Dossier technique

Projet PROTIS

Luiz Evaristo Da Silva

Enola Koenig

Aimé Matheron

Lilian Thevenin

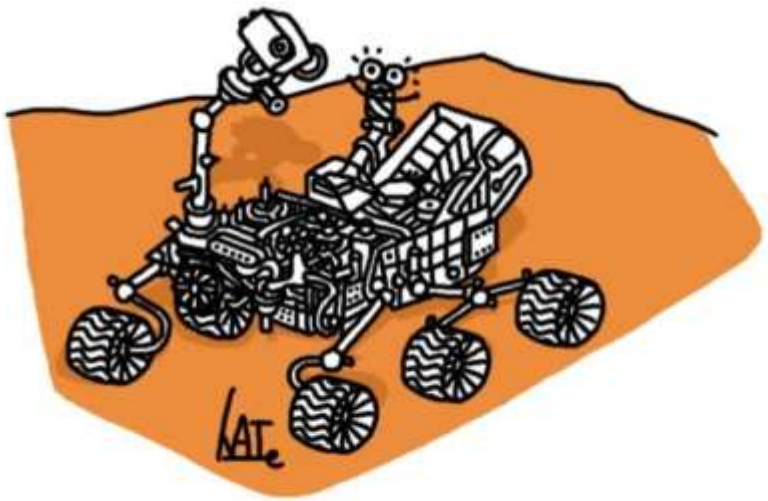
Camille Verdier

Table des matières

Introduction :

Le projet consiste à développer un robot guidé à distance pour la société SOLEC. Ce robot est destiné à être envoyé sur Mars afin d'explorer ses sols et détecter la présence d'eau. Il doit être capable de se déplacer en ligne droite et dans des virages selon des directives qui lui sont transmises depuis la Terre grâce à une interface Humain-Machine. Lors de son déplacement, il doit relever des données d'humidité et de température à l'aide de capteurs qui lui sont implantés et renvoyer ces données à intervalle régulier.
Ce robot doit également respecter certaines contraintes pour répondre aux performances demandées par l'utilisateur, notamment au niveau de sa rapidité (vitesse entre 10cm/s et 30cm/s) et de sa fiabilité (erreur de position maximale de 2 cm et d'angle de 3°). Il est aussi nécessaire que ce robot soit autonome sur un parcours de 1km et que l'interface Humain-Machine qui lui est associée soit facile d'utilisation.

En raison de la situation actuelle de pandémie de Covid-19, notre équipe, constituée de Luiz, Enola, Aimé, Lilian et Camille, a dû adapter la problématique. Il s'agit maintenant d'effectuer une modélisation du moteur à courant continu utilisé dans le robot et de l'asservir en choisissant les bons correcteurs afin de respecter le cahier des charges. D'un autre côté, il faut mettre en place l'interface Humain-Machine capable d'envoyer des trains de données à la modélisation du moteur. Les deux composantes du projet seront alors associées afin que le robot soit effectivement contrôlé à distance. Nous n'avons pas pu prendre en compte les mesures d'humidité et de température dans ce modèle numérique. Ce seront les seuls paramètres à ajouter en séance présentielle car l'interface sera au point et le moteur asservi. Il suffira donc finalement de monter le moteur sur la structure du robot, ajouter les capteurs et leurs codes de fonctionnement et ajouter une batterie pour l'autonomie du robot.



Selfies sur Mars / Source : https://undessinparjour.wordpress.com/

I- Architecture du prototype

Notre prototype est composé d'un moteur à courant continu modélisé sur Simulink. Ce dernier a été asservi en position afin de modéliser le déplacement d'une roue du robot. Il reçoit les données de son parcours par l'intermédiaire d'une interface Humain-Machine effectuée sur Matlab. Il doit alors pouvoir se déplacer en respectant le cahier des charges. Un retour sur l'interface permet de traiter les données de position et de vérifier la bonne tenue des performances attendues. On présente Figure 1 les différentes fonctions du prototype ainsi que la répartition sur les différentes tâches qui permettront de le mettre en œuvre.

