

Portage architecture Arduino vers ESP32

— Épita —
Équipe Robotique d'Exploration



Tony BERNIS Valentin DUMOUSSET Julien LE QUANG
Merlin VOTAT

31 décembre 2022

Résumé

Micro ROS (Robot Operating System) est un framework pour faciliter le développement robotique et qui sert d'interface entre les microcontrôleurs. Il permet ainsi l'abstraction du matériel, la transmission de messages etc. Micro ROS n'est pas disponible sur toutes les cartes Arduino. Dans notre projet, nous avons choisi une carte Arduino Nano RP2040. L'objectif de notre projet était dans un premier temps de faire bouger l'araignée et dans un second temps travailler sur la synchronisation avec une autre araignée.

Table des matières

Résumé	1
1 Introduction	3
2 État de l'Art	4
2.1 Les solutions	4
2.1.1 Solution 1	4
2.1.2 Solution 2	6
2.2 Installation de ROS 2	6
3 Réalisation	8
3.1 Implémentation technique	8
3.1.1 Les différences entre ROS et ROS2	8
3.1.2 Les mouvements de l'araignée	9
3.2 Synchronisation araignées	13
3.3 Construction de la coque	13
3.4 Limites et problèmes rencontrés	14
4 Conclusion	15

Introduction

ROS2 (Robot Operating System) est un système d'exploitation open source pour les robots. Il est conçu pour faciliter le développement de logiciels pour les robots en offrant des bibliothèques, des outils et des standards communs pour la robotique. ROS2 offre une infrastructure logicielle pour la programmation de robots, y compris des fonctionnalités telles que la communication entre les composants logiciels, la gestion des données, la planification et l'exécution de tâches, la visualisation de données, et bien plus encore. ROS2 est utilisé dans de nombreux types de robots, des petits robots de service aux grands robots industriels.

Micro-ROS est une version légère de ROS2 conçue pour fonctionner sur des systèmes embarqués avec des ressources limitées en termes de calcul et de mémoire. Micro-ROS offre une infrastructure logicielle pour la programmation de robots embarqués, y compris des fonctionnalités similaires à celles de ROS2, telles que la communication entre les composants logiciels, la gestion des données, la planification et l'exécution de tâches. Micro-ROS est conçu pour être facile à utiliser et à intégrer dans des projets de robotique embarqué, en offrant une grande flexibilité et une grande efficacité en termes de ressources.

Le but de notre projet était d'explorer les différentes options qui nous permettraient d'adapter le code des araignées mécaniques sur l'architecture Micro-Ros.

Il y a plusieurs avantages à utiliser Micro-ROS pour programmer une araignée mécanique. Tout d'abord, Micro-ROS est conçu pour fonctionner sur des systèmes embarqués avec des ressources limitées en termes de calcul et de mémoire, ce qui est idéal pour une araignée mécanique qui a des contraintes en termes de taille et de poids. En outre, Micro-ROS offre une infrastructure logicielle pour la programmation de robots. Enfin, Micro-ROS est une version légère de ROS2, ce qui signifie qu'il est moins gourmand en ressources que ROS2 tout en offrant les mêmes fonctionnalités de base. Cela peut améliorer les performances de l'araignée mécanique en utilisant moins de ressources pour exécuter le logiciel.

Dans ce rapport, nous allons présenter toutes les recherches que nous avons effectué pendant ce projet, les différentes solutions que nous avons conceptualisées ainsi que la solution que nous avons choisie et sa mise en œuvre.

