Département Génie Electrique et informatique

Travaux pratiques d'automatique

4ème année AE

Semestre 2

v2.0

Table des matières

MANIPULATION 1

Recherche de chemins dans un graphe : Robot Griffith

Le robot Griffith

1.1 But de la manipulation

L'objectif de cette manipulation est de commander un robot sur 2 roues motrices et une roue libre pour l'équilibre, dont la commande est semblable à celle d'un rover (Figure 1), de manière :

- à ce qu'il suive de manière efficace une ligne noire au sol pour guider sa trajectoire (commande de bas niveau de type commande continue : Niveau 0)
- à ce qu'il choisisse correctement un chemin à suivre en fonction d'un critère et de contraintes données (commande de haut niveau type Intelligence Artificielle : Niveau 2)
- à ce qu'il suive correctement le chemin voulu (commande type contrôleur d'exécution : Niveau 1)

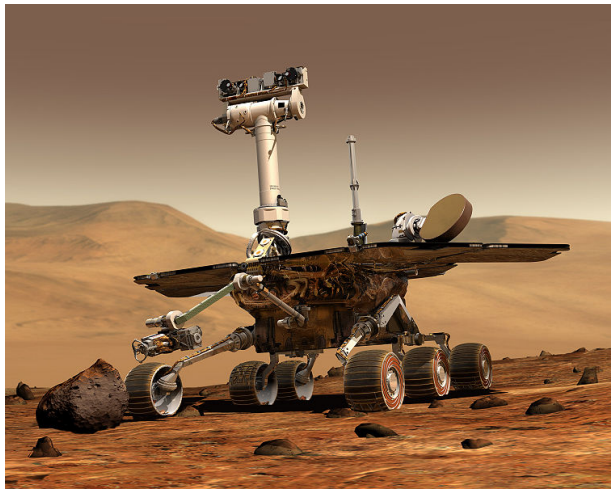


FIGURE 1 – Le rover Curiosity

Ces différents niveau de commande correspondent à différents champs du domaine de l'Automatique en générale, et sont illustrés sur la Figure 2.

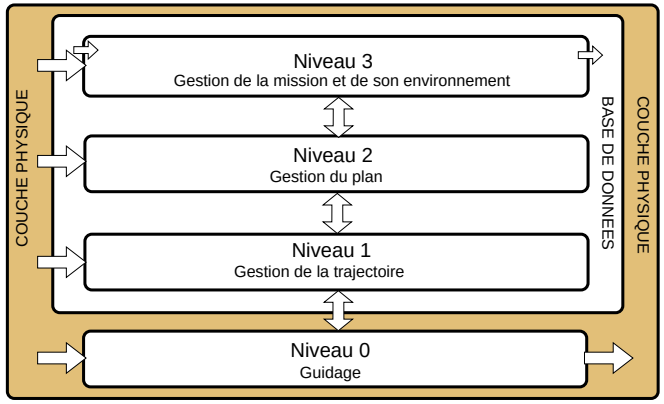


FIGURE 2 – Les différents niveaux de commande dans une architecture d'autonomie

1.2 Présentation de la maquette

On travaille avec un robot sur 2 roues et une roue libre pour l'équilibre représenté sur la Figure 3. Le cœur du robot Griffith est constitué d'une Raspberry Pi Pico : elle intègre un microcontrôleur RP2040 (processeur ARM Cortex M0+ à double cœur) qui est programmable en MicroPython à travers son port USB/ série (situé à l'arrière). MicroPython est une version allégée de Python conçue pour des systèmes embarqués aux ressources limitées. Le robot, une fois déconnecté du PC, est autonome. Il peut communiquer en Bluetooth vers un PC équipé d'un dongle USB/Bluetooth. Il est protégé par une carrosserie blanche, recouverte d'un dôme noir surplombé d'un bouton poussoir rouge (marche/arrêt), d'une LED

verte (état on) et bleue (status d'activité). Sur le devant se trouve la prise de rechargement 5V et une pièce saillante, en forme de bec noir, qui déporte et protège un capteur optique de ligne.

FIGURE 3 – le robot Griffith

La vitesse du robot est fournie par des encodeurs optiques placés dans les roues. Seul un de ces encodeurs est utilisé pour calculer la vitesse. La détection de ligne est basée sur une barrette rouge de 8 capteurs optiques digitaux. Ces capteurs renvoient un bit à 1 lorsque le couple est devant la couleur noire. Une fonction a été développée pour renvoyer une somme pondérée suivant la position du capteur par rapport à une ligne noire.