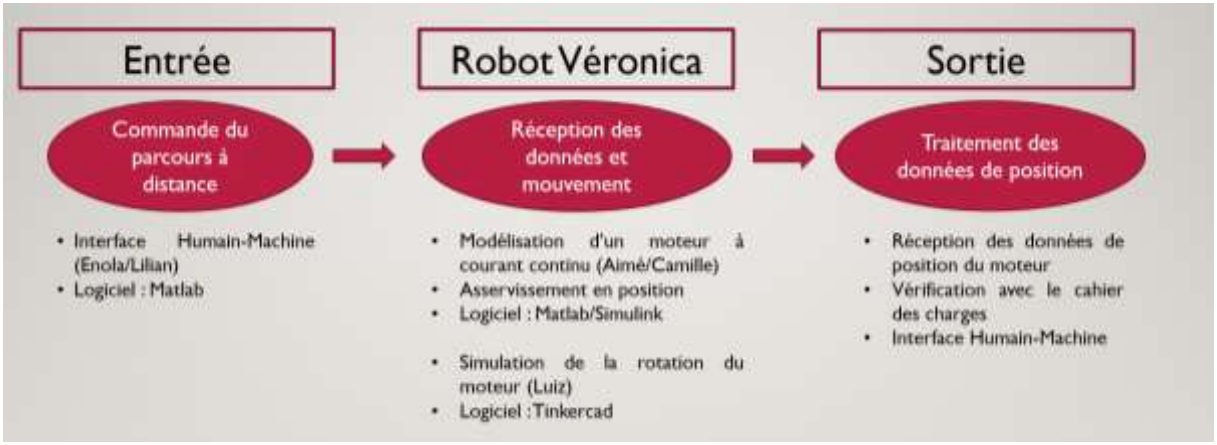
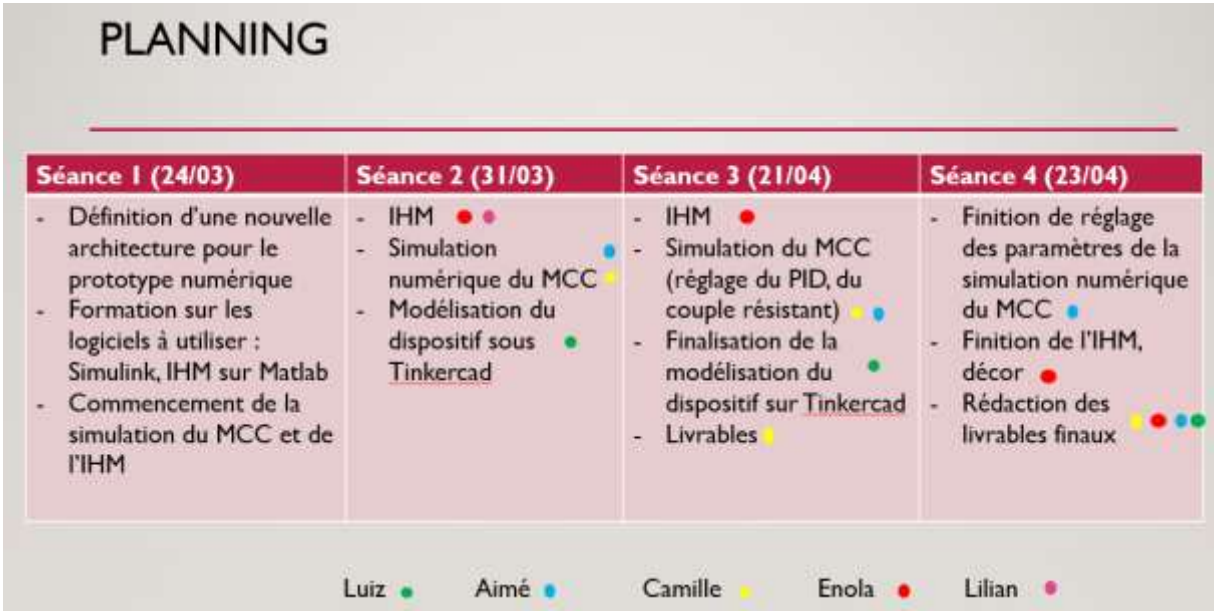


Figure 1 : Architecture du prototype

Pour réaliser notre prototype, nous avons suivi le planning suivant :



Nous avons utilisé la plateforme ***Basecamp*** comme moyen de communication dans laquelle nous avons pu nous partager les ressources nécessaires à l'avancée du projet. Nous y avons également établi notre liste « To Do » ainsi que notre planning afin de toujours être à jour dans notre projet.

II- Simulation et fonctions réalisées

Ce projet nous a tout d'abord permis d'apprendre à nous adapter rapidement face à une situation inédite. Nous sommes passés de la construction d'un robot à une simulation numérique de celui-ci.

Simulation du moteur à courant continu avec boucle d'asservissement en position :

Concernant la simulation numérique du robot en lui-même et de ses fonctions, nous avons choisi de modéliser simplement le robot par un moteur à courant continu sous Simulink. Nous nous étions déjà servis de Simulink lors du cours d'automatique, nous avons donc repris ces bases pour avoir notre simple MCC. Nous avons ensuite effectué une boucle d'asservissement en position afin de respecter le cahier des charges et contrôler la position finale à une incertitude de ± 2 cm. Pour la modélisation du moteur CC, l'idée est de faire un modèle causal en reprenant simplement les équations électromécaniques du moteur et en se plaçant dans l'espace de Laplace. (Figure 2)

