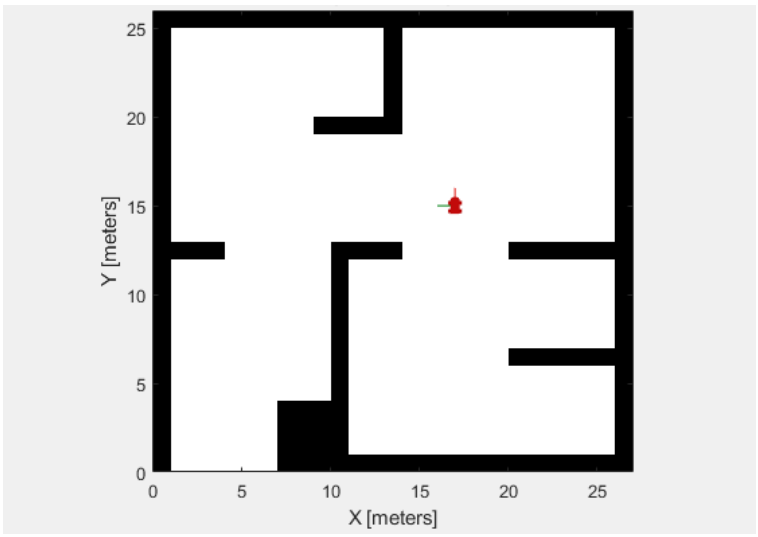
Automatisation Intelligente et Systèmes Cyber-Physiques

Rapport d'Etude de Cas



Membre du groupe :

BADOLO Christian Thomas

BILA Ahmed Kader

Encadré par :

M. Ahmed RAHMANI

Table des matières

Liste des figures

Introduction

Les systèmes cyber-physiques (CPS) représentent une convergence innovante entre les mondes virtuel et physique, où des systèmes informatiques interagissent étroitement avec des processus physiques en temps réel. Ce domaine émergent a un impact significatif sur divers secteurs, allant de l'industrie manufacturière à la santé et à la mobilité.

Dans le cadre du cours sur l'Automatisation Intelligente et les Systèmes Cyber-Physiques, notre étude de cas se concentre sur la planification de trajectoire pour un robot à entraînement différentiel, réalisée à l'aide de l'environnement de modélisation Simulink. Cette étude a débuté par une compréhension approfondie du modèle de base du robot, incluant ses caractéristiques cinématiques et dynamiques. Nous avons ensuite procédé à la proposition d'équations pour modéliser le comportement physique du robot. Pour finir, nous avons exploré plusieurs améliorations potentielles pour le modèle et l'algorithme de planification de trajectoire.

I. Compréhension du model de base

1. Planning

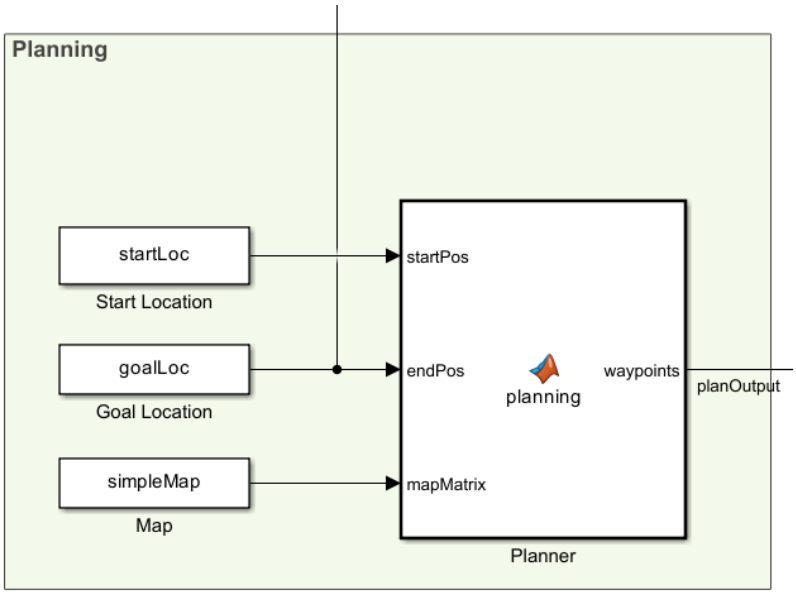


Figure 1 Planning section of the base model

L'objectif est de déterminer le trajet que le robot doit emprunter pour se rendre d'un point de départ à un point d'arrivée. En se basant sur la carte $simpleMap$, représentée par une matrice binaire, l'algorithme de planification renvoie un chemin allant du point de départ $startLoc$ au point d'arrivée $goalLoc$.

2. Contrôle

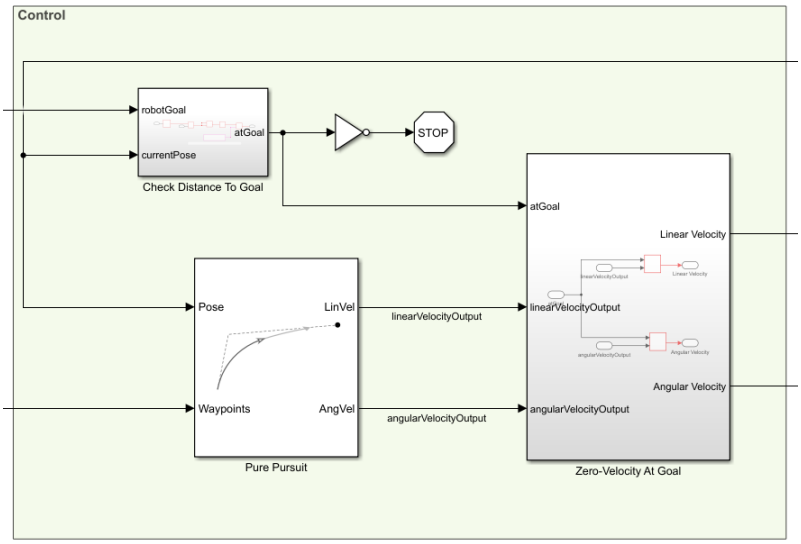


Figure 2 Control section of base model

Dans cette partie se fait l'ajustement la vitesse du robot. Plus précisément, en utilisant la liste des chemins reçus, le bloc $Pure\ Pursuit$ détermine une vitesse linéaire et une vitesse angulaire pour le robot.

