M2 ILIADE CPS

Projet : Boussole avec robot EV3 et RobotC

Présentation du matériel

La gamme des `Lego Mindstorms EV3` permet la construction de robots divers qui perçoivent leur environnement à l'aide d'informations issues de capteurs, les traitent à l'aide d'un micro-contrôleur et agissent à l'aide d'actionneurs, des servomoteurs.

La brique EV3

Les robots `EV3` sont équipés d'un microcontrôleur de type ARM doté d'un Linux embarqué.

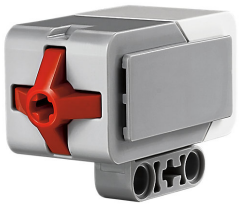
La façade de la brique comporte un écran de 178x128 pixels et six boutons dits `gauche`, `droit`, `haut`, `bas`, `centre` et `retour`, qui permettent de dialoguer avec la brique et de la configurer.

La brique comporte également un certain nombre de ports d'entrée/sortie : un port PC mini-USB, un port USB hôte, un port carte SD, 4 ports d'entrée (1,2,3,4) et 4 ports de sortie (A,B,C,D).

Les capteurs pour le projet

Le capteur de contact indique si on appuie dessus, si on le relâche ou si on lui donne une impulsion. Le gyromètre permet de mesurer les vitesses de rotation autour de l'axe vertical ainsi que l'angle de rotation cumulé (à partir d'une position choisie). Il peut donc servir de

compas relatif (on peut grâce à lui demander au robot de tourner de n degrés vers la gauche ou vers la droite par exemple).

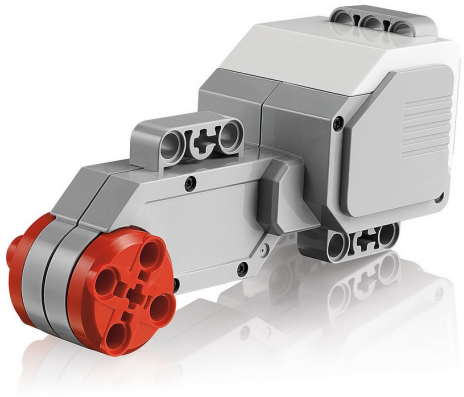


Contact

Gyro

Les actionneurs

Le robot Lego dispose de deux types d'actionneur, nous n'utiliserons que les grands moteurs (des servomoteurs) qui sont dotés d'encodeurs optiques permettant de mesurer la position angulaire des roues ainsi que leur vitesse de rotation.



RobotC

Nous allons utiliser l'IDE **RobotC**.
Pour cela il suffit de lancer *RobotC for Lego Mindstorm* à partir du menu *Tous les programmes* de Windows.
Pour plus d'informations consulter le cours sur RobotC disponible sur le Moodle.
Un manuel de référence est disponible sur https ://www.robotc.net/WebHelpMindstorms/index.htm.

Préliminaires

1. Ecrire un programme RobotC pour faire tourner un moteur dans le sens des aiguilles d'une montre pendant 4 secondes et l'arrêter.

2. Faire tourner un moteur dans le sens inverse des aiguilles d'une montre et l'arrêter sur pression sur le capteur de contact.
3. Faire démarrer le moteur dans un sens sur appui sur le capteur de contact. Ensuite changer de sens à chaque appui sur le capteur de contact. Deux appuis rapprochés signifiant l'arrêt du moteur.
4. Utiliser maintenant les boutons de la brique pour commander le moteur. Le bouton droit sert à faire tourner le moteur dans le sens des aiguilles d'une montre. Le bouton gauche sert à faire tourner le moteur dans l'autre sens. Le bouton haut sert à accélérer le mouvement. Le bouton bas sert à ralentir le mouvement. La puissance moteur va de 0 à 100, la diminution ou l'augmentation se fera par pas de 10.
5. Un peu de programmation multitâche. Fixer le gyromètre à la brique EV3. Ecrire une tâche qui observe à 10Hz la vitesse de rotation et garde en mémoire le min et le max. Ecrire une deuxième tâche qui stocke dans un fichier la vitesse observée toutes les minutes. Ecrire enfin une troisième tâche qui observe le Touch Sensor pour interrompre le programme lorsqu'on appuie dessus. Le min et le max devront être sauvegardés à la fin du fichier des vitesses observées avant de quitter le programme.

Projet : boussole

Montage du robot

Pour ce projet nous allons utiliser une brique, un moteur, un capteur de contact ainsi qu'un gyromètre. Monter le gyromètre sur l'axe du moteur, de cette manière le gyromètre va tourner en même temps que le moteur, puis solidariser le tout (moteur et gyromètre) avec la brique elle-même afin que l'ensemble soit stable. Brancher le gyromètre à la brique avec le plus long câble possible.