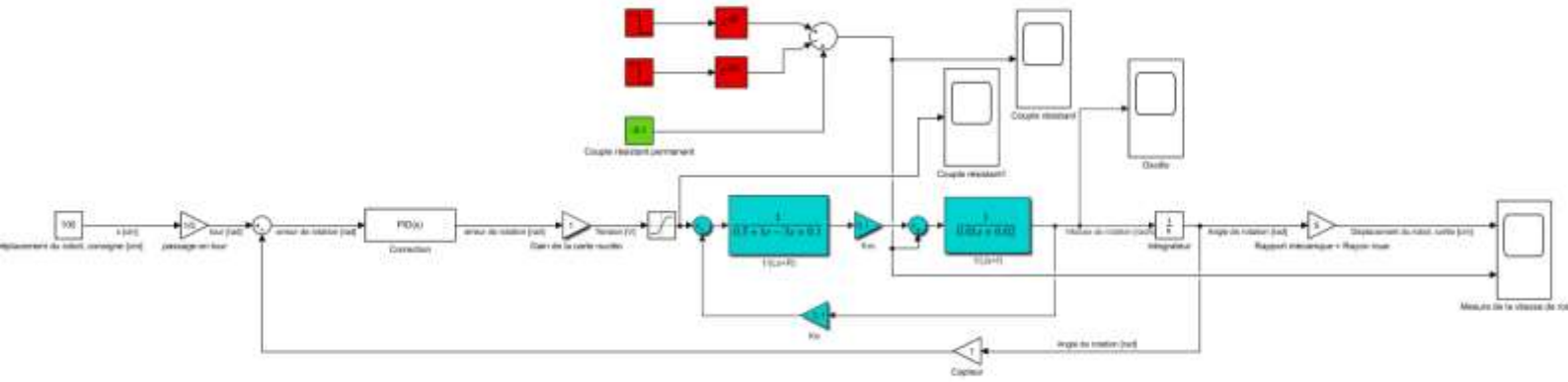


Figure 2 : MCC modélisé sous Simulink

On a coloré en bleu ce qui appartient au moteur, en rouge et vert respectivement le couple résistant perturbatif et permanent. On place ensuite un « seuil » de tension pour que celle-ci ne dépasse pas 3 Volts en valeur absolue. Cela est plus réaliste et entraine donc une limitation : la puissance fournie au moteur étant limitée, ce dernier ne pourra pas résister à toutes les valeurs de couple résistant. La limitation en tension induit ainsi une limitation en couple résistant. D'après les tests, un couple résistant total supérieur en valeur absolue à 0.8 N.m va entrainer le fait que le moteur ne pourra plus répondre à la consigne.

Concernant la simulation du couple résistant, on le simule grâce à deux parties : en vert, le couple résistant permanent qui impose au moteur l'équivalent d'une pente (si ce couple est négatif cela veut dire qu'il résiste contre le couple du moteur ce qui peut se traduire par le robot qui monte une pente, et inversement).

En rouge, le couple résistant perturbatif qui intervient au bout d'un certain temps et qui prends la forme d'un signal rectangle. Ce couple est présent pour qu'on puisse tester la robustesse du système à une perturbation. En effet, on ne détaillera pas trop la théorie sur ce point mais on peut néanmoins dire que, de façon générale, l'erreur globale du système s'écrit comme la somme de deux termes : $\varepsilon_{totale,s,v\ ou\ 3} = \varepsilon_{suivi\_de\_consigne} + \varepsilon_{erreur\_en\_régulation}$

L'erreur en suivi de consigne concerne uniquement la situation où la perturbation est nulle (à savoir un couple résistant égal à 0). Cette erreur quantifie l'aptitude du système à répondre à une consigne

donnée en l'absence de perturbation. Le deuxième terme, l'erreur en régulation, traduit l'aptitude du système à résister à une perturbation extérieure. Elle est mesurée pour une consigne nulle et une perturbation non nulle. Ici la perturbation est le couple résistant perturbatif et son intervention dans notre boucle de simulation est essentielle pour connaitre la réaction du système en erreur de régulation.

On donne ci-dessous le tracé temporel de l'évolution de la position du robot (Figure 3). La partie à gauche permet de voir si l'erreur en suivi de consigne est importante, la partie de droite permet de voir si l'erreur en régulation est importante (puisque le couple résistant perturbatif est introduit qu'à la 8ème seconde)

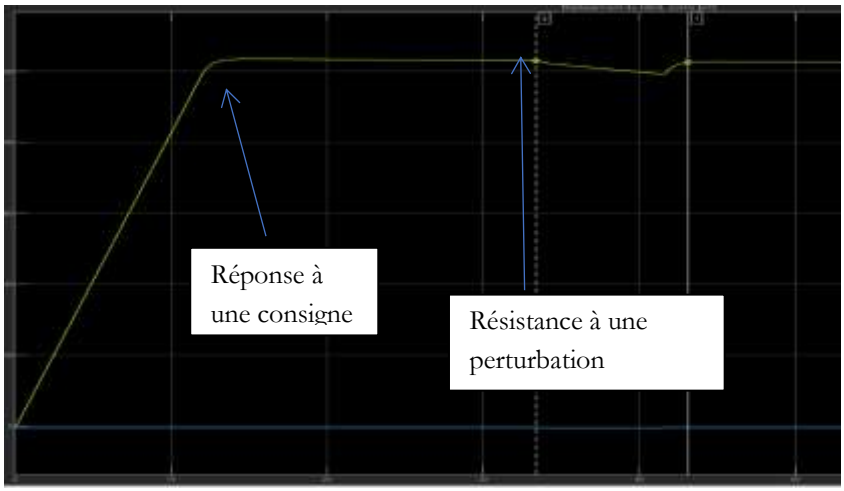


Figure 3 : Déplacement du robot et comportement face à une perturbation

Une fois le moteur complètement modélisé, on ajoute la boucle d'asservissement en position (donc avec un intégrateur en sortie du moteur pour transformer la vitesse de rotation en angle). On ajoute ensuite une correction PID afin que notre système réponde mieux au cahier des charges.