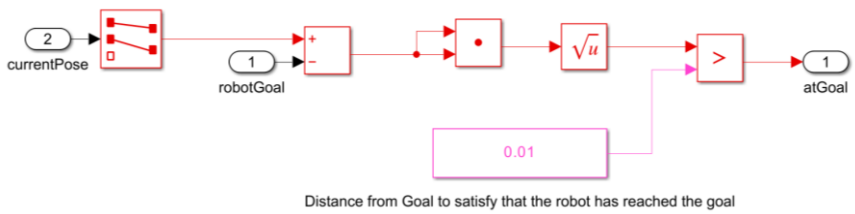


Figure 3 Robot final destination verification

Dans ce processus, la position actuelle du robot est comparée à sa position finale pour déterminer s'il est arrivé ou non. Lorsque la distance entre les deux positions est inférieure à un seuil prédéfini, nous considérons que le robot a atteint sa destination ; sinon, qu'il n'y est pas encore.

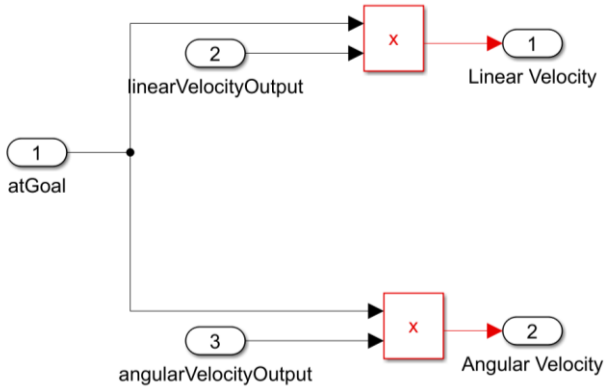


Figure 4 Zero Velocity at Final Position

Afin de garantir l'arrêt du robot une fois arrivé au point final, une condition qui annule sa vitesse à cet emplacement est introduite.

3. Plant model

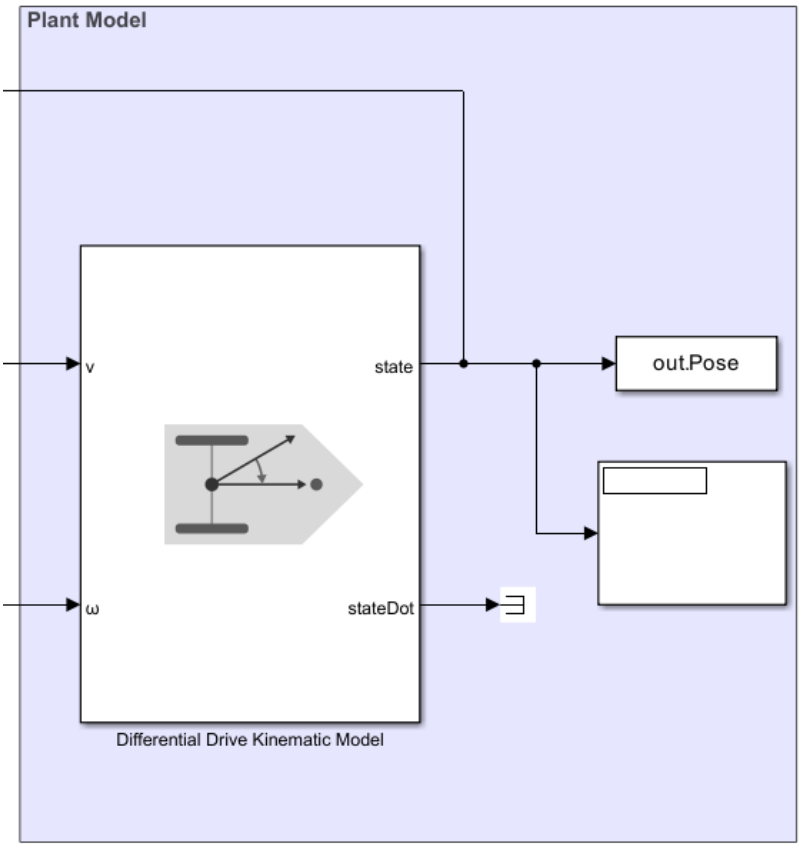


Figure 5 Plant model section of the base model

Il s'agit du model cinématique du robot. Dans ce cas le Differential Drive Kinematic Model a été utilisé. Il prend la vitesse linéaire et angulaire en entrée et donne l'état du robot en sortie qui est constitué de $x, y \; et \; \theta$.

Une variable $Pose$ contient le résultat de la simulation. L'on a entre autres les coordonnées qui constituent le chemin que le robot devra suivre pour aller du point de départ au point d'arrivé. Ces coordonnées serviront à tracer le chemin du robot sur la map.