Chapitre 2

État de l'Art

2.1 Les solutions

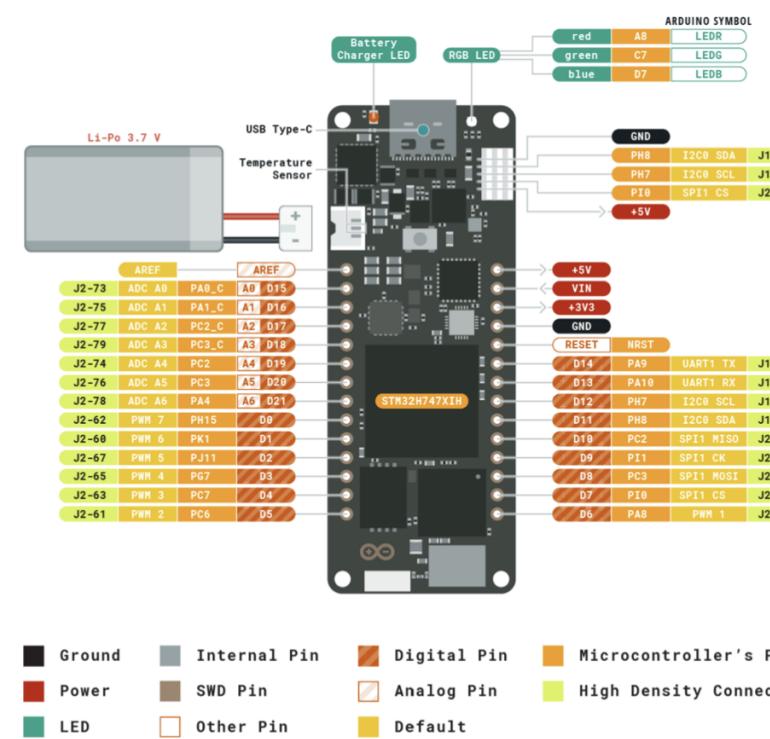
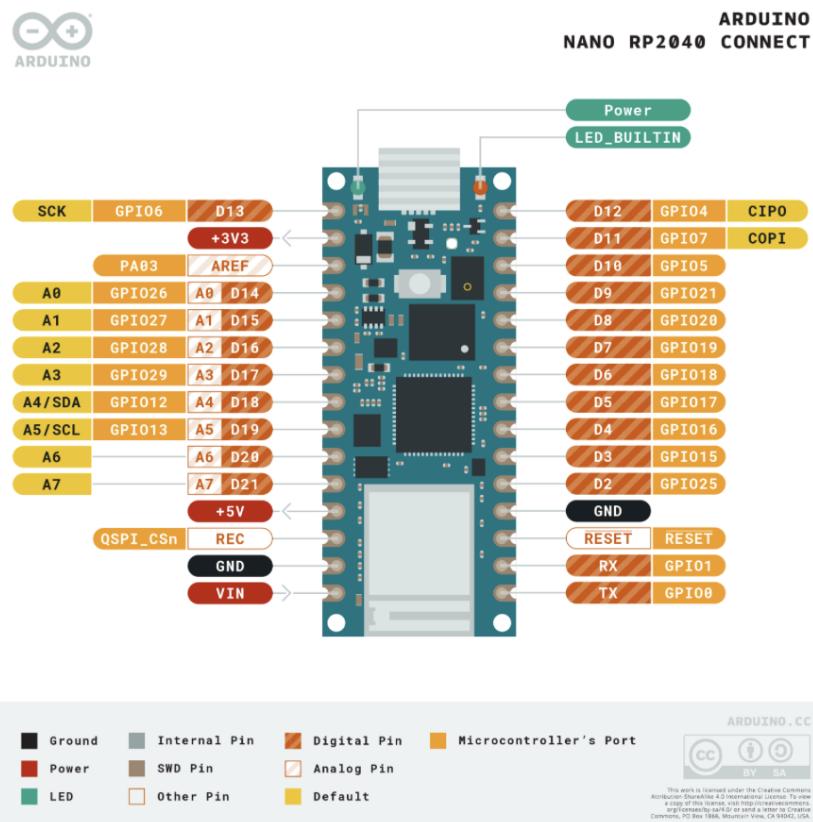
2.1.1 Solution 1

A l'heure actuelle, Micro-ros n'est supporté que par 2 modèles de cartes Arduino, Arduino Portenta H7 M7 Core <https://store.arduino.cc/portenta-h7> et Arduino Nano RP2040 Connect <https://docs.arduino.cc/hardware/nano-rp2040-connect>.

