Projet Complexe 3 - Groupe 20
24/03/2025

Guide de fabrication d’un robot souple modulaire

ADHAR Emna
BOTHOREL Alexian
NAVES DE OLIVEIRA ARAÚJO Yuri
PASCANET Achylle

1. Contexte et Objectifs du Projet

1.1. Contexte

Le projet répond au besoin du laboratoire LS2N de concevoir un robot souple actionné par des tendons. Ce robot, inspiré de structures naturelles comme la trompe d'éléphant ou les tentacules de poulpe, possède une structure flexible contrôlée par un réseau de tendons. Il est particulièrement adapté à des applications nécessitant un accès à des espaces confinés ainsi que des manipulations délicates. Parmi ces applications, on peut citer la chirurgie mini-invasive dans le domaine médical, l'industrie, la robotique sous-marine ou encore la récolte agricole.

1.2. Principe de fonctionnement

Le robot est composé de 8 modules identiques :

- 4 modules de translation, permettant de tirer sur les fils
- 4 modules de torsion, permettant de faire tourner les fils
  Chaque fil est ainsi contrôlé par un duo translation + torsion, ce qui permet d'obtenir tous les degrés de liberté nécessaires à la manipulation du robot souple.
- Il y a aussi 4 rails et les constituants communs qui permettent de rassembler les modules

1.3 Ressources nécessaires

- 8 moteurs triphasés (à caractériser : couple, vitesse nominale, dimensions, tension d'alimentation…)
- 1 alimentation AC (spécifications à préciser selon les moteurs et ESC)
- 8 ESC bidirectionnels (Electronic Speed Controller)
- 2 cartes Raspberry Pi Pico
- Fils plastiques (type à préciser)
- 1 imprimante 3D
- 4 fils de pêche (nylon 0,3 mm)
- Vis (une boîte M6 X 14 SOCKET HEAD et une boîte M3X10) et écrous correspondants

2. Contrôle des moteurs

2.1 Partie électronique

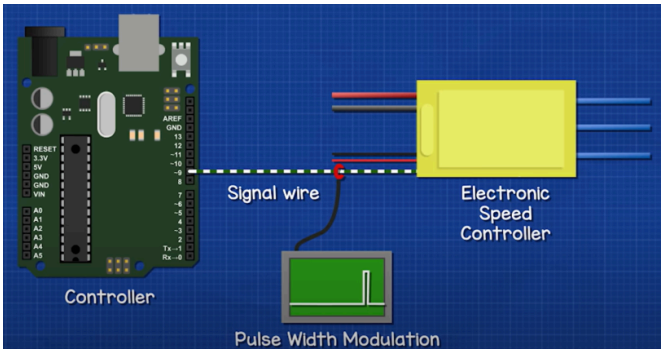
Les moteurs utilisés dans notre robot sont des moteurs triphasés brushless. Ils sont chacun contrôlés à l'aide d'un ESC (Electronic Speed Controller) qui a pour but de prendre en entrée une PWM de commande de type servo moteur et de distribuer 3 signaux déphasés, correspondant à la commande, aux 3 entrées du moteur. Ceux-ci sont de forme suivante :

Ceux-ci possèdent :

- 2 câbles d'alimentation (rouge et noir) à reliés au GBF ou à la source d'alimentation prévu.
- 3 câbles (noirs) à relier aux trois entrées du moteur (l'ordre de branchement des câbles impacte le sens de rotation du moteur).
- 3 câbles pour le contrôle du moteur avec un câble noir de GND, un câble rouge d'alimentation (utilisé uniquement lorsque le moteur fonctionne avec un potentiomètre) et un câble (blanc ou autre) permettant de contrôler la vitesse et le sens de rotation du moteur à l'aide d'une PWM de type servomoteur.

L'ampérage maximum supporté par nos ESC est de 1.2A ce qui n'a jamais en pratique été approché au cours du projet. De plus, le voltage que nous utilisions en entrée de l'ESC était d'environ 20V.

2.2 Partie signal de contrôle



Ainsi, l'ESC demande une PWM (Pulse Width Modulation) de 50 Hz de fréquence et avec une période haute entre 1 et 2 ms. 1,5 ms étant la position neutre (le moteur ne bouge pas) et avec le moteur tournant dans un sens entre 1 et 1,5 ms, puis dans l'autre sens entre 1,5 et 2 ms (ce sens de rotation peut simplement être modifié en échangeant deux câbles d'alimentation d'un moteur).