3.1. Modélisation du différential drive kinemetic model

3.1.1. Modèle 1

3.1.1.1. Modélisation : équations différentielles

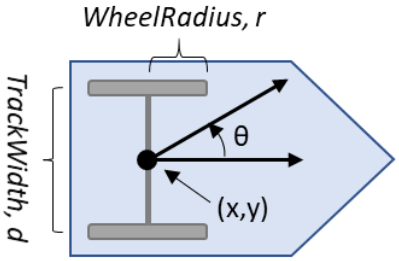


Figure 6 Différential drive kinemetic model

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\cos(\theta) & 0\\ \sin(\theta) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}$$

Variables

- $\boldsymbol{x}$ *: Position globale du véhicule sur l'axe x, en mètres*
- $\boldsymbol{y}$ *: Position globale du véhicule sur l'axe y, en mètres*
- $\boldsymbol{\theta}$ *: Orientation globale du véhicule, en radians*
- $\boldsymbol{v}$ *: Vitesse du véhicule en mètres/seconde*
- $\boldsymbol{\omega}$ *: Vitesse angulaire de l'orientation du véhicule en radians/seconde*

La résolution de l'équation différentielle donne :

$$\boldsymbol{x = v\int \cos(\theta)} \qquad \boldsymbol{y = v\int \sin(\theta)} \qquad \boldsymbol{\omega = \int \dot{\theta}}$$

3.1.1.2. Modélisation : schéma block

Nous avons modélisé le schéma block suivant en utilisant les équations ci-dessous.

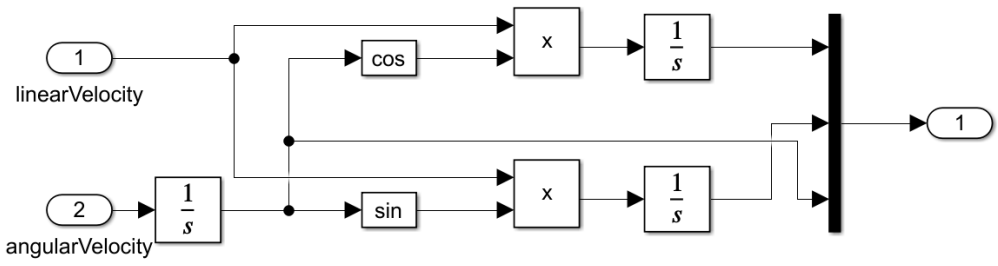
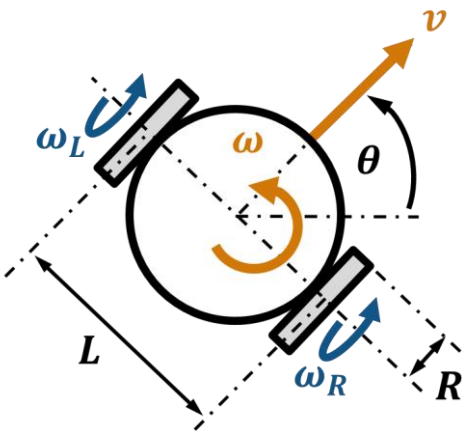


Figure 7 differential drive kinematic model schema bloc

3.1.2. Modèle 2

3.1.2.1. Modélisation : équations différentielles



Entrée *: Vitesses des roues gauche et droite, en rad/s.*

Outputs :

- *Vitesse linéaire v, en m/s*
- *Vitesse angulaire ω, en rad/s*

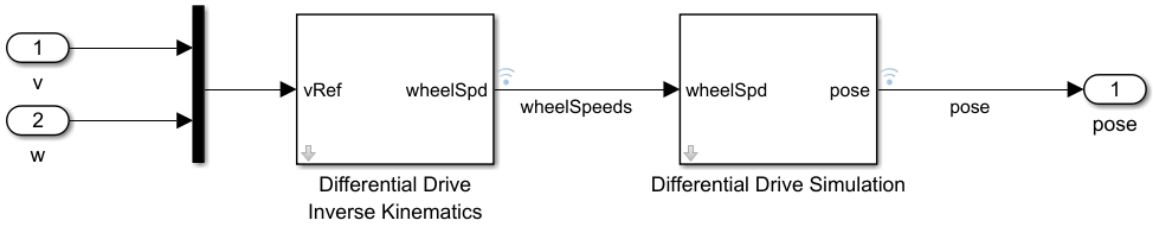
Forward Kinematics :

$$v = \frac{r}{2}(\omega_r + \omega_d), \qquad \omega = \frac{r}{d}(\omega_r - \omega_d)$$

Inverse Kinematics :

$$\omega_L = \frac{1}{r}\left(v - \frac{\omega L}{2}\right), \quad \omega_R = \frac{1}{r}\left(v + \frac{\omega L}{2}\right)$$