On a réglé le correcteur PID de la manière suivante : on ajuste le gain proportionnel de sorte à ce que, sans perturbation, le système réponde correctement à la consigne. On ajoute ensuite la perturbation et on règle le gain intégral de sorte à ce que le système de mette à pomper (oscillations auto-entretenues). On est alors dans une situation de limite de stabilité. On divise alors notre gain intégral par 4 pour faire en sorte de récupérer de la stabilité tout en maintenant l'action intégrale qui absorbe l'erreur statique en présence de perturbation. (Remarque : dans les méthodes classiques comme celle de Ziegler et Nichols, on divise plutôt par deux le gain intégral de pompage de sorte à obtenir le gain optimal, mais on a trouvé ici que cette valeur induisait une trop grande instabilité et, avec ça, un dépassement trop important de la valeur finale (de l'ordre de 20%). Le gain dérivé est ensuite ajusté de sorte à récupérer de la dynamique et de la stabilité sur le système. On a pris ici comme valeurs de gain : $\mathbf{K_P=1}$, $\mathbf{K_i=0.01}$ et $\mathbf{K_d=0.5}$.

Avec ces valeurs, on a un système stable qui résiste bien aux perturbations. On peut, pour vérifier si le système répond au cahier des charges en termes de marges de stabilité, tracer le diagramme de Bode de sa FTBO (fonction de transfert en boucle ouverte).

On voit sur la Figure 4 que l'action dérivée induit la bosse de phase ayant un maximum en $\omega_{max,bosse\_phase}$. Si on compare cette valeur à la valeur de ω_{0dB}, on peut facilement voir que le système n'est donc pas optimisé à 100%. En effet, si on avait réglé idéalement le correcteur PID, le maximum de la bosse de phase devrait correspondre à la pulsation à 0dB du gain, à savoir de

sorte à avoir une marge de phase maximum. Cependant, cela n'est ici pas grave puisque notre système reste quand même très stable. On donne en effet les marges de stabilité suivantes :

$$\boldsymbol{M_g = 16\,dB} \quad \text{et} \quad \boldsymbol{M_\varphi = 64\,°}$$

ce qui est largement compris dans le cahier des charges. Au contraire, de trop grandes marges de stabilité auraient entrainées une bande passante trop faible ce qui n'aurait pas été forcément bénéfique pour notre cas d'utilisation.

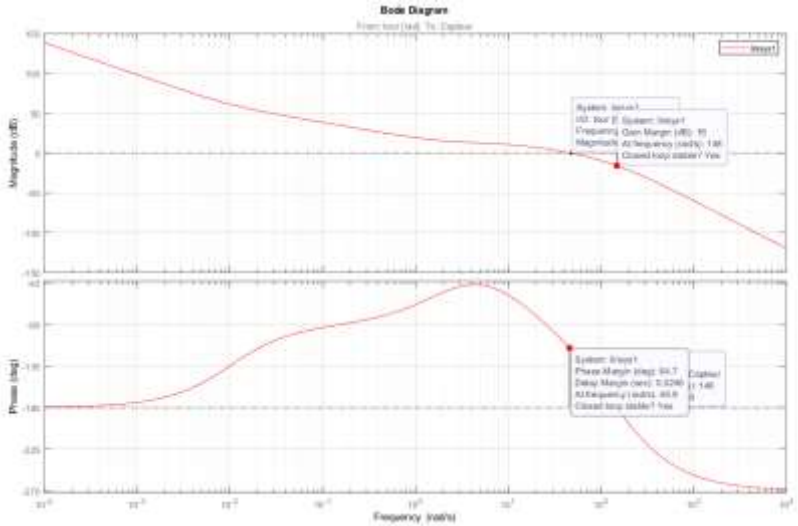


Figure 4 : Tracé du diagramme de Bode de la FTBO du système asservi.

On a enfin placé des blocs (tout à gauche et tout à droite du schéma Simulink) correspondants à des gains purs et représentant le rapport de réduction lié au rayon de la roue qui permet de passer d'une consigne en tour [rad] à une consigne en distance à effectuer [cm].

Concernant le capteur incrémental placé à l'arrière du moteur CC qui envoie un « tick » à chaque passage devant un trou et qui nous permet donc de récupérer l'information sur l'angle de rotation, nous avons simplement placé un gain proportionnel fixé arbitrairement. En effet, l'idéal aurait été de pouvoir éventuellement discrétiser cette partie du système puisque les « ticks » sont envoyés à intervalle de temps constant et représentent donc des valeurs discrètes de l'angle. Cependant, si on fait l'hypothèse que le moteur tourne assez rapidement, on a donc des valeurs discrètes mais très rapprochées ce qui peut s'apparenter à un continuum de valeurs et donc un gain proportionnel.

Le système global simulé nous renseigne donc sur la façon dont réagit le moteur à une sollicitation sous la forme d'une consigne échelon en présence d'une perturbation. On a pu régler le correcteur PID de sorte qu'il donne un système globalement stable et suffisamment rapide. Le cahier des charges est donc rempli.

Modélisation sous Tinkercad :

Tous les aspects possibles du robot Veronica sont ici simulés grâce à Tinkercard. Une partie importante de cette modélisation était de vérifier le fonctionnement des capteurs de température et d'humidité, le code Arduino (équivalent à la carte Core) et les composants électriques qui seraient utilisés.

Nous avions choisi d'utiliser pour le robot réel des moteurs à courant continu avec codeur incrémental, des capteurs d'humidité et de température ainsi que un couple récepteur/émetteur

radiofréquence pour la communication avec l'interface. Le logiciel a permis d'étudier presque tous ces composants électriques, ne manquant que d'un répéteur radiofréquence. A la place, nous avons utilisé un wifi, ce qui est suffisant pour tester le fonctionnement du robot à l'intérieur du laboratoire. Il sera tout de même nécessaire de le remplacer plus tard par une communication RF pour être plus conforme au cahier des charges (communication longue distance à travers des objets).