Nous avons d'ailleurs vu qu'une personne a installé Micro ROS sur ces cartes : <https://www.youtube.com/watch?v=mq1uFGsYqeU>.

Il était probablement possible d'adapter Micro-ROS à la carte Arduino que nous avions à notre disposition mais cela risquait de nous prendre plus de temps que nous n'avions à disposition.

L'Arduino Nano RP2040 Connect est une carte de développement basée sur le processeur ARM Cortex-M0+ RP2040 conçue par Raspberry Pi. Elle est conçue pour être petite, flexible et facile à utiliser pour les projets de robotique et d'Internet des objets (IoT). L'Arduino Nano RP2040 Connect est équipée de 20 broches de E/S numériques, de 6 broches PWM, de 6 broches analogiques, d'un port USB-C pour l'alimentation et la communication, d'un connecteur de batterie, d'un connecteur de microphone, d'un connecteur de haut-parleur et d'un connecteur de bouton-poussoir. La carte est également compatible avec les bibliothèques Arduino pour faciliter le développement de logiciels pour les projets de robotique et d'IoT. Après nos différentes recherches, nous avons conclu que ce serait la carte idéale pour mettre en place le projet.



2.1.2 Solution 2

Nous avons cependant pensé à une solution alternative. Nous pouvons aussi utiliser une carte ESP32 qui est déjà compatible avec l'architecture Micro-ROS (<https://micro.ros.org/blog/2020/08/27/esp32>).

ESP32 est une puce de microcontrôleur à double cœur conçue par Espressif Systems. Elle est principalement utilisée dans les applications de robotique, d'Internet des objets (IoT) et de réseaux locaux sans fil (Wi-Fi et Bluetooth). La puce ESP32 est équipée d'un processeur principal Xtensa Dual-Core LX6, d'un processeur coprocesseur ultra-basse consommation, d'une mémoire SRAM de 520 Ko, d'une mémoire flash de 16 Mo, d'un module Wi-Fi 802.11b/g/n/e/i et d'un module Bluetooth v4.2. La puce ESP32 offre également de nombreux E/S numériques, analogiques et PWM, ainsi que des fonctionnalités avancées telles que le traitement du signal numérique, le traitement en temps réel, l'interface de caméra et l'interface de bus série. En raison de ses performances et de ses fonctionnalités avancées, la puce ESP32 est largement utilisée dans de nombreux projets de robotique et d'IoT.

On peut installer cette carte sur notre robot et la brancher directement en SPI (Serial Peripheral Interface) avec la carte Arduino Nano pour ne pas avoir à la connecter en radio ou en wifi. L'ESP32 s'occupera de faire tous les contrôles et enverra des ordres à la carte Arduino qui se contentera de les exécuter ; Les problèmes que l'on pourrait rencontrer avec cette méthode sont le manque de place sur le robot, un poids peut-être trop important, ainsi qu'un manque d'alimentation, l'alimentation actuelle du robot ne sera peut-être pas suffisante pour alimenter les deux cartes.

2.2 Installation de ROS 2

Nous allons détailler dans cette section comment installer ROS 2 sur Linux et Windows.

Sur Linux (Recommandé) :

Suivre la documentation suivante : <https://docs.ros.org/en/rolling/Installation/Ubuntu-Install-Debians.html>

Sur Windows :

1. Téléchargez la dernière version de ROS2 depuis le site Web officiel de ROS (<https://index.ros.org/doc/ros2/>). Sélectionnez la version de ROS2 qui convient à votre version de Windows (32 bits ou 64 bits).
2. Installez les outils de build Colcon en suivant les instructions du site Web de ROS (<https://index.ros.org/doc/ros2/Installation/Colcon/>). Colcon est un outil de build utilisé pour compiler et intégrer les logiciels ROS2.
3. Créez un environnement de développement ROS2 en utilisant l'outil "ament" (<https://index.ros.org/doc/ros2/Installation/Ament/>). Ament est un outil qui permet de gérer les dépendances et les environnements de développement pour les projets ROS2.