Dans notre cas, nous n'avons pas besoin d'une grande vitesse de rotation et nous sommes donc resté dans les bornes de [1,4;1,6]ms pour la partie haute de notre PWM. Toutefois, nos modules ayant des performances variables pour le

déplacement longitudinal, du au frottements pouvant varier d'un module à l'autre et directement aux performances moteur changeantes, nous avons dû utiliser des périodes hautes différentes et ne pouvons donc pas généraliser les commandes à chaque moteur. Ce problème devrait disparaître avec l'intégration de régulation dans le système.

2.3 Code

Il ne suffit donc que d'un signal PWM pour contrôler un moteur ce qui permet de facilement contrôler les 8 moteurs avec un seul microcontrôleur (dans notre cas une Raspberry Pico W). Le code en micro-Python de base utilisé pour générer ce signal est le suivant :

```python
from machine import Pin, PWM

# Configuration de la broche PWM
pwm_pin = Pin(15)  # Remplacez 15 par le numéro de votre broche
pwm = PWM(pwm_pin)
pwm.freq(50)  # Fréquence de 50 Hz (période de 20 ms)
pulse = 1500


def set_pwm_pulse_width_us(pulse_width_us):
    """
    Définit la durée du signal haut en microsecondes (entre 1000 et 2000 us).
    """
    # Calcul du duty cycle en fonction de la période de 20 ms (20000 us)
    duty_cycle = int(pulse_width_us / 20000 * 65535)
    pwm.duty_u16(duty_cycle)

set_pwm_pulse_width_us(pulse)
```

Ce système de fonctionnement peut simplement être multiplié pour faire fonctionner plusieurs moteurs en parallèles (cf. *Scenario4Moteurs.py* et *Scenario4MoteurBis.py*).

2.4 Autres systèmes explorés

Lors de notre projet nous avons aussi essayé d'utiliser d'autres moyens pour contrôler nos moteurs qui n'ont pas fonctionné pour des raisons différentes. Ceux-ci étant :

- Launchpad et Booster par Texas Instrument
- Escon Module par Maxon Motor
- Codeur AEDM par Avago Technologies

Launchpad et Booster par Texas Instrument

Le projet d'origine dont est tiré le modèle mécanique du projet (cf. annexes) fonctionne avec des Launchpad et des Booster Texas Instrument pour faire fonctionner les moteurs. Toutefois, nous n'avons pas réussi à les faire fonctionner à notre convenance dû à la documentation peu fournie et difficilement compréhensible. Nous avons donc abandonné cette solution.

Escon Module par Maxon Motor

Nous avons ensuite essayé d'utiliser un Escon module de chez Maxon Motor que nous n'avons pas gardé car l'interface proposée par la marque rendait l'utilisation des moteurs peu intuitive et la parallélisation des moteurs compliqué. Nous sommes alors passé à la Raspberry Pico, sachant qu'il est aussi possible de l'échanger avec une carte Arduino car elles ont des modes de fonctionnement très similaires.

Codeur AEDM par Avago Technologies

Finalement, nous avons aussi chercher à implémenter les codeurs AEDM présent dans le projet original pour pouvoir réaliser une régulation en boucle fermée pour le contrôle de nos moteurs. Nous avons donc réaliser un code permettant de compter les impulsions détectées par le codeur et de déterminer le sens de rotation (cf. Codeur.py) mais nous n'avons pas pu l'implémenter à cause de trop grandes erreurs sur les mesures effectuées. Celles-ci sont probablement dû à un problème de glissement entre le moteur et l'arbre, puis entre l'arbre et le codeur.

3. Développement de la partie mécanique

3.1. Développement et production des pièces

La plupart des composants étaient déjà préconçus et notre équipe s'est contentée d'en adapter certains ou d'en créer de nouveaux afin d'accroître la robustesse et la résistance du robot.

Tous les dessins techniques sont dans notre drive, avec non seulement le dessin de chaque composant, mais aussi la représentation de son assemblage, réalisé dans Inventor AutoDesk. Pour accéder au lecteur, il suffit de suivre ce lien. Ce même lien mène aux manuels d'assemblage des modules d'actionneurs, ainsi qu'à l'assemblage des capteurs.

Pour reproduire un tel projet, il faut d'abord imprimer les éléments structurels à l'aide d'une imprimante 3D. Vous pouvez visualiser et connaître tous les composants nécessaires en consultant le fichier "Assemblage complet" dans le fichier "Toutes les éléments".

3.2. Procédure d'assemblage

Ensuite, il est nécessaire de respecter l'ordre d'assemblage des modules de l'actionneur, tel qu'indiqué dans le manuel d'instructions d'assemblage de l'actionneur.