3.1.2.2. Modélisation : schéma block



3.1.3. Validation

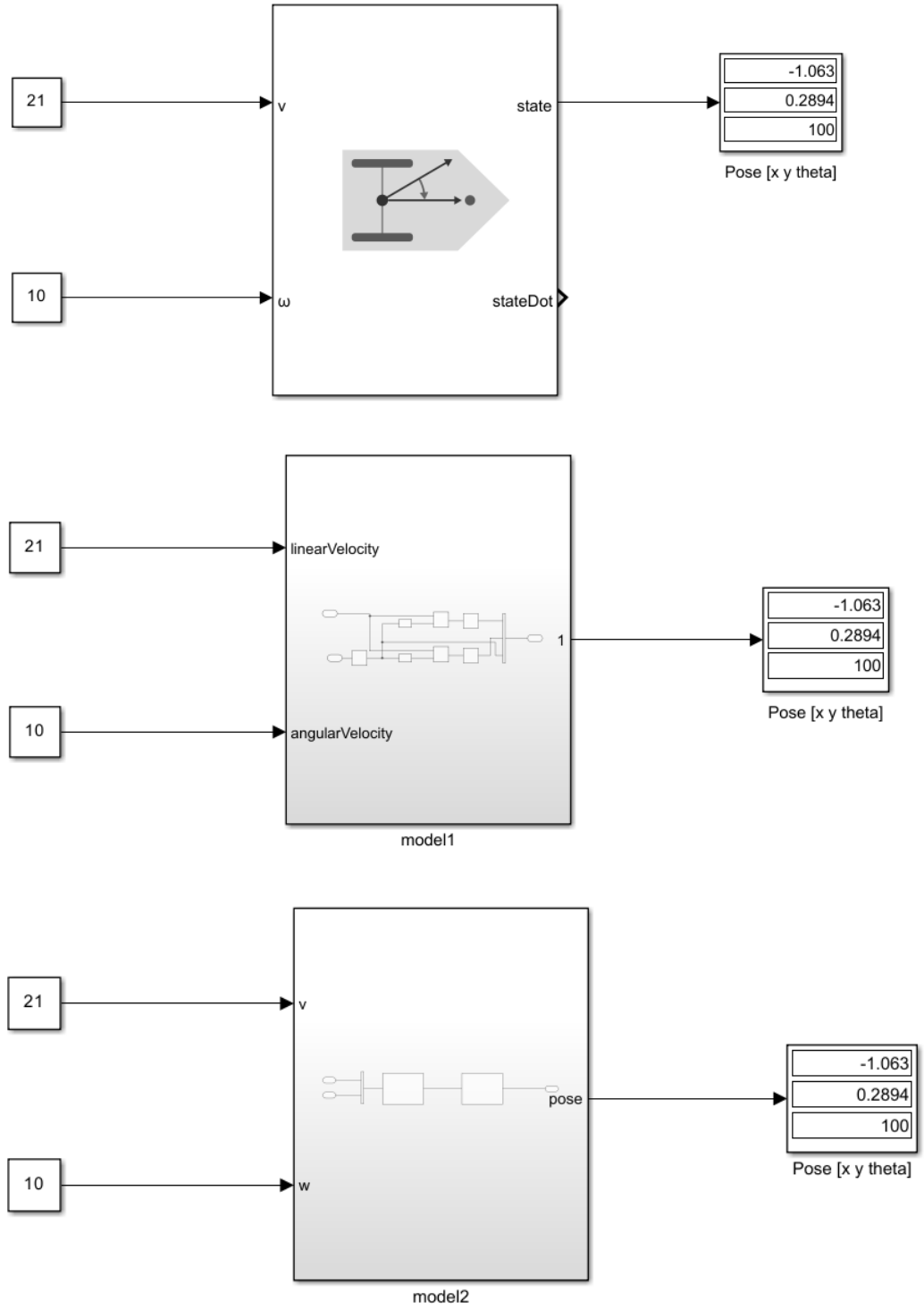


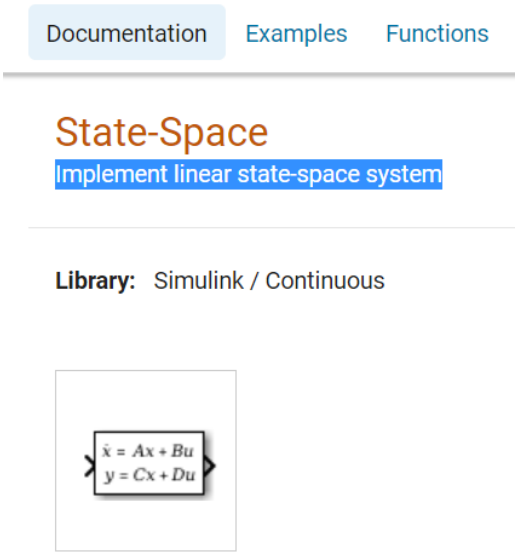
Figure 8 Validation de nos modèles

Nous avons entré les mêmes vitesses linéaire et angulaire dans les trois modèles, et nous avons obtenu les mêmes résultats, c'est-à-dire les mêmes valeurs pour x, y et θ.

3.2. Equations d'états / Contrôle linéaire (lqr contrôler)

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Le modèle représentant la cinématique différentielle du robot est malheureusement non linéaire. La modélisation en temps continu en utilisant les blocs liés à l'espace d'état dans Simulink (et PAS en utilisant des blocs intégrateurs) s'est avérée impossible.



Les matrices d'état A, B, C, D contenaient des fonctions des variables d'état en l'occurrence $cos(\theta)$, que nous ne pouvons pas définir dans le bloc d'espace d'état de Simulink. Le bloc "State-Space" ne prend pas en charge les fonctions matricielles non linéaires comme le dit l'aide "Implémenter un système linéaire en espace d'état". Nous devons donc linéariser le système.

Consider the differential drive car dynamics

$$\dot{x} = V\cos\theta$$
$$\dot{y} = V\sin\theta$$
$$\dot{\theta} = \omega$$

State is $[x, y, \theta]$ control is $[V, \omega]$

$$\delta\dot{x} = A\,\delta x + B\,\delta u$$

$$A = \frac{\partial f}{\partial x} \qquad B = \frac{\delta f}{\partial u} - \begin{bmatrix} v \\ \omega \end{bmatrix}$$