4. Installez les bibliothèques de support de Windows pour ROS2 en suivant les instructions du site Web de ROS (<https://index.ros.org/doc/ros2/Installation/Windows-Support/>). Ces bibliothèques sont nécessaires pour exécuter ROS2 sur Windows.
5. Ajoutez les variables d'environnement ROS2 à votre système en suivant les instructions du site Web de ROS (<https://index.ros.org/doc/ros2/Installation/Windows-Environment-Variables/>). Ces variables d'environnement sont nécessaires pour exécuter les commandes ROS2 et accéder aux outils et aux bibliothèques ROS2 depuis n'importe quel emplacement de votre système.
6. Vérifiez que ROS2 est correctement installé en exécutant la commande "ros2 run demo_nodes_cpp talker" depuis une invite de commandes. Cette commande devrait exécuter un noeud "talker" de démonstration qui envoie des messages de test à un noeud "listener" de démonstration. Si la commande s'exécute correctement, cela signifie que ROS2 est correctement installé et configuré sur votre système.
7. Vous pouvez maintenant commencer à développer vos propres applications ROS2 en suivant les tutoriels et les exemples disponibles sur le site Web de ROS (<https://index.ros.org/doc/ros2/>). N'oubliez pas de configurer votre environnement de développement et de compiler vos logiciels avec Colcon avant de les exécuter.

L'étape suivante est de générer l'agent Micro ROS et le lancer. On peut retrouver à cet effet un tutoriel à cette adresse : <https://gist.github.com/Redstone-RM/0ca459c32ec5ead8700284ff56a136f7>

Chapitre 3

Réalisation

Nous avons accompli notre objectif qui était de synchroniser les mouvements de plusieurs araignées entre elles grâce à Micro-ROS. Une démo peut être consultée à cette adresse : <https://www.youtube.com/watch?v=0yxngs2QC1o>. L'araignée est contrôlée par une manette Xbox et il est possible d'augmenter ou diminuer la vitesse des mouvements en appuyant sur les joysticks.

3.1 Implémentation technique

3.1.1 Les différences entre ROS et ROS2

Une des principales différences entre ROS et ROS2 est leur architecture. ROS est basé sur une architecture monolithique, dans laquelle tous les composants logiciels sont intégrés dans un seul et même système d'exploitation. ROS2, en revanche, est basé sur une architecture modulaire, dans laquelle chaque composant logiciel peut être exécuté indépendamment des autres dans un système d'exploitation distribué. Cette architecture modulaire permet une meilleure flexibilité, une meilleure évolutivité et une meilleure tolérance aux pannes que l'architecture monolithique de ROS.

Une autre différence importante entre ROS et ROS2 est leur modèle de communication. ROS utilise un modèle de communication basé sur le publisieur-abonné, dans lequel les composants logiciels peuvent envoyer et recevoir des données en utilisant des "topics" définis. ROS2, en revanche, utilise un modèle de communication basé sur les services et les requêtes, dans lequel les composants logiciels peuvent interagir en invoquant des services et en envoyant des requêtes. Ce modèle de communication permet une meilleure gestion des données et une meilleure flexibilité pour les applications de robotique.

Enfin, ROS et ROS2 diffèrent également en termes de langages de programmation et de plateformes de développement. ROS prend en charge principalement le langage de programmation C++, bien qu'il soit également possible de l'utiliser avec d'autres langages tels que Python. ROS2 prend en charge plusieurs langages de programmation, tels que C++, Python, Java, et bien d'autres. De plus, ROS utilise principalement le système de build Catkin pour la compilation et l'intégration de logiciels, alors que ROS2 utilise l'outil de build Colcon.

