Les objectifs de ce projet sont les suivants :

Le but principal est de savoir mettre en place un système Cyber-Physique en comprenant ce qu'est la commande des systèmes, leur conception, leur forme et les paramètres importants.

Nous voulons également être capable de mettre en place un dispositif numérique de commande tout en structurant le code de ces systèmes, de traduire le comportement de systèmes commande en algorithmes.

Finalement nous voulons être capables de récupérer les mesures issues des capteurs et transmettre les commandes aux actionneurs pour mettre en place une communication entre les sous-systèmes. Nous allons donc faire déplacer un robot en lui donnant une position à atteindre.

Contraintes : le véhicule ne peut pas tourner au-delà de 300/l et -300/l avec l la distance entre la chenille et le robot, soit au-delà de 3,3° pour la vitesse angulaire $\dot{\theta}$, la tension ne doit pas excéder les 9V sinon on arrive en saturation.

Pour ce faire, dans un premier temps nous allons réaliser une simulation du système décomposé en plusieurs parties :

On commence par réaliser l'asservissement des moteurs gauches et droits.

Ci-dessous la modélisation des moteurs :

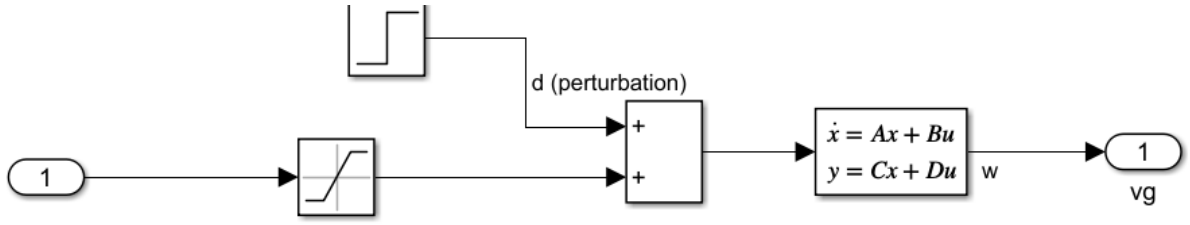

Figure 1 : Modélisation des moteurs

Ici dans nos test nous n'ajouterons pas de perturbations. Les valeurs de nos vecteurs A,B,C et D ont été préalablement définies dans notre fichier script Matlab.

Pour l'asservissement des moteurs, on rend le système commandables en intégrant les gains Ki et Kx dans notre boucle.

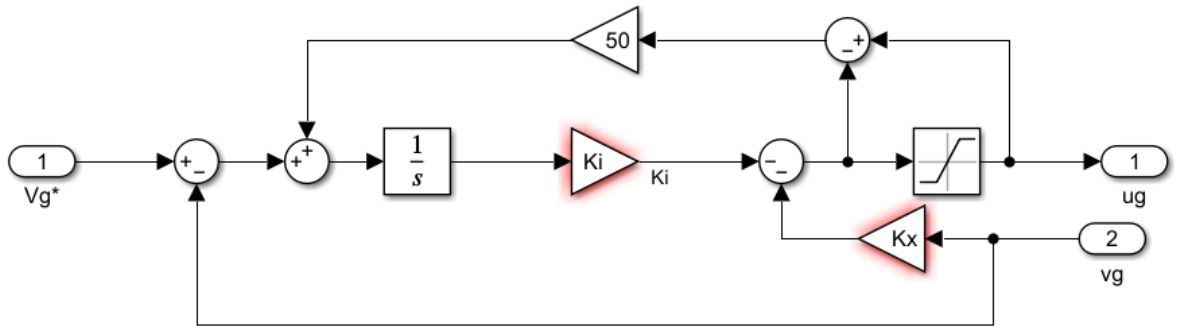

Figure 2 : Asservissement des moteurs

Afin d'obtenir la vitesse et l'orientation à partir de notre position, il nous faut réaliser l'inverse de la matrice M(Θ) pour linéariser la commande.

M(Θ)= $\begin{pmatrix} C\theta & -Ld\theta \\ D\theta & Lc\theta \end{pmatrix}$ det(M(Θ))=L donc la matrice est toujours inversible.