Une prise USB B permet de connecter le robot au PC (vu comme un port COM série) : on accède ainsi à l'interface de programmation du robot en MicroPython, à travers l'IDE Thonny installé sur le PC. La Raspberry Pi Pico possède un interpréteur de commande Python interactif, appelé REPL (Read-Eval-Print Loop).

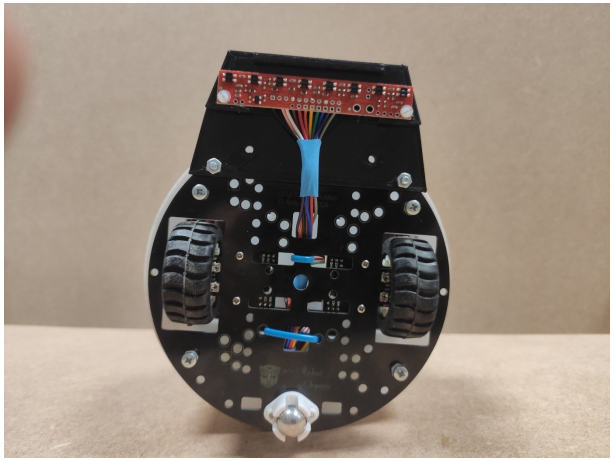


FIGURE 4 – Vue de dessous

En dehors d'un câble physique USB B, le robot communique aussi grâce à un module ajouté en interne, en Bluetooth, vers un dongle USB/Bluetooth branché sur un PC. Le robot peut ainsi recevoir des ordres simples pour se déplacer (avancer, tourner etc.) et envoyer des informations (acquittement de commande, niveau de batterie etc.). On utilise, dans notre cas, ce canal pour communiquer avec une autre application utilisant Python et l'IDE, Pycharm : la communication se fait simplement entre deux ports série (le protocole Bluetooth est transparent). Le dongle, rouge et jaune, est à brancher sur un port USB du PC qui l'identifiera comme un port COM.

Le robot se déplace sur une carte constituée de lignes noires et de points identifiés.

1.3 Présentation du dossier TPGriffith

Télécharger dans votre répertoire le dossier TP Griffith disponible sous moodle. Le dossier Griffith correspond aux fichiers de programmation du robot. Le code sera embarqué sur la carte du robot. Le dossier PC correspond aux fichiers de programmation du PC qui va communiquer avec le robot.

1.4 Présentation du dossier Griffith

Le dossier Griffith contient le fichier `robotRoverv3.py`. Ce fichier permet de configurer les ports d'entrées/sorties du robot et d'effectuer la commande de bas niveau en envoyant des commandes PWM vers les moteurs. Ce code est embarqué sur la carte Raspberry Pi Pico du robot.

1.5 Présentation des fichiers PC

Le dossier PC contient deux dossiers :

- `interface_graphique_Dijkstra` : permet le lancement du Dijkstra sur le PC via une interface graphique.
  - `calcul_Dijkstra.py` : calcule le plus court chemin par l'algorithme de Dijkstra
  - `carte_parcours_robot.py` : donne la carte (coordonnées des points de la carte, indication des arcs existants, vitesse associée aux arcs). C'est dans ce fichier qu'on pourra modifier les vitesses associées aux arcs
  - `data_graphe.py` : calcul les coûts associés à chaque arc du graphe, défini le graphe
  - `envoi_robot.py` : exécution d'une commande élémentaire vers le robot
  - `portcom.py` : gestion des ports com
  - `commandes.py` : lance un dialogue avec l'utilisateur pour envoyer des commandes de bas niveau vers le robot
  - `rover.py` : lance l'interface graphique principale pour calculer le Dijkstra et l'appliquer sur le robot

1.6 Mise en place de la partie Guidage

Dans un premier temps, l'objectif est de mettre en place la commande de bas niveau sur le robot, c'est-à-dire le guidage autour d'une ligne noire, suivant le schéma de la Figure 5.