3.1.2 Les mouvements de l'araignée

Nous avons implémenté les mouvements de l'araignée pédagogique SEALK dont le code nous a été fourni. Voici la liste des différents mouvements :

sit : fait asseoir l'araignée



FIGURE 3.1 – Sit

stand : met debout l'araignée



FIGURE 3.2 – Stand

walk forward, walk back, walk left, walk right : bouge l'araignée dans les quatre directions

pas d'image :(

turn left, turn right : tourne l'araignée à gauche et à droite

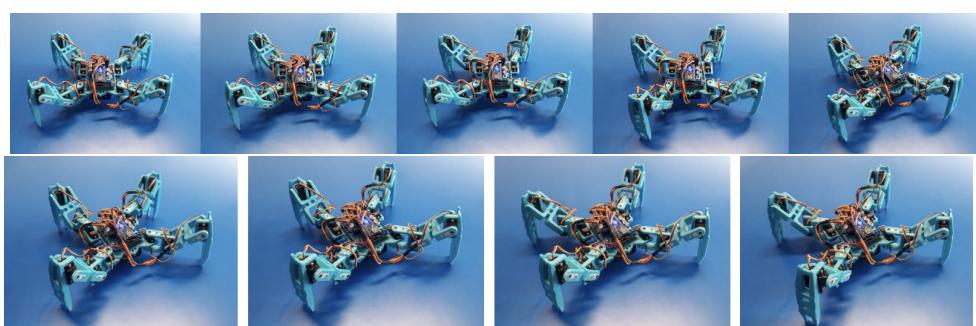


FIGURE 3.3 – Turn left

rave : mouvement de divagation

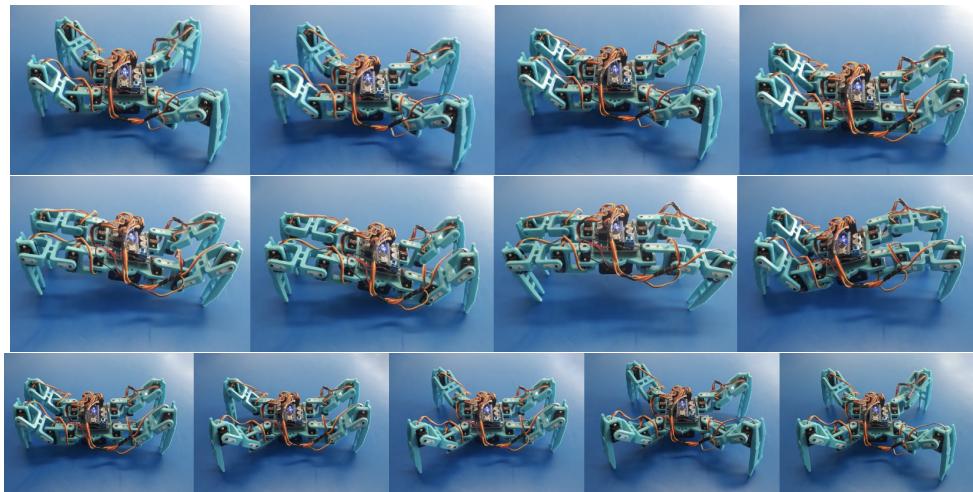


FIGURE 3.4 – Rave

flex : fait un mouvement de "flex" au sol

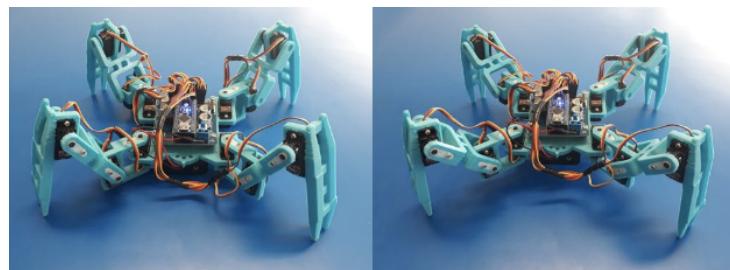


FIGURE 3.5 – Flex

hello : fait un salut

