Partie carte Robot

Ici est donné une présentation de la carte robot avec les différents modules auxquels elle est attachée.

Caractéristiques techniques

https://drive.google.com/file/d/1o3b0Ypi8UGPQKdmxpCNDQjcIiINDcvSg/view

Caractéristiques propres

Plusieurs caractéristiques propres du strandbeest nous ont contraint à nous orienter vers des modes de fonctionnement très particulier.

Le robot n'est pas équipé de roue mais de sorte de pattes arachnides, qui introduit beaucoup de vibrations dans la structure lorsqu'il se déplace. De plus, le fait de ne pas avoir de roue ne nous permet pas de pouvoir nous localiser dans l'espace à l'aide du rayon des roues et du nombre de tour effectué.

Les moteurs, pour une même tension donnée, ne tournent pas à la même vitesse, et le strandbeest n'est pas équipé d'accéléromètre au niveau des moteurs.

Il nous ait donc impossible de connaître la vitesse réelle des moteurs.

L'accéléromètre présent sur le module MPU6050 n'est pas assez précis (il fluctue aléatoirement autour de la valeur réelle) pour nous permettre d'intégrer la valeur en fonction du temps et en déduire la position dans l'espace.

Choix techniques

Ne pouvant pas connaître la position réelle du robot nous avons opté pour un modèle de cases au niveau du mapping (expliqué plus en détail dans la partie Raspberry).

Ce choix nous permet de tolérer des imprécisions sur le déplacement du strandbeest (la case ayant de plus grande dimensions que le robot, celui-ci a de la "marge" au niveau de son placement, on peut tolérer des petits écarts).

Mais il est important que ces imprécisions restent assez petites et qu'elles ne s'accumulent pas en fonction du temps à défaut de se positionner de se positionner sur une mauvaise case sans le savoir.

- Angle:

Le seul paramètre sur lequel nous pouvons nous baser pour vérifier que le robot se déplace bien comme nous le souhaitons est son angle de "lacet" (dit "yaw" en anglais). Celui-ci peut-être calculer de manière précise à l’aide du module MPU6050 grâce à une méthode nommée « filtre complémentaire » (voir <http://www.pieter-jan.com/node/11>). En résumé, nous pouvons de base calculer l’angle à l’aide de l’accéléromètre et du gyroscope du module MPU6050, mais chaque capteur possède des problèmes : le gyroscope est très précis mais dérive progressivement en fonction du temps ; tandis que l’accéléromètre est trop sensible ce qui fait que l’angle calculé n’est jamais très précis et se situe dans une fourchette de valeurs trop large pour être exploité, mais cette fourchette ne dérive pas (ou peu) en fonction du temps. La méthode dites « filtre complémentaire » consiste à combiner ces deux angles calculés via la formule suivante : angle= (1-x) \* (angle +Gyr \* dt) + x \* Acc, avec x une faible valeur (en générale comprise entre 0 et 0,1), Acc l’angle calculé par l’accéléromètre, Gyr l’angle calculé par le gyroscope, et dt le temps entre deux mesures. Après plusieurs tests, nous avons trouvé une valeur de x optimale égale à 0,07.

Petit problème à noter : si l’angle n’est pas fréquemment calculé, sa valeur va progressivement dériver au court du temps. Cela s’explique par le terme *Gyr \* dt* : si un trop grand temps se passe entre deux mesures d’angle, toutes les variations qui aura eu lieu entre ces deux mesures ne seront pas pris en compte et la valeur calculé par le gyroscope sera multiplié par une très grande valeur de temps, ce qui explique la dérive. Nous avons donc dû faire en sorte de calculer l’angle fréquemment dans notre code et de limiter sensiblement l’utilisation des delay.

- Moteurs:

Nous avons étalonné différentes valeurs de PWM des moteurs pour pouvoir connaitre leurs vitesses correspondantes.

Grâce à cela, en affectant une certaine valeur de PWM aux moteurs, qui est associé à une vitesse, et en lui donnant un temps d'exécution, nous pouvons connaître la distance parcourue.

Nous pouvons alors en déduire différents couples vitesse/temps qui permet au robot d'avancer de la taille d'une case.

Pour autant, comme dit précédemment, les moteurs n'ont pas exactement les mêmes performances ce qui fait que le robot ne se déplace pas exactement en ligne droite.

En connaissant l'angle yaw du robot à tout instant, nous pouvons corriger la vitesse des moteurs en temps réel si le robot se trouve à un angle non souhaité.

Nous avons donc asservi à l'aide d'un PID la tension du moteur droit (choix arbitraire) pour qu'il se colle à la vitesse du moteur gauche en prenant comme valeur d'entrée l'angle yaw du robot. A l’aide de cette solution technique, nous pouvons faire avancer notre robot en ligne droite de manière assez précise.

Pour autant, le robot, après s’être déplacé d’une case, peut s’arrêter en introduisant une plus ou moins grande erreur sur l’angle. S’il reprend ensuite sa marche en allant en ligne droite, il va partir dans une direction erronée, et petit à petit, les erreurs vont s’accumuler et le robot aura complètement dévier de là où il devrait être. Pour pallier à ce problème, nous calculons lors de l’allumage du robot son angle d’origine. Lorsque le robot avancera, il devra respecter en consigne cet angle d’origine. Et dans les cas où le robot tournerait à droite ou à gauche, cet angle d’origine se verra ajouter ou enlever 90°. Ainsi des imprécisions pourront intervenir lors de ses déplacements mais la direction dans laquelle le robot devra aller sera toujours connu.