On donne en entrée ux et uy les vitesses en x et y commandées.

On peut alors poser le système suivant : $\begin{pmatrix} ux \\ uy \end{pmatrix}$= M(Θ) $\begin{pmatrix} v \\ \dot{\theta} \end{pmatrix}$ par changement de variable on obtient alors $\begin{pmatrix} ux \\ uy \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}$

Voici l'implémentation de l'inverse dans notre système avec M(Θ)= $\begin{pmatrix} \cos(Teta) & -210\sin(Teta) \\ \sin(Teta) & 210\cos(Teta) \end{pmatrix}$

On obtient alors la matrice simplifiée M(Θ)^-1= $\begin{pmatrix} \cos(Teta) & \sin(Teta) \\ -\frac{\sin(Teta)}{210} & \frac{\cos(Teta)}{210} \end{pmatrix}$

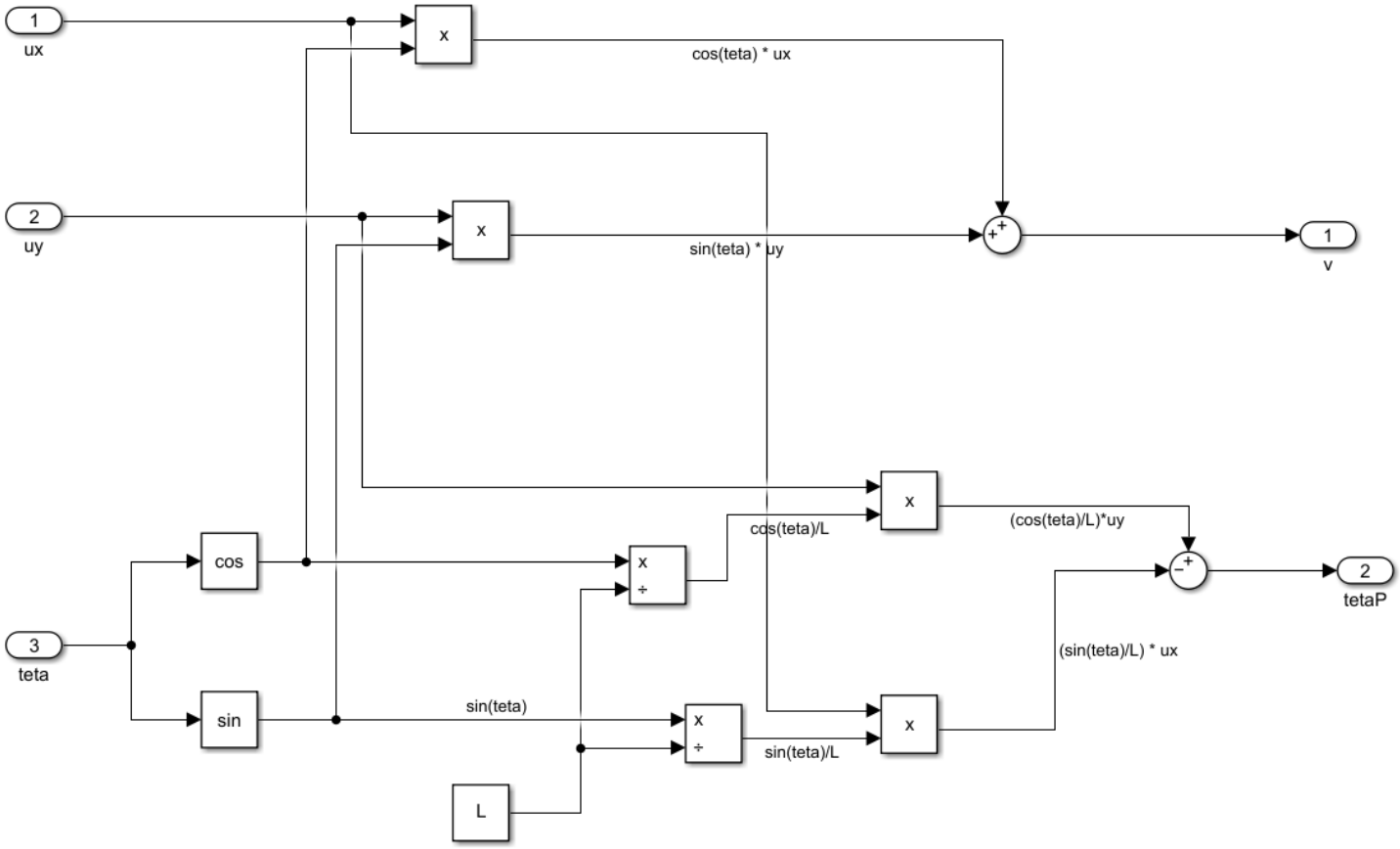

Figure 3 : Inversion de la matrice M(Θ)