Le logiciel utilisé était Tinkercad, appliqué par Autodesk qui permet de modéliser les composants électriques et principalement de simuler le fonctionnement de l'Arduino et du code.

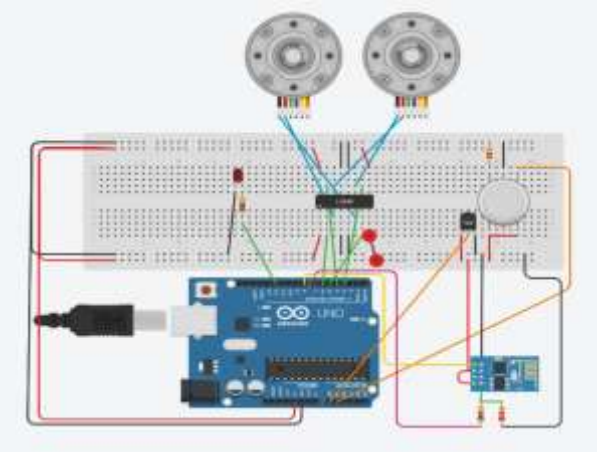


Figure 5 : Simulation finale du circuit électrique du robot Veronica.

La simulation (Figure 5) utilise deux moteurs à courant continu avec encodeur. Le moteur doit pouvoir tourner dans les deux sens, car nous l'utiliserons pour avancer en ligne droite et tourner autour de l'axe du robot.

Pour cela, un circuit électrique avec convertisseur de puissance est nécessaire pour inverser le sens du courant dans les pôles du moteur, nous utilisons donc un L239D. Le L293 est conçu pour fournir des courants d'attaque bidirectionnels jusqu'à 1 A avec des tensions variant de 4,5 V à 36 V. Nous pouvons utiliser deux sorties numériques de l'Arduino pour contrôler la direction des moteurs. Ainsi, le système fonctionne comme un interrupteur permettant de changer le sens du courant traversant les moteurs et donc leur sens de rotation.

Le capteur de température utilisé était un TMP 36. L'opération est très simple, nous alimentons le circuit en 5 V sur l'une des bornes, nous connectons l'autre à la terre (masse). Nous mesurons alors la tension sur la borne centrale de l'équipement. Cependant, il est nécessaire d'écrire la fonction d'étalonnage du capteur, par conséquent, nous utilisons la fonction Map qui nous permet de remapper les valeurs de tension, nous obtenons donc directement la température enregistrée dans la variable Température après lecture. On utilise un port analogique de l'Arduino.

Le capteur d'humidité n'est malheureusement pas présent dans le logiciel, nous utilisons donc un capteur aussi similaire que possible : un capteur de gaz. La simulation permet de créer une animation d'un gaz en concentration variable permettant alors de donner différentes valeurs de référence au capteur pour configurer le code le port analogique de l'Arduino de la mesure.

Malheureusement, il n'est pas possible de tester la transmission et la réception du signal à l'aide du module Arduino sur ce logiciel. Cependant, il a été possible d'écrire un code et de vérifier le bon branchement et la bonne utilisation des composants, car le logiciel avertit si certains équipements reçoivent un courant supérieur à la fiche technique.

III- Interface graphique

Afin de communiquer directement avec le modèle simulé sur Simulink nous avons fait le choix de créer notre interface via Matlab. Le but étant alors de commander la simulation et non le robot réel. L'interface Matlab est créée à partir de l'outil **App Designer** du logiciel.

C'est un environnement annexe de Matlab permettant de créer des applications à l'aide de composants déjà prédéfinis. Il suffit alors simplement de les déposer dans l'interface et de leur associer les fonctions, appelées callbacks, souhaitées. Le code de création de l'interface est automatiquement généré, nous avons seulement besoin d'associer les bons callbacks aux bons composants pour obtenir l'application voulue.

L'interface doit permettre de commander le système Simulink présenté ci-dessus. L'utilisateur doit donc pouvoir sélectionner le modèle puis entrer les paramètres de commande que nous avons jugés nécessaire à savoir : la distance de déplacement, le couple résistant et perturbatif et le temps d'exécution de la simulation. Afin de voir l'efficacité de la simulation réalisée, on affichera directement sur l'interface, les réponses données par le système. (Figure 6)

Figure 6 : Interface Machine-Humain

Le premier bloc « Choix du modèle » permet la sélection d'un modèle à simuler. Un callback est associé à l'action d'appuyer sur le bouton Select (Figure 7). Il permet l'ouverture d'un fichier du type modèle Simulink exclusivement (.slx) à l'aide de la fonction ***uigetfile*** puis le nom du fichier sans son attribut (grâce à la fonction ***erase***) est associé à un champ d'édition pour permettre à l'utilisateur de vérifier que le fichier sélectionné est bien le bon. Attention, une fois le fichier sélectionné, l'interface nous montre le modèle sous Simulink (ou le Workspace s'il n'a pas été ouvert), il faut donc penser à revenir sur l'interface pour communiquer avec le système.

```
% Button pushed function: SelectButton
function SelectButtonPushed(app, event)
    filename=uigetfile('*slx','selectionner le modèle');  %ouvrir le fichier
    filename=erase(filename,'.slx');  %supprimer l'attribut
    app.EditField.Value=filename;    %ecrire le nom du fichier
end
```

Figure 7 : code du callback générant la sélection d'un modèle Simulink

Le second bloc se rapporte à la définition des variables de consignes qui seront envoyées dans la simulation. Les variables Déplacement, Couple résistant et Couple perturbatif sont les consignes de la simulation, elles doivent alors être enregistrées dans le Workspace de Matlab. On utilise pour cela la fonction ***assignin*** qui permet d'enregistrer les variables définies sur App

Designer dans le Workspace, elles seront donc bien reconnues par le modèle Simulink lors de sa simulation.