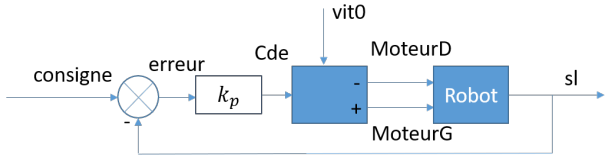


FIGURE 5 – Schéma de régulation

- Expliquer le schéma de régulation. Pourquoi les commandes envoyées sur les deux moteurs sont-elles différentes ?
- Compléter le fichier `robotRoverv3.py` de façon à mettre en place la boucle de régulation proportionnelle autour d'une consigne en vitesse notée `vit0` et d'une position centrale sur la ligne noire (consigne = 4.5), sachant que l'erreur et la commande sont de type (int). La valeur de kp est fixée et vaut 15 par défaut. La partie à compléter est indiquée dans le code.
- Tester votre régulation en réel avec le fichier `commandes.py` avec une valeur de vitesse constante réglée à 250.
- Faire varier la vitesse entre 0 et 500 de manière à déterminer la plage de valeurs acceptable pour la vitesse.
- Faire varier Kp entre 5 et 40 par exemple de manière à mettre en évidence les caractéristiques d'un régulateur proportionnel.

Une fois la commande de bas niveau (Niveau 0 : guidage) validée, on va mettre en place la commande de niveau 1 : gestion de la trajectoire.

1.7 Mise en place de la partie Gestion de la trajectoire

Cette partie consiste à vérifier que la gestion de la trajectoire est bien effectuée sur le robot. Pour cela, on se contente de valider le bon séquencement d'une mission écrite "à la main".

- En modifiant le fichier `commandes.py`, créer à la main une mission allant du point A au point Q sur la carte, avec une vitesse de 250 en utilisant des commandes du type : `exec(vitesse, longueur, angle)` ; avec les angles en degrés, la longueur en mm et la vitesse en degrés par seconde.
- Valider cette mission sur le robot.

1.8 Mise en place de la partie Gestion du plan

Cette partie consiste à élaborer une stratégie pour l'élaboration d'un plan de mission. Pour cela, on est amené à définir un graphe dont les sommets sont les points visitables et les arcs les chemins entre ces points.

- Formaliser le problème d'optimisation que vous allez tenter de résoudre pour élaborer un chemin pour le robot en établissant un critère d'optimisation, en vous inspirant de l'annexe intitulée "Théorie des graphes - Algorithme de Dijkstra".
- Compléter la fonction `cout` dans le fichier `data_graphe.py` de manière à définir le poids des arcs dans le graphe suivant le critère choisi.
- Proposer sur le papier une méthode de recherche de chemin intuitive, par exemple un algorithme glouton qui permettrait de trouver un chemin entre un point de départ et un nœud fin.
- Ouvrir le fichier `calcul_Dijkstra.py`. Etudier le code de manière à retrouver les étapes du pseudo-algorithme donné en annexe.
- Compléter le fichier `calcul_Dijkstra.py` :
  - la fonction `noeudpluspetitlambda()` doit retourner le nœud avec la plus petite valeur de lambda et lambda.
  - la fonction `dijkstra` doit être complétée
- Tester cet algorithme en utilisant un critère de distance (plus court chemin).
- On s'intéresse maintenant à faire un chemin le plus rapide en termes de temps de trajet. Modifier la fonction `cout` dans le fichier `data_graphe.py` de manière à adapter l'algorithme.
- Mettre en évidence un changement suite à ce changement de critère en changeant éventuellement les vitesses sur la carte.

1.9 Extensions possibles

Cette partie vise à étendre l'algorithme pour prendre en compte d'autres critères ou d'autres façons de modéliser le problème. Par exemple,

- Proposer une solution permettant au robot de partir d'un nœud initial, d'arriver à un nœud fin, en passant par des points particuliers précisés et non ordonnés, de manière optimale (typique d'une tournée de facteur par exemple).
- Proposer une solution si les ressources du robot sont limitées et qu'il peut par exemple effectuer une distance limitée (carburant limité).
- On imagine maintenant un rover sur mars, qui doit effectuer des points de mesures de plus ou moins grande importance, avec une quantité de ressource limitée. Imaginer un algorithme permettant de résoudre sur problème en maximisant les récompenses sur les points de mesures tout en garantissant que les ressources seront suffisantes.