On a maintenant une vitesse et une orientation, on veut maintenant obtenir une vitesse pour chaque moteur, on va donc faire un calcul de vitesse.

On intègre la saturation qui nous permet de ne pas dépasser les 9V, on lui demande donc de ne pas dépasser les 300mm/s-1.

Pour le calcul de vitesse, on va utiliser les équations données :

$$vd = v + l\dot{\theta}$$

$$vg = v - l\dot{\theta}$$

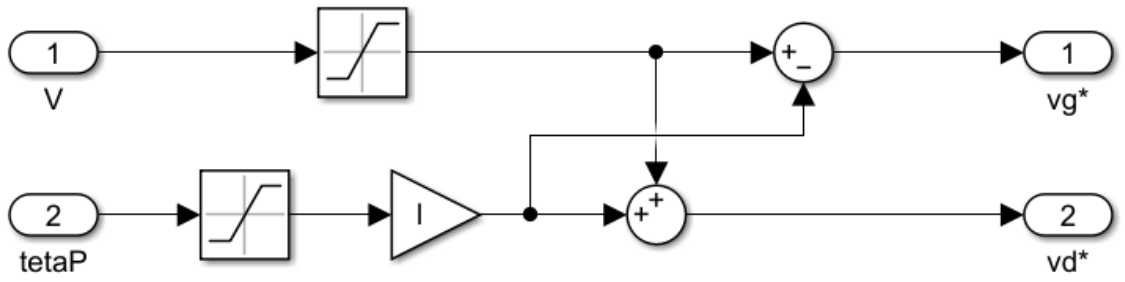

Figure 4 : Calcul de vitesse

Après avoir linéarisé la commande, calculé la vitesse du système et asservit les moteurs, on peut considérer les moteurs comme parfait d'un point de vue cinématique.

On reprend le système d'équation précédent.

On peut simplifier ça en additionnant les 2 systèmes d'équations.

On obtient alors : $v = \frac{vg + vd}{2}$ et $\dot{\theta} = \frac{vg - vd}{-2l}$

Voici comment nous l'avons réalisé dans notre simulation :

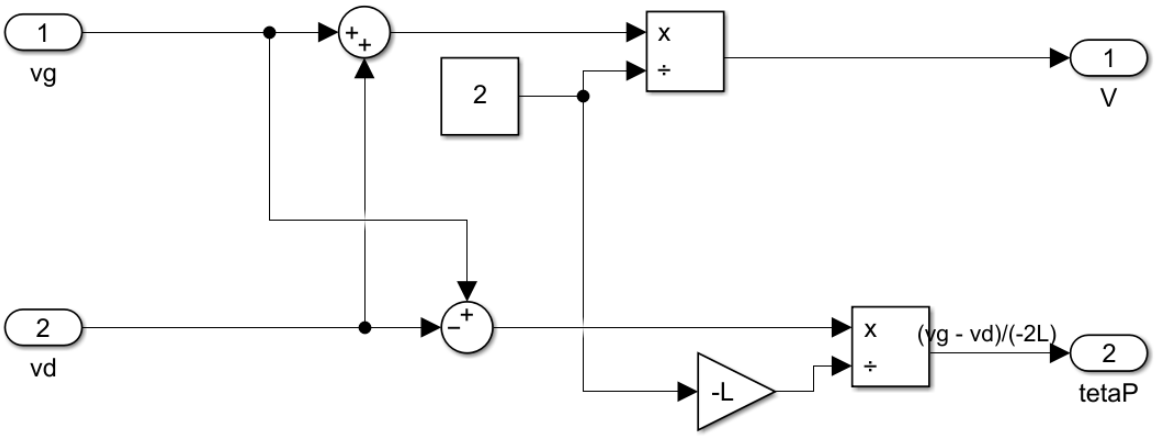

Figure 5 : Moteur cinématique

Finalement, voici le système global simulé :