La variable Temps d'exécution permet de gérer le temps de simulation du système, elle est directement utilisée pour simuler le système à partir de la fonction ***sim***, on n'a donc pas besoin de la définir dans le Workspace. Toutes ces actions sont associées au callback de l'appuie sur le bouton Simuler (Figure 8). Le système est ainsi commandé à la guise de l'utilisateur.

Enfin on trace les réponses simulées par le modèle du déplacement du robot et de la variation du couple résistant sur deux graphiques différents. Ces courbes sont tracées à la fin de la simulation d'où le fait de l'avoir mis dans le même callback que précédemment. Il est nécessaire d'ajouter des blocs ***To Workspace*** dans le modèle Simulink pour récupérer ces données.

```
assignin('base','Cr_perm',app.Couple.Value);       %assigner au workspace
assignin('base','Cr_pert',app.Couple_pert.Value);
assignin('base','d',app.Deplacement.Value);
simout=sim(SimulinkModel,'TimeOut',app.Temps.Value);    %simulation
plot(app.UIAxes,simout.OutDeplacement.Time,simout.OutDeplacement.Data)   %tracé
plot(app.UIAxes2,simout.OutCouple.Time,simout.OutCouple.Data)
```

Figure 8 : callback d'enregistrement des variables de simulation et tracés des réponses

Attention toutefois, pour faire fonctionner la simulation il est impératif que le modèle Simulink soit ouvert sur l'ordinateur (il faut aussi que tous les fichiers soient dans le même dossier mais cela est un impératif connu de l'utilisation de Matlab). Pour rappeler à l'utilisateur de l'ouvrir nous avons muni l'application d'une Startup fonction dont les actions associées sont effectuées à l'ouverture de l'application. Nous affichons alors grâce à cette fonction un message de rappel avec la fonction ***msgbox()*** pour que l'utilisateur ouvre le modèle Simulink avant de le simuler. Pour des raisons d'esthétisme, nous avons également choisi d'afficher l'application au centre de l'écran à son ouverture avec la fonction ***movegui***.(Figure 9)

```
% Code that executes after component creation
function startupFcn(app)
    movegui(app.UIFigure,'center');
    msgbox('Ouvrir le fichier Simulink avant utilisation','Attention','warn')
end
```

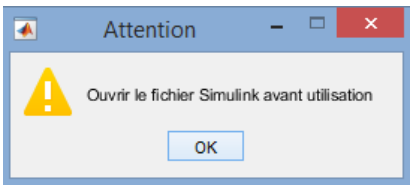
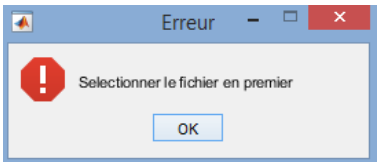


Figure 9 : StartUp fonction de l'interface et message d'erreur

Si l'utilisateur n'ouvre sélectionne pas de fichier et simule quand même en appuyant par erreur sur le bouton de simulation, un second message d'erreur apparait pour le lui signaler. Ce message est génère avec la fonction ***errorddlg()*** et permet d'éviter au code d'essayer d'exécuter une simulation dans laquelle il manque des variable.



L'interface permet ici de commander un système simulé sous Simulink. Elle ne peut pas commandée un robot via une carte Nucléo. Cette dernière option nécessite des modifications du code de l'interface pour communiquer avec des ports de contrôle et non un modèle Simulink. Mais dans le cas de l'interface du robot réel nous n'aurions pas codé sur Matlab mais plutôt sur un langage typé pour plus de rigueur.

IV- Préparation au retour en présentiel

Nous avons pensé dans cette partie à préparer les actions à mener sur le prototype concret de notre robot. Tout d'abord, nous avons pu voir avec notre simulation numérique du moteur à courant continu, comment gérer uniquement la translation en ligne droite. Or dans le cahier des charges initial, le robot devait également être capable d'effectuer des rotations. Pour décrire cette action, nous avons mis en place un algorithme sous forme d'organigramme (figure 10) montrant comment gérer une rotation à partir de deux moteurs :