Une fois les modules actionneurs correctement assemblés, l'ordre d'assemblage suivant est recommandé pour le robot :

1. Assembler "Wall_of_platform" avec "Head_of_platform".
2. Fixer le support de crémaillère ("Support crémaillère") à la base frontale ("Head_of_platform").
3. Insérer les supports de rail ("Pillow_blocks") dans les rails.

4. Insérer les supports de rails dans la base frontale et les visser.
5. Insérer les charriots déjà assemblés avec les modules dans les rails.
6. Insérer les supports de crémaillère dans la base arrière (“Nouvelle base courte”) (il s'agit d'une version améliorée que nous avons décidé d'utiliser dans notre projet)
7. Connexion de la base arrière à la base frontale + ensemble de rails
8. Une fois les deux moitiés assemblées, placez-les l'une au-dessus de l'autre et fixez-les à l'aide de boulons et d'écrous aux endroits prévus à cet effet.

Pour le bras, utilisez une poutre en fibre de carbone de 3 mm et collez les disques ensemble avec de la colle forte en fonction de la distance entre les disques que vous jugez idéale. Nous avons utilisé 12 cm entre chaque disque. Ensuite, attachez le fil de pêche au dernier disque et fixez-le aux actionneurs avant. Pour ce faire, nous avons percé un trou dans le "shaft_gear" qui permet d'attacher et de détacher le fil de pêche.

Pour fixer le bras au robot, nous avons percé 2 trous diamétralement opposés dans le disque de base, et 1 trou dans chaque base avant pour fixer le disque de base à la structure à l'aide d'une vis et d'un écrou.

4. Procédure

Procédure pour la mise en fonctionnement de notre robot :

1. Connecter le moteur à l’ESC bidirectionnel.
2. Relier l’ESC à l’alimentation ainsi qu’aux broches appropriées de la carte Raspberry Pi Pico.
3. faire de même pour chaque moteur
4. Dans l’environnement Thonny, écrire le code en MicroPython (cf. 2.3).
    - Le code permet de modifier la vitesse et le sens de rotation du moteur.
5. Téléverser le code sur la carte Pico.
6. Assemblage mécanique des pièces (cf. 3.2).
7. Ajout des 4 chariots dans leurs rails respectifs en faisant attention à leur orientation.
8. Fermeture du boitier et ajout du bras.
9. Mise en place des tendons aux vertèbres voulus et tension de ceux-ci.

5. Conclusion

En conclusion, ce projet constitue une base solide pour la conception d’un robot souple actionné par des tendons, reproductible et viable, et dont la structure est hautement personnalisable grâce à l'impression 3D de ses composants. Ce type de système offre de nombreuses perspectives d'applications dans des domaines variés tels que le médical ou l'industrie. De plus, il est possible d’envisager une évolution du système en remplaçant les fils par des tiges rigides, actionnées non seulement en translation par les moteurs existants, mais également en torsion grâce à l’ajout de moteurs dédiés. Cette configuration permettrait d’introduire des rotations du robot autour de lui-même, élargissant ainsi considérablement ses capacités de mouvement. Toutefois, pour garantir son efficacité et sa fiabilité, des études supplémentaires restent nécessaires, notamment en ce qui concerne la mise en place d’une régulation et le développement d’un modèle de contrôle adapté.

Annexes

Documentation sur un projet utilisant les mêmes moteurs :
https://github.com/open-dynamic-robot-initiative/open_robot_actuator_hardware/tree/master

Liens vers la documentation du code du même projet :
https://open-dynamic-robot-initiative.github.io/documentation_portal/

Documentation d'un autre projet similaire au nôtre :
https://github.com/ContinuumRoboticsLab/OpenCR-Hardware/blob/main/mechanics/LOTR_TDCR-spatial/README.md

Datasheet des encodeurs:
https://www.mouser.com/datasheet/2/678/avgo_s_a0001422768_1-2290945.pdf?srsltid=AfmBOoo6IEAnr1tjLinl_LVz-lsKy8QvK8D0zakHaQNjzNFqGXHODZ3Z

Datasheet de la carte pico: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

Dossier avec des dessins mécaniques :
https://drive.google.com/drive/folders/1-UzpPLCmslqcSs-0DEPWHgyKrePXdWNa?usp=drive_link

Manuels d'assemblage : Unité d'actionnement modulaire
https://drive.google.com/file/d/1cwoCN-Bz9DuOh9sWZBFv4c0yCU_1f9Yh/view?usp=drive_link

Assemblage encodeurs
https://drive.google.com/file/d/1M4ncAQqrbUTjT0UvDa6qT_8cq7Nisqg9/view?usp=drive_link