Caractérisation du moteur

Cette partie constitue un travail préliminaire servant à déterminer un certain nombre de constantes utiles pour le projet en lui-même. Le code doit donc être développé dans un fichier à part. Pour cette partie nous n'utiliserons pas le gyromètre, pour éviter que son câble gêne le plus simple est de le débrancher. Il s'agit ici de déterminer la vitesse de rotation du moteur en fonction de la puissance. On supposera que la batterie est pleinement chargée (car la puissance réelle varie en fonction de la charge).

Ecrire une fonction `motorCharacterization()` qui tabule pour toutes les puissances possibles (entre 0 et 100 par pas de 1) la vitesse du moteur en degrés par seconde (on supposera que le moteur est symétrique et qu'il tourne dans le sens où sa vitesse est positive). Enfin toutes les données tabulées doivent être sauvegardées dans un fichier pour l'étude qui suit.

Avec l'outil de votre choix (Excel, gnuplot, python ...) tracer la courbe de la vitesse moteur en fonction de la puissance. On remarquera que la vitesse de rotation du moteur

semble proportionnelle à la puissance reçue. Cependant au-delà d'une certaine puissance le moteur ne tourne pas plus vite, il arrive à la limite de ses capacités. Déterminer, visuellement, la puissance minimale `minPower` à laquelle le moteur commence à tourner, puis, la plus petite puissance `maxPower` donnant la vitesse maximale `maxSpeed`. Déterminer enfin la pente de la droite passant par les points `(0,0)` et `(maxPower,maxSpeed)`.

Commande en vitesse du moteur

A l'aide des données précédemment obtenues, écrire la fonction `launchMotorSpeed(speed)` qui lance le moteur à la vitesse `speed`. Si `|speed|>maxSpeed` le moteur sera lancé à la vitesse `maxSpeed`. Cette fonction, contrairement à la précédente, fait partie du code du projet.

Suivi de consigne pour un robot fixe

L'idée est de faire tourner le moteur de 10° vers la droite sur appui du bouton droit de la brique, de 10° vers la gauche sur appui du bouton gauche, de 90° vers la droite sur appui du bouton haut et de 90° vers la gauche sur appui du bouton bas. Ces changements de consigne se feront avec les fonctions RobotC `setMotorTarget` ou `moveMotorTarget` et une puissance égale à 20.

Comme un changement de 90° peut prendre un peu de temps et que nous voulons pouvoir prendre en compte deux appuis successifs sur le bouton haut par exemple, il va falloir faire de la programmation multitâche.

Ecrire une tâche `keepHeading()` qui effectue en boucle le changement de consigne. Si la consigne ne change pas et qu'elle est jugée atteinte, ne pas envoyer de commande moteur pour éviter des micros corrections permanentes.

Ecrire une deuxième tâche `watchButtons()` qui scrute les boutons de la brique pour faire le suivi de consigne. Le programme doit se terminer sur appui du bouton central. Attention la consigne est une variable utilisée en écriture par `watchButtons()` et en lecture par `keepHeading()`.

Ecrire une tâche `IHM()` qui explique à l'utilisateur à quoi servent les boutons et qui affiche la consigne ainsi que le cap atteint par le moteur.

Faire ensuite fonctionner le tout dans la tâche principale.

Régulation

On va écrire notre propre commande de suivi de consigne via un contrôleur PD basé sur le calcul de la vitesse moteur suivant :

$$Vitesse = P.(consigne - capActuel) + D.vitesseAngulaire$$

où la *vitesseAngulaire* et le *capActuel* sont délivrés par l'encodeur optique du moteur. La qualité de ce type de contrôleur dépend directement du choix des coefficients P et D (qui sont

des réels pouvant être négatifs). Des méthodes existent pour aider l'automaticien à trouver de bons coefficients mais ce n'est pas l'objet de ce projet.

Ecrire une tâche `keepHeadingPD()` effectuant la même chose que la fonction `keepHeading()` mais en utilisant `launchMotorSpeed` et la loi PD ci-dessus. On s'arrangera également pour stopper le moteur lorsque la consigne est jugée atteinte.

Réutiliser les tâches `watchButtons()` et `IHM()` pour faire fonctionner le tout.

Faire quelques essais pour proposer des coefficients P et D qui paraissent bons et les mettre en dur dans le code.

Suivi de consigne pour un robot mobile

Pour la suite il est nécessaire de brancher le câble reliant le gyromètre à la brique.
Si l'encodeur optique convient dans le cas d'un robot fixe, le gyromètre est incontournable si le robot devient mobile. Si le robot peut changer de direction indépendamment du moteur supportant le gyromètre (par exemple en déplaçant le robot à la main), l'encodeur optique ne peut mesurer cette rotation alors que le gyromètre lui en est capable.