Annexe

MANIPULATION A

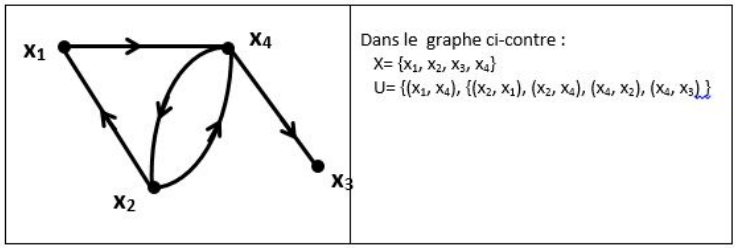
Annexe : Théorie des graphes - Algorithme de Dijkstra

La théorie des graphes est un outil privilégié de modélisation et de résolution de problèmes de décision dans un très grand nombre de domaines allant des sciences fondamentales (physique, informatique) aux applications technologiques les plus concrètes (conception de réseaux de télécommunications, problèmes d'affectation, GPS et recherches de plus courts chemins, etc.). Les concepts de base de la théorie des graphes sont présentés en détail dans le cours Graphes du semestre 2 de la 4AE. Ne sont donc introduits ici que les éléments indispensables pour la mise en oeuvre de l'algorithme de Dijkstra permettant de rechercher le plus court chemin entre un sommet et tous les autres dans un graphe.

A.1 Concepts de base sur les graphes

Graphe, sommets, arcs

Un graphe $G(X, U)$ est constitué
- d'un ensemble de sommets $X = \{x_1, x_2, \ldots, x_n\}$
- d'un ensemble d'arcs reliant ces sommets $U = \{u_1, u_2, \ldots, u_m\}$. Un arc u est défini par un couple de sommets : $u = (x_i, x_j) x_i \in X; x_j \in X$. x_i est l'origine de l'arc u et x_j son extrémité.



Longueur d'un arc et graphes valués

Il est possible d'associer une valeur l_{ij} à chaque arc (x_i, x_j). Cette valeur (qui peut être positive, négative ou nulle) peut par exemple représenter une distance, ou une vitesse, ou un coût ou toute autre grandeur représentative du problème modélisé par le graphe. On parle alors de graphe valué.

Chemins et circuits

Un chemin est une suite d'arcs telle que l'extrémité d'un arc est l'origine de l'arc suivant dans ce chemin. Ainsi par exemple, $(X_2 - X_1 - X_4)$ est un chemin entre les sommets x_1 et x_4.

Un circuit est un chemin dont l'extrémité finale est aussi l'origine du chemin. Ainsi par exemple, $(X_2 - X_1 - X_4 - X_2)$ est un circuit.

La longueur d'un chemin (ou d'un circuit) est égale à la somme des longueurs des arcs qui composent le chemin (ou le circuit).

A.2 Recherche de plus courts chemins

Une problématique classique dans un graphe valué est de rechercher le plus court chemin entre un sommet du graphe que l'on notera s (sommet source) et tous les autres. La théorie des graphes propose différents algorithmes de recherche de plus courts chemins adaptés aux propriétés du graphe. Dans le cas particulier où toutes les longueurs d'un graphe valué sont positives, on peut appliquer l'algorithme de Dijkstra.

Cet algorithme est itératif. On associe 3 informations à chaque sommet xi :
- λ_i : longueur du plus court chemin trouvé à l'itération courante entre s et x_i
- q_i : dernier sommet avant x_i sur ce chemin (permet de reconstituer le plus court chemin à la fin de l'algorithme)
- $m_i \in \{0, 1\}$: le sommet x_i est dit "marqué" lorsque $m_i = 1$. Il est dit "non marqué" lorsque $m_i = 0$. Lorsque le sommet

est marqué, λ_i correspond à la longueur du plus court chemin entre s et x_i (λ_i a sa valeur définitive).