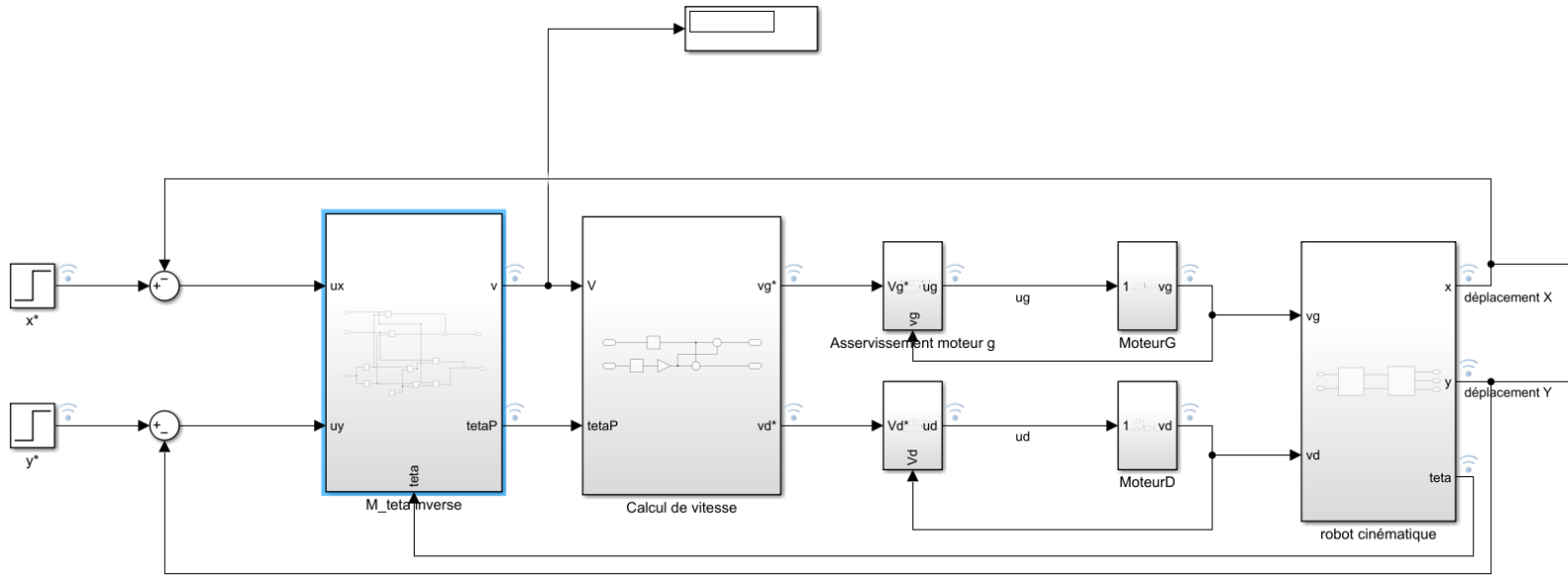

Figure 6 : Simulation complète du système via Simulink

On donne en entrée les positions x et y qui nous permettent de calculer les vitesses et angles à donner à notre robot, on envoie les vitesses souhaitées pour les moteurs gauches et droits dans le système d'asservissement moteur qui nous donne en sortie une tension à appliquer à chaque moteur.

D'un point de vu graphique, voilà comment notre simulation se traduit, ici nous donnons en paramètre $\dot{x} = 1000\ et\ \dot{y} = 500$ pour vérifier que le robot peut tourner dans la simulation, on lui demande d'aller à 1 mètre tout droit et 50 centimètres à gauche.