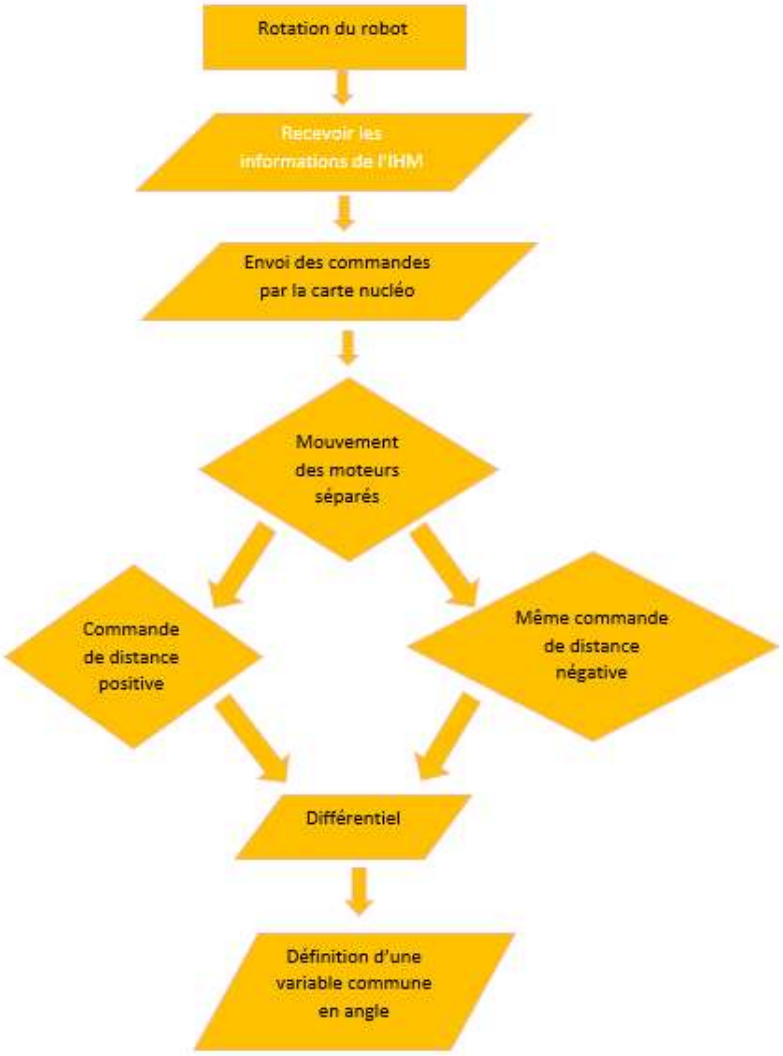


Figure 10 : Algorithme de rotation du robot

En effet, sur le prototype du robot, nous avions pensé à monter deux moteurs indépendants pour commander chacune des roues gauche et droite. Pour effectuer une rotation, il suffit alors de faire tourner les moteurs en sens inverse avec la même intensité. Pour évoluer notre modèle d'un cran supplémentaire, on décide de lui permettre d'effectuer une trajectoire d'un point A à un point B, comprenant des virages et des obstacles. Pour cela, les deux moteurs devront tourner dans le même sens avec une intensité différente. Ainsi, le robot sera capable d'adopter des trajectoires circulaires selon les commandes qui lui sont transmises par l'interface Humain-Machine.

D'un point de vue code de commande, les moteurs se contrôlent grâce à une commande PWM gérant leur vitesse de rotation grâce au courant fournit. Il suffit donc de changer le rapport

cyclique du PWM et le sens de passage du courant dans les moteurs pour faire tourner les roues dans des sens différents et à des vitesses différentes.

Pour répondre complètement au cahier des charges, le robot devait effectuer des mesures de température et d'humidité à intervalle régulier et les envoyer en retour à l'IHM. Pour répondre à cette fonctionnalité, nous aurions suivi les étapes suivantes (Figure 11), décrites ici pour la mesure d'humidité mais similaire pour les mesures de température :

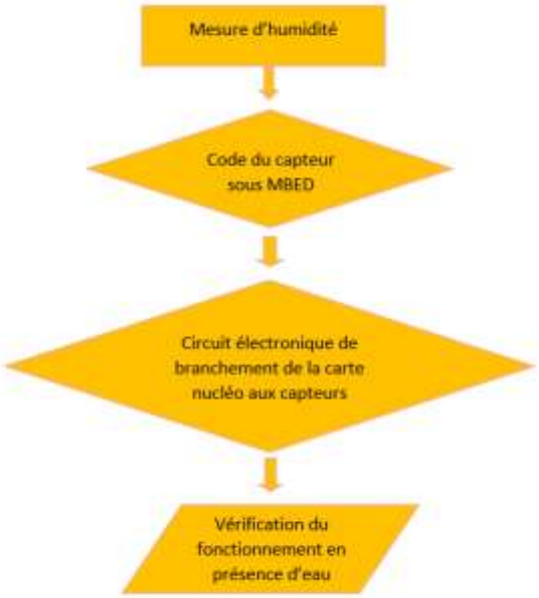


Figure 11 : Algorithme de mesures effectuées par les capteurs

On aurait finalisé le montage de tous les composants, c'est-à-dire de l'architecture du robot comprenant les deux moteurs asservis en position et les capteurs choisis au préalable (pour l'humidité le VMA303 de chez Velleman et pour la température, le ER400TRS-02 de chez LPRS) reliés par un circuit électronique à la carte nucléo ainsi que la batterie pour tenir l'autonomie. On aurait trouvé le moyen de faire fonctionner l'IHM couplée à une carte nucléo à distance. Il aurait fallu finalement tester le prototype, c'est-à-dire envoyer à partir de l'IHM les données de la trajectoire du robot comprenant une trajectoire courbée dans laquelle il aurait rencontré des flaques d'eau et des fluctuations de température.