Principe général de l'algorithme

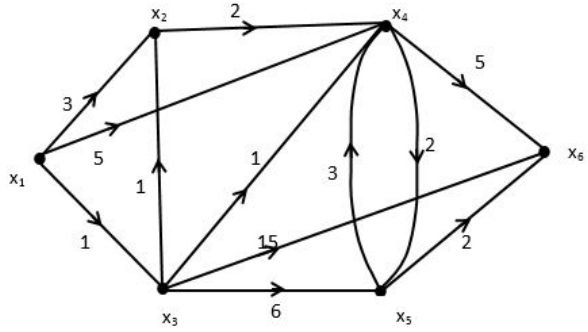
- Initialisations : λ_s est initialisé à 0 (le plus court chemin entre s et s est de longueur nulle) ; les $\lambda_i, \forall i \neq s$ sont initialisé à ∞ (les plus courts chemins entre s et x_i ne sont pas connus en début d'algorithme : leur longueur est donc ∞) ; ces longueurs ne sont pas définitives $\Rightarrow m_i = 0 \forall i$
- A chaque itération :
  - On repère le sommet non encore marqué ($m_i = 0$) qui a le plus petit λ.
  - On marque ce sommet ($m_i = 1$) car, compte tenu de la propriété du graphe, on peut être sûr qu'il n'existe pas de plus court chemin entre s et ce sommet.
  - On essaie d'actualiser les λ des sommets suivants immédiats du sommet qui vient d'être marqué (cf. algorithme détaillé).
  - On recommence une nouvelle itération jusqu'à ce que tous les sommets soient marqués ($m_i = 1$).

Algorithme détaillé

```
-- initialisations
λs ← 0 ; qs ← s ; ms ← 0
pour tout i de 1 à n avec i ≠ s répéter
    λi ← ∞ ; qi ← ϕ ; mi ← 0
fp
-- itérations
tant qu'il existe un sommet non marqué (mi = 0) répéter
    Choisir parmi les sommets non marqués celui dont le λ est minimum ; soit xi ce sommet
    Marquer ce sommet : mi ← 1
    -- actualiser éventuellement les λ des sommets suivants xi
    Pour tout sommet xj ∈ S(i) et tel que mj = 0
        si λi + lij < λj alors
            λj ← λi + lij
            qj ← i
        fsi
    fin pour
fin tant que
Remarque : on peut alors reconstituer le plus court chemin entre s et xi en utilisant qi
```

Mise en oeuvre sur un exemple

Considérons le graphe valué dessiné ci-dessous. On recherche les plus courts chemins entre s=x1 et les autres sommets du graphe.



Le tableau ci-dessous détaille les différentes itérations de l'algorithme. Les cases grisées montrent le sommet qui est marqué au début de chaque itération (sommet non marqué ayant le plus petit λ. Les valeurs actualisées au cours de l'itération des λ et m des sommets suivants celui qui est marqué apparaissent sur la même ligne.

	x_1			x_2			x_3			x_4			x_5			x_6		
	λ	q	m	λ	q	m	λ	q	m	λ	q	m	λ	q	m	λ	q	m
initialisations	0	x_1	0	∞	-	0	∞	-	0	∞	-	0	∞	-	0	∞	-	0
Itération 1	0	x_1	1	**3**	**x_1**	**0**	**1**	**x_1**	**0**	**5**	**x_1**	**0**	∞	-	0	∞	-	0
Itération 2				**2**	**x_3**	**0**	1	x_1	1	**2**	**x_3**	**0**	**7**	**x_3**	**0**	**16**	**x_3**	**0**
Itération 3				2	x_3	1				2	x_3	0	7	x_3	0	16	x_3	0
Itération 4										2	x_3	1	**4**	**x_4**	**0**	**7**	**x_4**	**0**
Itération 5													4	x_5	1	**6**	**x_5**	**0**
Itération 6																6	x_5	1

Reconstitution des plus courts chemins :

Par exemple le plus court chemin entre x_1 et x_6 est de longueur 6 ($\lambda_6 = 6$). Le dernier sommet avant x_6 sur ce chemin est x_5 (car $q_6 = x_5$). Le dernier sommet avant x_5 sur ce chemin est x_4 (car $q_5 = x_4$). Le dernier sommet avant x_4 sur ce chemin est x_3 (car $q_4 = x_3$). Le dernier sommet avant x_3 sur ce chemin est x_1 (car $q_3 = x_1$). Le plus court chemin est donc : $x_1 - x_3 - x_4 - x_5 - x_6$.

A.3 Références bibliographiques

★ Notice NXTway-GS Model-Based Design - Control of self-balancing two-wheeled robot built with LEGO Mindstorms NXT, Yorihisa Yamamoto.

★ Réalisation, réduction et commande des systèmes linéaires, A. Rachid, D. Medhi, Technip, Collection Méthodes et techniques de l'ingénieur (Paris)