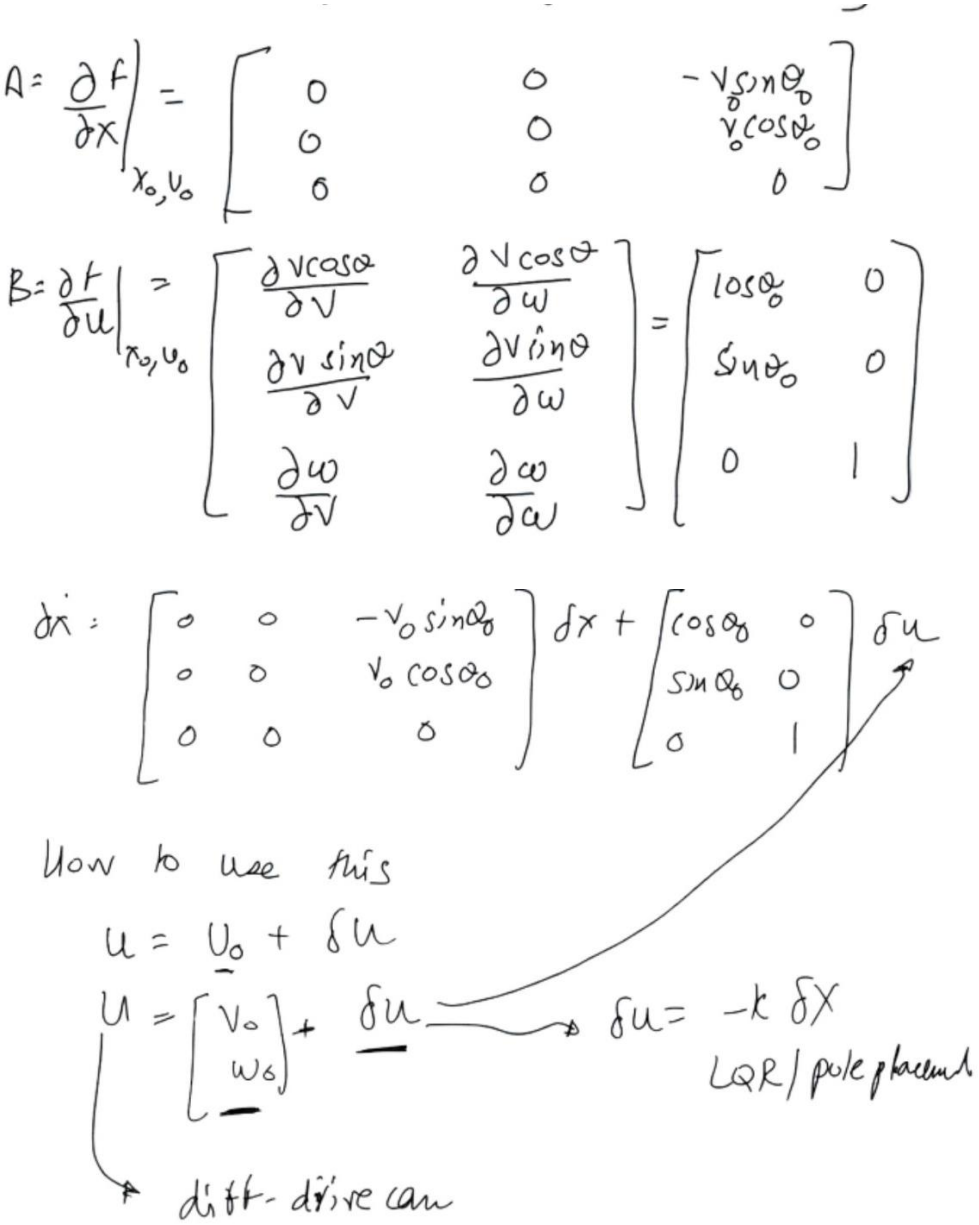
$$\dot{X} = F(X)$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}}_{f}$$

$$f = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}$$

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial v\cos\theta}{\partial x} & \frac{\partial v\cos\theta}{\partial y} & \frac{\partial v\cos\theta}{\partial \theta} \\ \frac{\partial v\sin\theta}{\partial x} & \frac{\partial v\sin\theta}{\partial y} & \frac{\partial v\sin\theta}{\partial \theta} \\ \frac{\partial \omega}{\partial x} & \frac{\partial \omega}{\partial y} & \frac{\partial \omega}{\partial \theta} \end{bmatrix}$$

$$\downarrow \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$



Comme mentionné, les matrices d'état A, B, C, et D. Pour implémenter un système dans Simulink, qui est basé sur des modèles linéaires en espace d'état, nous devons linéariser le système. Le contrôleur LQR est conçu pour des systèmes linéaires, ce qui en fait une approche appropriée pour notre cas.

Nous avons implémenté un code python pour analyser et contrôler le système linéaire en espace d'état obtenu. Il calcule les valeurs propres du système non contrôlé, évalue sa contrôlabilité, réalise le placement des pôles pour la stabilisation, et enfin, calcule les gains du contrôleur LQR pour obtenir une commande optimale en minimisant une fonction coût quadratique. Ces étapes fournissent un cadre complet pour l'analyse et le contrôle efficace du système

II. Modifications apportées

1. Changement de la map avec ajout d'une zone de chargement

Le modèle de base possédait d'un point de départ et d'un point d'arrivé. Nous nous sommes proposés d'y ajouter un point intermédiaire qui sera comme un point de chargement.

1.1. Démarche

Pour intégrer une zone de chargement comme point intermédiaire dans le modèle de base, nous avons ajouté un troisième point sur la carte, symbolisant la zone de chargement. En utilisant un Stateflow Chart dans Simulink, nous avons défini le comportement du robot lors de son déplacement, en lui faisant passer par la zone de chargement avant d'atteindre le point final. Le Stateflow Chart permet de gérer les actions du robot à chaque étape du trajet, comme l'arrêt pour le chargement ou le déchargement, assurant ainsi une gestion efficace du parcours du robot incluant la zone de chargement.