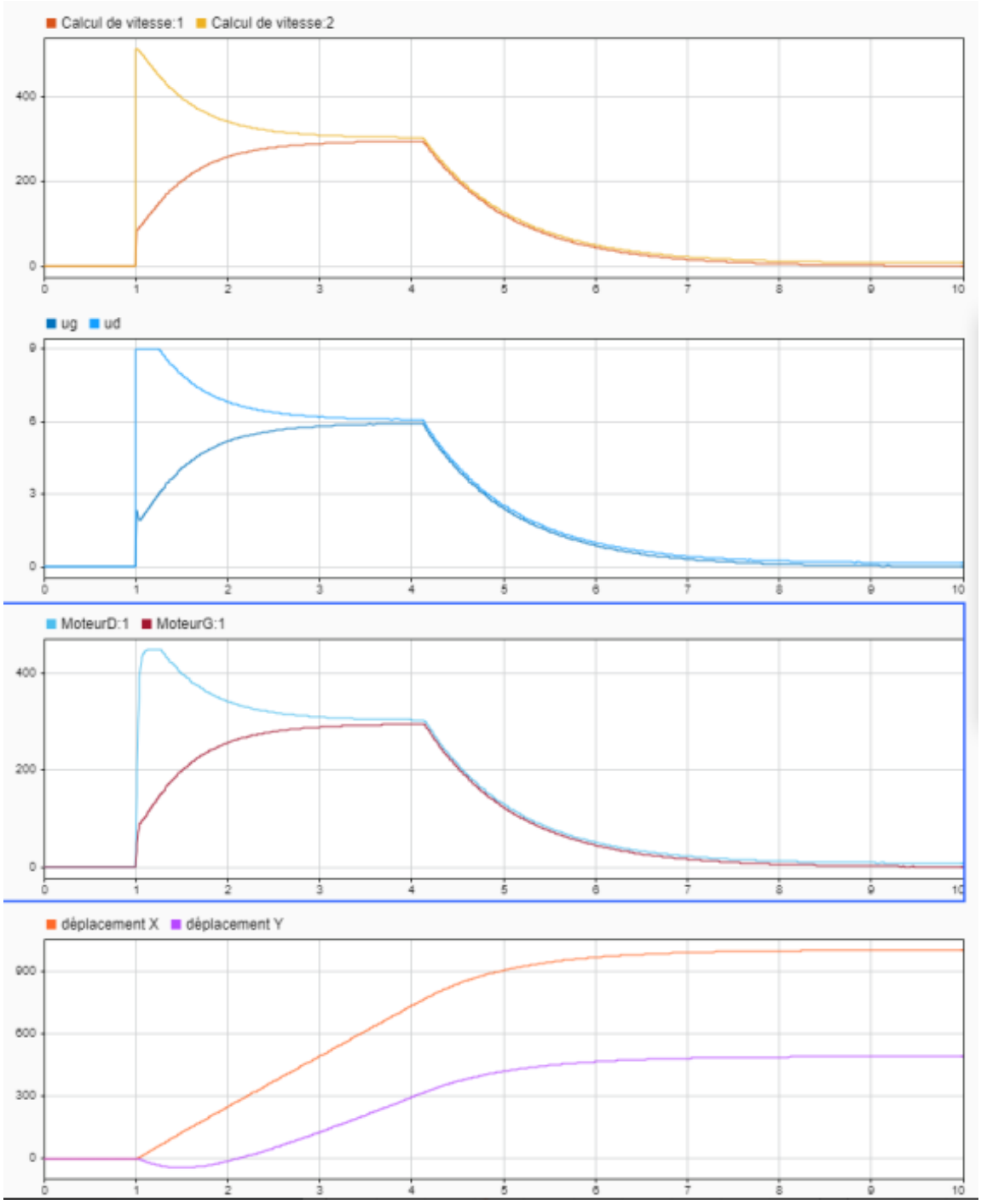

Figure 7 : Représentation graphique obtenues par la simulation

On voit bien que nos courbes sur les diagrammes de calculs de vitesse, de tension et de vitesse des moteurs sont les mêmes.

On peut vérifier que la vitesse du moteur droit augmente rapidement puisqu'on lui demande de tourner à gauche, et la vitesse du moteur gauche est moins importante. Les 2 moteurs doivent s'arrêtent en même temps une fois arrivé au point demandé.

On visualise également l'angle sur le graphique représentant M(Θ)^-1, on voit qu'il tourne rapidement avant de retrouver un angle nul.