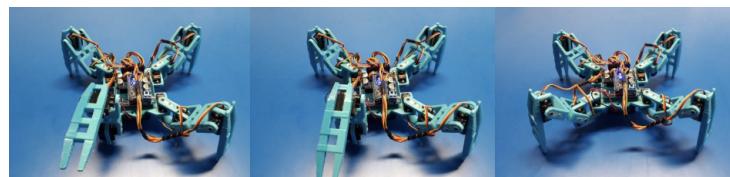


FIGURE 3.6 – Hello

applaud sym : l'araignée simule un applaudissement en levant d'abord deux pattes en diagonale et les repose puis lève et repose les deux autres

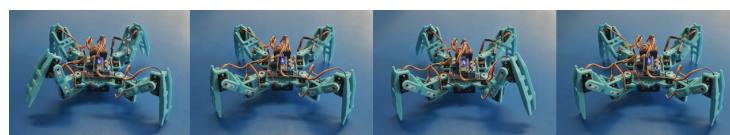


FIGURE 3.7 – Applaud sym

applaud wave : similaire au mouvement précédent mais lève une seule patte à la fois



FIGURE 3.8 – Applaud sym

applaud : tape une patte sur le sol

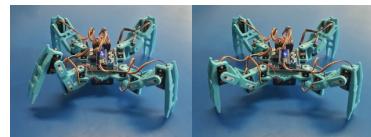


FIGURE 3.9 – Applaud

sputnik : lève les pattes en l'air



FIGURE 3.10 – Sputnik

cross : étale les pattes sur le sol (en forme de croix)

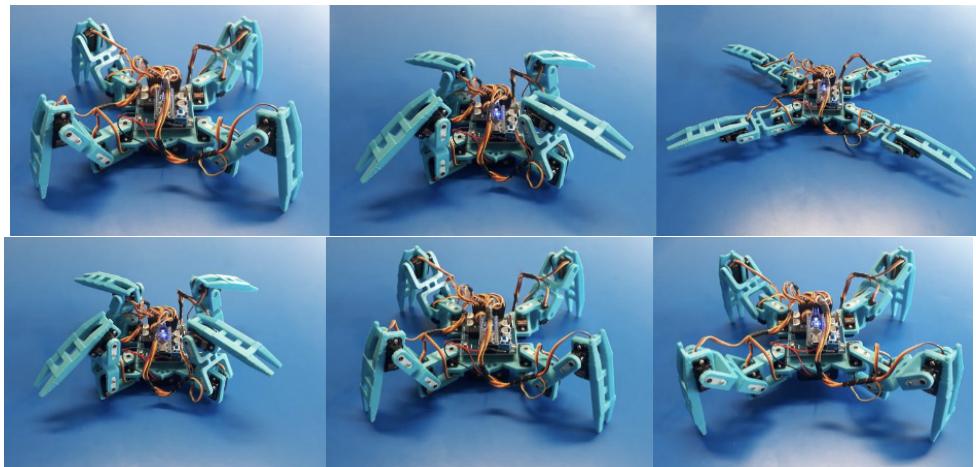


FIGURE 3.11 – Cross

bolting : inspiré de la pose d'Usain Bolt : lève et étire deux pattes en diagonale