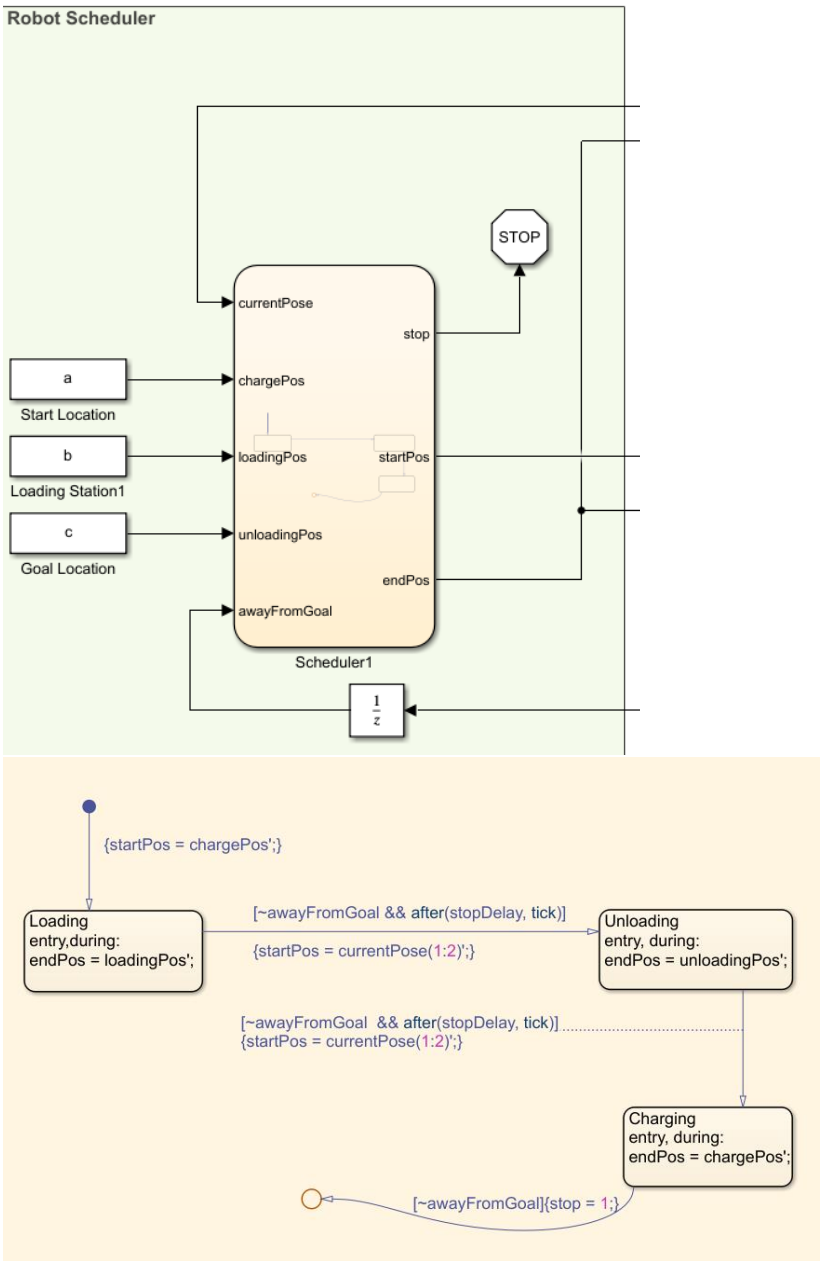


Figure 9 State flow chart pour intégrer un point de chargement

1.2. Résultat

Voici la présentation de la modification apportée.