Caractérisation du gyromètre

Cette partie est à développer dans un fichier séparé. Comme pour la caractérisation du moteur, elle ne sert qu'à déterminer un certain nombre de constantes utiles pour le projet lui-même.

Comme nous allons utiliser la même loi PD que précédemment rien ne sert de demander au moteur d'aller très vite si le gyromètre n'est pas capable de mesurer cette vitesse.

Ecrire une fonction `gyroRateCharacterization()` qui tabule pour toutes les puissances possibles (entre 0 et 100 par pas de 1) la vitesse délivrée par le gyromètre en degrés par seconde (on supposera que le moteur est symétrique et qu'il tourne dans le sens où sa vitesse est positive). Contrairement à la caractérisation du moteur il faudra songer à faire des allers retours pour éviter l'enroulement du câble. Toutes les données collectées doivent être copiées dans un fichier avant de quitter la fonction.

Comme pour le moteur, tracer la courbe de la vitesse en fonction de la puissance et déterminer la puissance minimale `minPower2` à laquelle le moteur commence à tourner, puis, la plus petite puissance `maxPower2` correspondante à la vitesse maximale `maxSpeed2` ainsi que la pente de la droite entre les points `(0,0)` et `(maxPower2,maxSpeed2)`.

Commande en vitesse du moteur

A l'aide des données précédemment obtenues, écrire la fonction `launchMotorSpeed2(speed)` qui lance le moteur à la vitesse `speed`. Si `|speed|>maxSpeed2` le moteur sera lancé à la vitesse `maxSpeed2`. Cette fonction, contrairement à la précédente, fait partie du code du projet.

Boussole

On veut maintenant que, à la manière de l'aiguille d'une boussole, le moteur pointe toujours dans la même direction même si on tourne le robot à la main. Ecrire une nouvelle version de `keepHeadingPD()` appelée `keepHeadingPD2()` qui pour la régulation utilise le cap et la vitesse de rotation délivrés par le gyromètre plutôt que ceux délivrés par l'encodeur optique. `keepHeadingPD2()` utilisera également `launchMotorSpeed2`. Tester le comportement du robot en le tournant à la main et en vérifiant que le gyromètre pointe bien toujours dans la même direction.

Ecrire une deuxième tâche `watchButton2()` qui quitte la régulation sur appui du bouton central.

Ecrire une tâche `IHM2()` qui explique à l'utilisateur comment quitter le programme et qui affiche la consigne ainsi que le cap atteint par le moteur.

Faire ensuite fonctionner le tout dans la tâche principale.

Initialisation

Lorsque le gyromètre tourne le câble s'enroule autour de l'axe et peut finir par freiner voire bloquer le moteur. Pour anticiper cela on souhaite pouvoir initialiser la position du gyromètre où bon nous semble avant de commencer l'action voulue. Ecrire une fonction `initialize()` qui fait tourner le moteur dans le sens inverse des aiguilles d'une montre avec une puissance de 20 sur appui du bouton gauche, dans le sens opposé sur appui du bouton droit et quitte le programme sur appui du bouton central. Prévoir une petite interface pour expliquer à l'utilisateur de quoi il s'agit et l'utilité des boutons.

Interface

Le programme final devra, au lancement, proposer un menu avec deux choix possibles : `Robot fixe` sur appui du bouton gauche, `Robot mobile` sur appui du bouton droit. Chacun de ces choix doit commencer par proposer une phase d'initialisation à l'utilisateur pour positionner le gyromètre à l'endroit souhaité (surtout par rapport au câble) puis une fois celle-ci achevée poursuivre par l'action associée. Dans les deux cas l'appui sur le bouton central doit permettre de revenir au menu initial. Enfin, on pourra interrompre le programme à n'importe quel moment sur appui du capteur de contact.

Travail à rendre

Vous devez rendre un rapport au format pdf contenant :

1. le code commenté de la caractéristaion du moteur
2. le graphe commenté de la vitesse moteur en fonction de la puissance fournie
3. le code commenté de la caractéristaion du gyromètre

4. le graphe commenté de la vitesse donnée par le gyromètre en fonction de la puissance fournie
5. les deux graphes précédents superposés sur un même graphique avec un commentaire sur la comparaison des deux capteurs : l'encodeur optique et le gyromètre
6. le code commenté du projet boussole

Le rapport **au format pdf** sera envoyé par email à l'adresse **goulven.guillou@univ-brest** pour le 16 novembre 2022 au plus tard. D'autre part la séance de TP du 16 novembre (10h15-12h15) sera consacrée à la démonstration de vos programmes et à la restitution du matériel.