V- Bilan des acquis

Simulation du moteur : (Aimé et Camille)

- On a appris à régler un PID de manière empirique en se plaçant au point de pompage et en ajustant selon la méthode de Ziegler et Nichols, le gain intégral. Méthode de réglage d'un PID de Ziegler et Nichols : https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Ziegler-Nichols
- On a appris à simuler un moteur à courant continu sous Simulink à l'aide des blocs de base en reprenant les équations électromécaniques. Méthode de l'académie de Toulouse : https://disciplines.ac-toulouse.fr/sii/sites/sii/files/se_former/formation_modelisation_multiphysique/2-tp_mcc_modelecausal.pdf

Interface Humain-Machine : (Enola)

- J'ai appris à prendre en main l'environnement Matlab App Designer. Pour cela j'ai essentiellement regardé les tutoriels vidéo proposés par l'ingénieur Benito Sébastian sur son site https://benitosebastian.com/tutorials/matlab-app-designer/. Ils sont classés par thème et traite d'un sujet à la fois ce qui permet de facilement isoler la vidéo nécessaire pour résoudre le problème auxquels on est confronté.
- J'ai appris à simuler et analyser les données d'un système Simulink directement dans le Worspace de Matlab. Pour cela je me suis inspirée des codes proposés par Caroline Kulsar dans son TP d'automatique (TP2) pour connaitre les fonctions qui permettent de simuler le système mais aussi les blocks Simulink permettant l'extraction des données vers le Worspace : https://ecampus.paris-saclay.fr/course/view.php?id=20549#section-4
- Pour tout autre problème rencontré, j'ai fait appel directement à la boite à outils de matlab avec centre d'aide https://ecampus.paris-saclay.fr/course/view.php?id=20549#section-4

Modélisation du prototype : (Luiz)

- J'ai appris à coder et simuler les fonctionnalités de la carte Arduino afin de planifier les circuits électriques ainsi que l'acquisition de données par le circuit. Il est possible de faire varier le temps entre chaque mesure, de visualiser la sortie en série de la carte Arduino et de tracer certaines courbes représentant les variables de notre système. Par exemple, sur la figure 12, on observe la valeur de la température et de l'humidité que l'équipement a acquis à chaque instant et on peut tracer les courbes de ces mêmes variables. Il est également possible d'effectuer des animations pour modifier et tester le fonctionnement et la réponse du circuit. (Figure 13)

Figure 12-13 : Moniteur série Arduino et visualisation des variables définies – animation

VI- Retour d'expérience

Dans cette partie, nous allons vous faire part des sensations de chacun lors de ce projet.

Camille :
« Cette expérience à distance a été enrichissante car elle nous a permis de découvrir des logiciels de modélisation et de simulation, jusqu'ici trop peu connu pour ma part. J'ai l'impression qu'elle a été également bénéfique dans l'organisation des tâches au sein du groupe. En effet, comme c'était plus compliqué de communiquer et d'échanger sur nos modèles et nos résultats, on a développé une meilleure écoute et une meilleure entraide. Cependant, je regrette un peu de ne pas avoir pu insérer l'aspect de mesures d'humidité et de température normalement effectuées par les capteurs. J'ai en quelque sorte une sensation d'inachevé, notre prototype final n'a pas de construction concrète pour moi. C'est dommage mais je suis consciente que nous n'avions pas d'autres choix. »

Aimé :
« Quel bonheur de voir une simulation de machine à courant continu fonctionner correctement après des heures de travail intensif pour comprendre pourquoi elle ne fonctionnait pas jusqu'ici. Plus sérieusement, le changement de cahier des charges que nous nous sommes imposés en raison de la situation actuelle s'est avéré très réussi. J'ai appris qu'on pouvait simuler des systèmes complexes avec de simples blocs reliés entre eux. On aurait pu aller encore plus loin et simuler peut-être la rotation du robot sur lui-même etc… L'interaction au sein du groupe était très réussie et ne manquait pas de sérieux et de compétences. »

Enola :
« L'annonce du maintien du projet à distance et sa réalisation sous forme de simulation n'a pas été une si mauvaise nouvelle pour moi car j'aime plutôt coder. J'ai donc bien abordé ce changement même si j'ai été un peu déçue de ne pas pouvoir me confronter aux soucis de branchement et de fonctionnement réel du robot. Je me suis concentrée sur la partie interface qui m'a plutôt amusée, j'ai été surprise par le nombre d'application qu'on pouvait réaliser lorsqu'on sait maitriser App Designer. J'ai travaillé avec Lilian au début du projet qui s'y connait plutôt bien en informatique, il a donc pu m'apprendre quelques astuces à ce sujet comme l'utilisation de Gitlab pour sauvegarder les versions des codes. Malgré le fait de ne pas être présent tous ensemble nous avons réussi à bien communiquer avec les outils à disposition. Nous avons été très efficaces pour nous adapter mais j'ai trouvé dommage de ne pas pouvoir voir les autres travailler sur leur partie du projet. »

Luiz :
« J'ai fait des recherches sur un logiciel comme celui-ci car j'étais très intéressé par la partie électrique et vu le fonctionnement du robot dans la version finale, malheureusement nous ne pourrons pas construire le projet final. Le programme permet vraiment de réaliser une planification adéquate des circuits de base, je crois que c'est un excellent moyen de commencer à planifier un projet, avant même de partir acheter les composants.
Le travail en ligne a été une nouvelle expérience qui sera très importante, car nous avons de plus en plus d'exemples de bureau à domicile et la nécessité de partager le travail, d'être clair lors de la communication et la recherche de moyens alternatifs pour résoudre les problèmes auxquels nous sommes confrontés servira de guide pour faire face à l'environnement des affaires. »