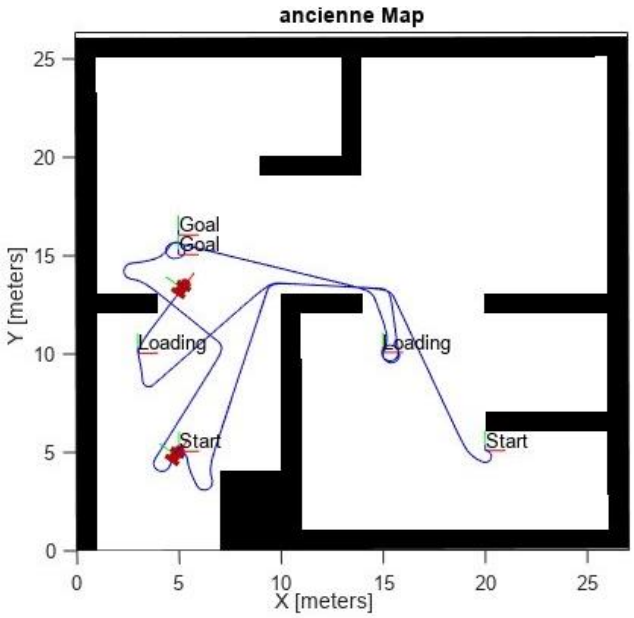


Figure 10 Ancienne map

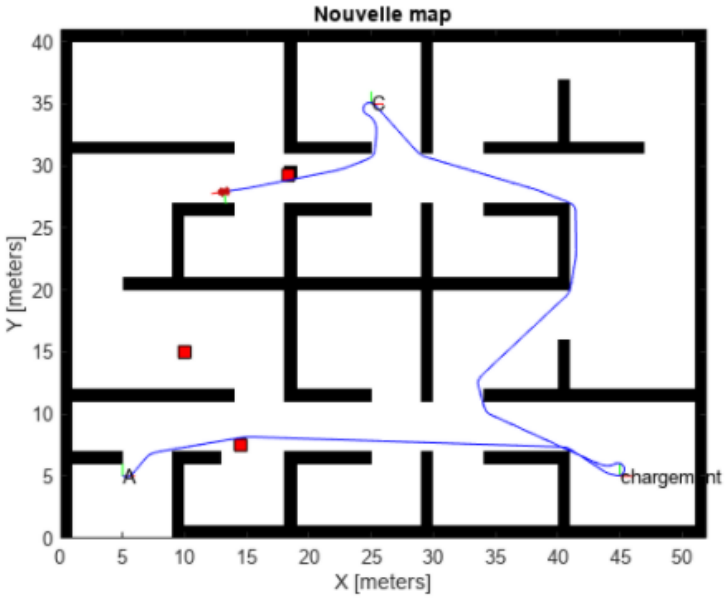


Figure 11 Nouvelle map

2. Ajout de deux autres robots

2.1. Démarche

Comme précédemment mentionné, la variable $Pose$ fournit une liste de coordonnées que le robot doit suivre. Ainsi, en effectuant plusieurs simulations avec différentes variables $Pose$, nous obtenons plusieurs chemins distincts et donc plusieurs robots associés à ces chemins.

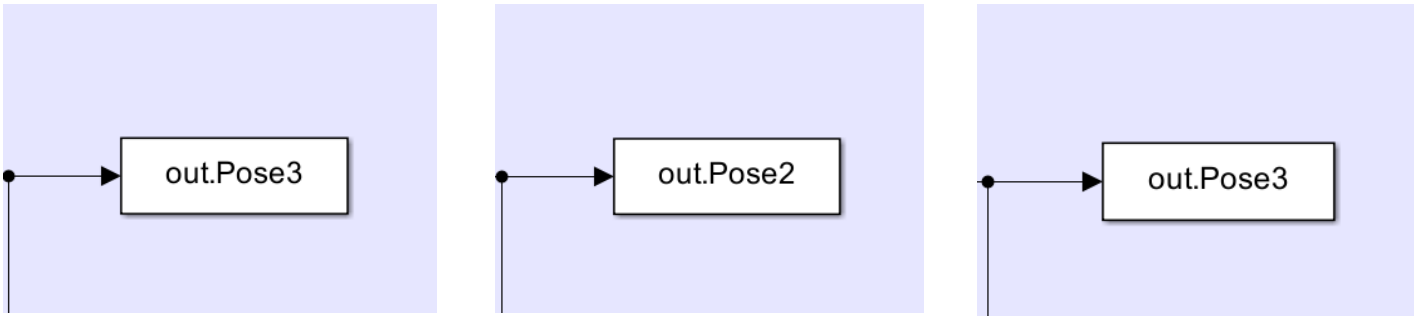


Figure 12 Simulations out.Pose

2.2. Résultat

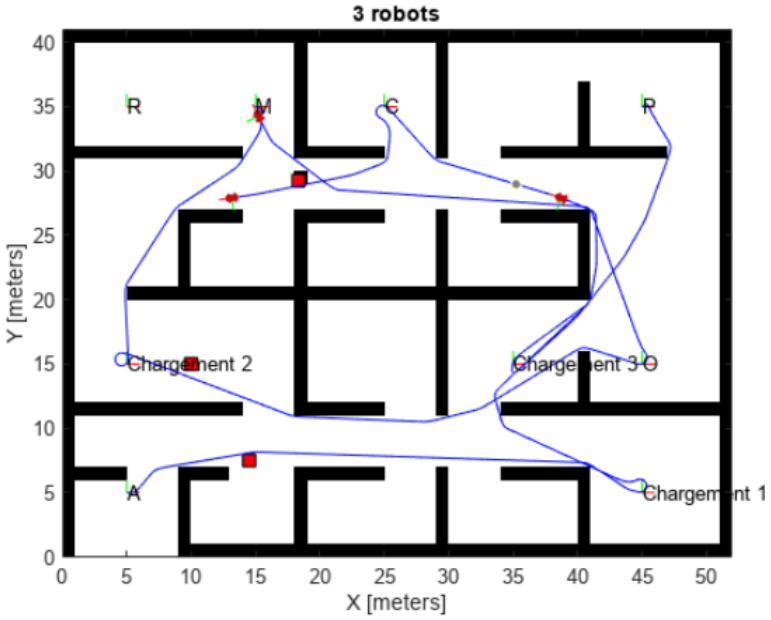


Figure 13 Ajout de 2 autres robots

3. Gestion de la collision entre les robots

Nous avons exploré la problématique de l'évitement d'obstacles mobiles en intégrant deux autres robots capables de communiquer entre eux afin d'éviter les collisions. Pour se faire nous nous sommes aidé de la librairie **Mobile Robotic Simulation**.