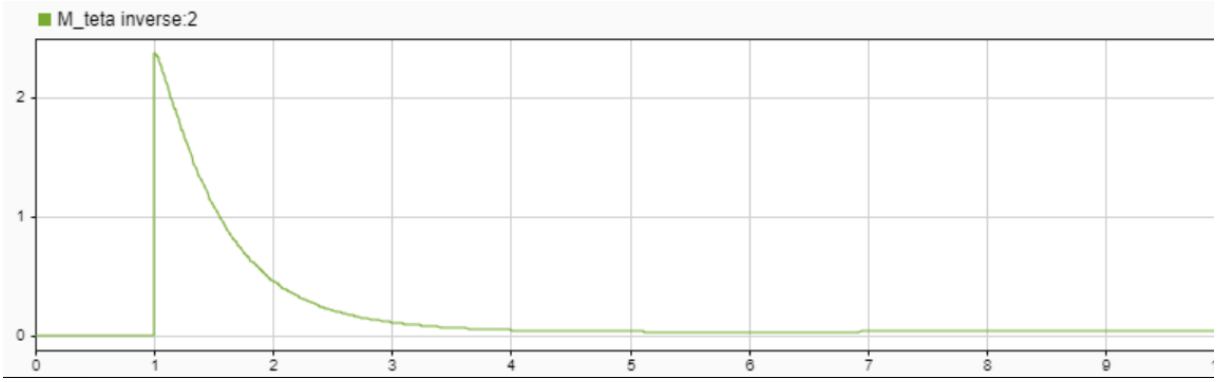

Figure 8 : Représentation graphique de l'angle en sortie de M(Θ)^-1 avec les entrée x=1000 et y=500

Finalement, on peut voit sur le dernier graphique les déplacement en x et y du robot où on constate définitivement qu'il tourne.

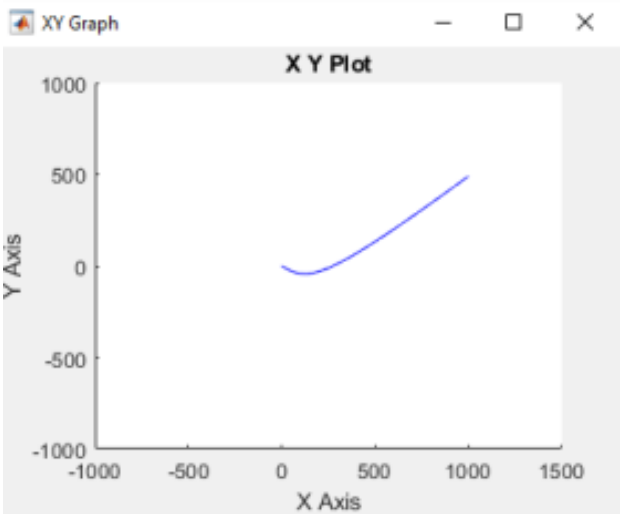

Figure 9 : Représentation graphique des déplacements en x et y

Au niveau des critères, notre objectif est d'avoir une précision à 10% près, nous n'avons pas définit de critère de robustesse mais avons mis en place un anti wind up pour une fiabilité maximale.

Nous pouvons alors générer une partie du code c à partir de notre simulation afin de contrôler notre robot.

Au niveau du matériel fourni, nous travaillerons avec un Arduino MEGA qui sera connecté aux moteurs et à l'encodeur qui seront eux-mêmes connectés aux GPIO.

Nous aurons également un gyroscope et un accéléromètre qui nous permettrons de gérer la vitesse de rotation et l'accélération en translation depuis les ports I2C.

Finalement, le Raspberry PI sera connecté en UART2 à l'Arduino.

On doit dans un premier temps être capable de récupérer les informations du marvelmind depuis le Raspberry Pi, on doit ensuite pouvoir communiquer entre notre Raspberry et l'Arduino.

Ci-dessous, un schéma d'architecture pour expliquer la communication entre Raspberry, Arduino les moteurs et le marvelMind.

