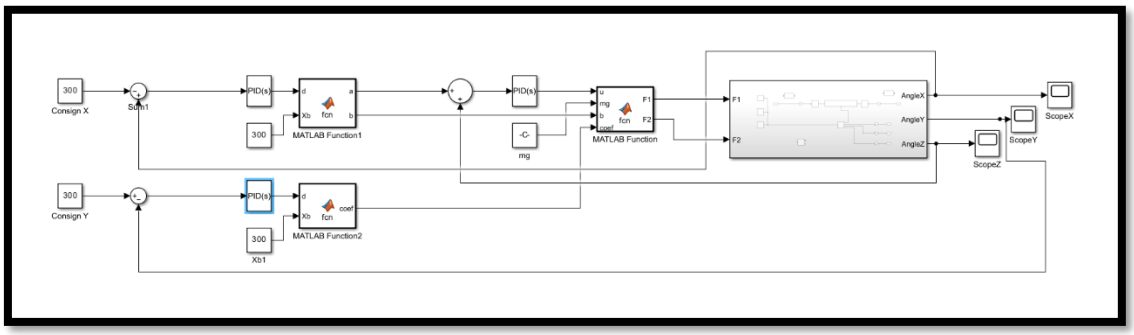


   - Les paramètres sont configurés comme suit :
     - *kP*=150
     - *kI*=0.0001e−15
     - *kD*=1000.

4. **Création de constantes pour *Xb*** et *ConsigneY* :
   - Une constante Xb=300Xb = 300Xb=300 est ajoutée pour représenter la hauteur cible du drone.

- Une autre constante, nommée "Consigne Y", est ajoutée pour définir la consigne verticale.

5. **Ajout d'un bloc Sum pour l'axe Y** :
   - Un bloc **Sum** est configuré pour calculer la différence entre la consigne et la position actuelle sur l'axe *Y*.
   - Les connexions sont réalisées comme suit :



   - L'entrée + est connectée au port *AxeY*.
   - L'entrée − est connectée à la sortie de "Consigne Y".
   - La sortie du bloc Sum, correspondant à l'erreur entre la consigne et la position actuelle, est reliée au troisième PID.

Résultat attendu

Avec ce régulateur PID supplémentaire et la fonction MATLAB associée, le système est maintenant capable de contrôler la position verticale du drone. Cela complète le contrôle tridimensionnel (***X,Y,Z***) en ajoutant un degré de liberté vertical.