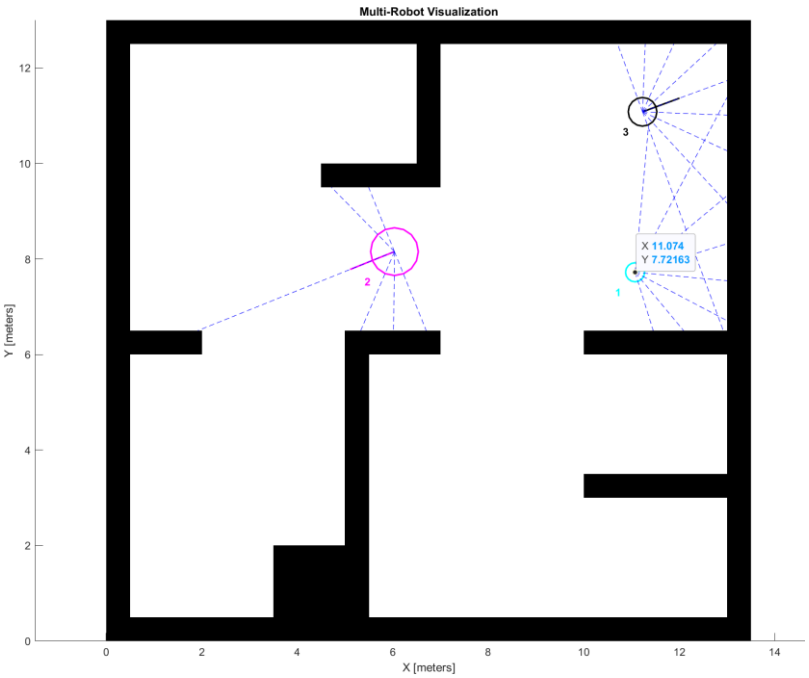


Figure 14 Evitement d'obstacle

4. Ajout d’un capteur lidar pour éviter les obstacles

Nous avons intégré un capteur LiDAR à notre robot afin de lui permettre d'éviter les obstacles en utilisant la bibliothèque **Mobile Robotic Simulation.**

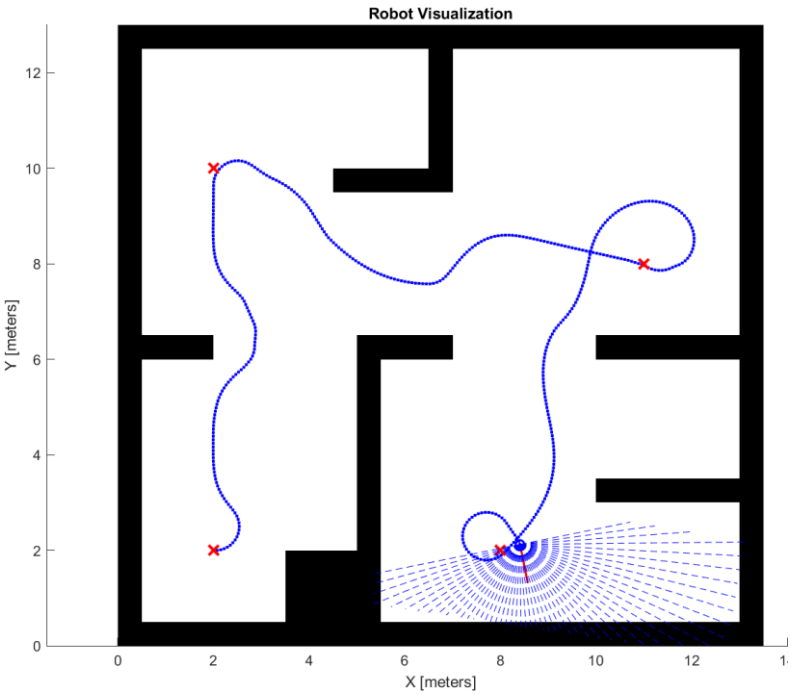


Figure 15 Intégration du capteur LiDAR

Conclusion

L'étude de cas sur la planification de trajectoire pour un robot à entraînement différentiel dans le cadre des systèmes cyber-physiques a été une expérience enrichissante, mais également confrontée à divers défis actuels et de complexité technique. La gestion des interactions entre les contraintes physiques, les obstacles environnementaux et les exigences de mouvement a exigé une approche rigoureuse et une compréhension approfondie des principes de la robotique autonome.

Malgré ces obstacles, cette étude a permis d'explorer les possibilités et les limites des CPS en matière de robotique autonome. Les leçons apprises sur la modélisation, la simulation et l'optimisation des trajectoires pour les robots différentiels sont précieuses pour la communauté de recherche et les professionnels travaillant dans le domaine.

Pour de futurs travaux, il serait essentiel d'envisager une implémentation pratique avec du matériel physique, ce qui permettrait de valider et d'améliorer nos approches de planification de trajectoire en tenant compte des défis réels rencontrés sur le terrain. Cette approche pratique renforcerait la pertinence et l'applicabilité de nos résultats dans des contextes industriels et réels.

Bibliographie

1. MathWorks. "Path Planning - MATLAB & Simulink". [En ligne]. Disponible : https://www.mathworks.com/discovery/path-planning.html#:~:text=Path%20planning%20lets%20an%20autonomous,and%20goal%20states%20as%20input.
2. MathWorks. "Mobile Robot Kinematics Equations - MATLAB & Simulink". [En ligne]. Disponible : [https://www.mathworks.com/help/robotics/ug/mobile-robot-kinematics-equations.html] (https://www.mathworks.com/help/robotics/ug/mobile-robot-kinematics-equations.html).
3. MathWorks. "Lidar - MATLAB & Simulink". [En ligne]. Disponible : [https://www.mathworks.com/help/lidar/index.html?s_tid=srchtitle_site_search_1_lidar] (https://www.mathworks.com/help/lidar/index.html?s_tid=srchtitle_site_search_1_lidar).
4. MathWorks. "occupancymap - MATLAB & Simulink". [En ligne]. Disponible : [https://www.mathworks.com/help/nav/ref/occupancymap.html?s_tid=srchtitle_site_search_4_map] (https://www.mathworks.com/help/nav/ref/occupancymap.html?s_tid=srchtitle_site_search_4_map).
5. "How to Draw ODEs in Simulink," MathWorks Blogs, May 23, 2008. [En ligne]. Disponible : https://blogs.mathworks.com/simulink/2008/05/23/how-to-draw-odes-in-simulink/.
6. "Simulink Tutorial for Beginners | Simulink Introduction | Simulink Training | Edureka," Edureka, YouTube, [Vidéo en ligne], 7 janvier 2021. Disponible : https://www.youtube.com/watch?v=owv_2YdO8rA.