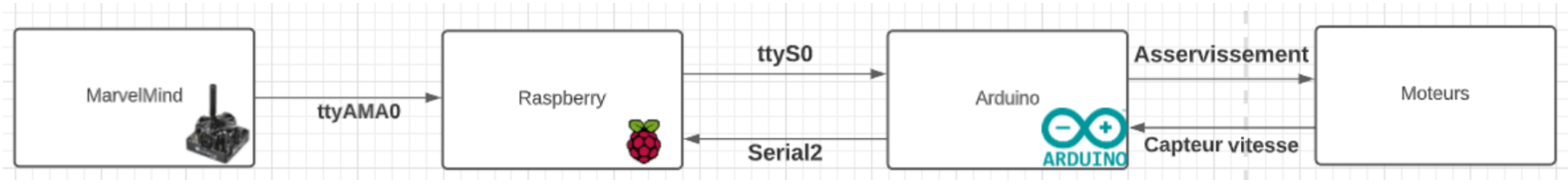

Figure 10 : Schéma d'architecture physique expliquant la communication entre les différents modules

Dans notre code python, nous voulons récupérer les coordonnées x, y et z du robot par rapport au marvelmind depuis le Raspberry. Ci-dessous, le résultat obtenu :

```
open serial port: /dev/ttyACM0
port opened
59: X: 2.575, Y: -1.296, Z: -0.697, Angle: 0 at time T: 368.325
59: X: 2.574, Y: -1.295, Z: -0.694, Angle: 0 at time T: 369.863
59: X: 2.575, Y: -1.294, Z: -0.691, Angle: 0 at time T: 371.400
59: X: 2.573, Y: -1.302, Z: -0.701, Angle: 0 at time T: 372.937
59: X: 2.577, Y: -1.300, Z: -0.694, Angle: 0 at time T: 374.474
59: X: 2.574, Y: -1.313, Z: -0.712, Angle: 0 at time T: 376.011
59: X: 2.575, Y: -1.308, Z: -0.707, Angle: 0 at time T: 377.548
59: X: 2.574, Y: -1.301, Z: -0.708, Angle: 0 at time T: 379.085
59: X: 2.578, Y: -1.303, Z: -0.713, Angle: 0 at time T: 380.622
59: X: 2.574, Y: -1.302, Z: -0.718, Angle: 0 at time T: 382.159
59: X: 2.576, Y: -1.303, Z: -0.713, Angle: 0 at time T: 383.696
```

Figure 11 : Résultat obtenu à l'exécution du code python depuis le Raspberry récupérant les informations du marvelmind

Pour la communication entre le Raspberry et l'Arduino, voici le code Arduino réalisé. On commence par initialiser les ports puis on lit et envoie les informations en Serial2.

```
void setup() {
  Serial.begin(9600);
  Serial2.begin(9600);
}

void loop() {
  if (Serial.available()) {      // If anything comes in Serial (USB),
    Serial2.write(Serial.read());   // read it and send it out Serial1 (pins 0 & 1)
  }

  if (Serial2.available()) {     // If anything comes in Serial1 (pins 0 & 1)
    Serial.write(Serial2.read());   // read it and send it out Serial (USB)
  }
}
```

Figure 12 : Code C Arduino implémenté pour la communication avec le Raspberry

Du côté Raspberry, voici le code implémenté :

```
# Ce programme permet d'entrer un message du côté raspberry et de le lire du côté arduino
import serial, time

# Connecte le port serie a l'arduino
with serial.Serial("/dev/ttyS0", 9600, timeout=.1) as arduino:
    time.sleep(0.1)
    # Si le port est accessible
    if arduino.isOpen():
        print("{} connected".format(arduino.port))
        try:
            # On boucle pour ecrire dans le bus uart de données
            while True:
                entree = input("Valeur a envoyer")
                # on ecrit la valeur saisie dans l'input
                arduino.write(entree.encode())
        except KeyboardInterrupt:
            print("Controle c pressé")
```

Figure 13 : Code Python exécuté depuis le Raspberry pour communication avec Arduino

On peut ensuite asservir les moteurs avec un code Arduino, il nous faut au préalable charger les librairies TimerFive et la Librairie MakeBlock.

On peut déjà visualiser l'angle avec le gyroscope dans le moniteur en passant en Bode 115200.