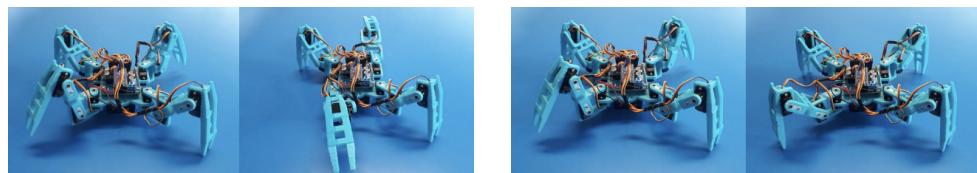


FIGURE 3.12 – Bolting

winx : simule un battement d'aile

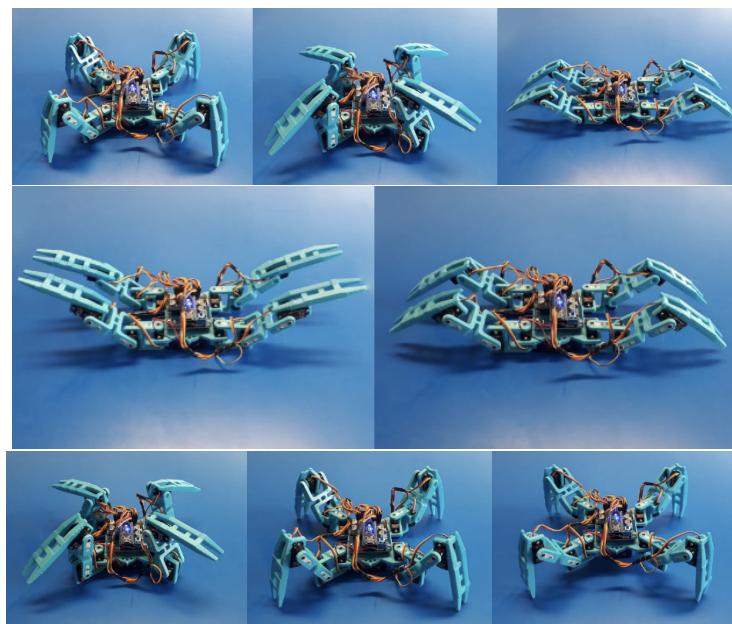


FIGURE 3.13 – Winx

pls : position par défaut, araignée écrasée



FIGURE 3.14 – PLS

3.2 Synchronisation araignées

Nous avons testé la connexion wifi entre l'agent sur l'ordinateur et la carte en utilisant le code exemple de la librarie micro ROS : https://github.com/micro-ROS/micro_ros_arduino/blob/galactic/examples/micro-ros_publisher_wifi/micro-ros_publisher_wifi.ino L'exemple initie la connexion wifi et envoie un message à la carte. Lorsque la connexion wifi échoue, la diode sur la carte se met à clignoter de manière intempestive.

Nous avons ensuite travaillé sur la synchronisation des araignées : elles sont connectés par wifi à l'ordinateur qui envoient les messages en UDP.

3.3 Construction de la coque

Pour modéliser notre coque nous avons utilisé OpenSCAD qui est un logiciel de modélisation 3D de type "constructif". Cela signifie que les modèles 3D sont créés en utilisant une syntaxe de script plutôt que par une interface graphique. OpenSCAD permet de créer des modèles 3D précis en combinant des primitives géométriques de base (par exemple, des cubes, des cylindres, des sphères) et en les modifiant à l'aide de transformations géométriques (par exemple, la rotation, la translation, l'échelle).

Nous avions plusieurs idées de personnalisation de cette coque. La première idée était de faire un mini échiquier avec des petites pièces qui viendrait s'insérer par-dessus. La deuxième idée était d'ordre plus pratique. En effet, la carte Arduino que nous utilisons a besoin d'une source d'alimentation en plus de celle déjà disponible sur l'araignée. Nous avons donc dû utiliser une batterie externe. L'idée était donc d'ajouter un socle qui pourrait tenir notre batterie sur l'araignée.

Malheureusement, par manque de temps, nous avons dû nous contenter d'une coque plus basique qui nous permet cependant de faire tenir la batterie grâce à un élastique.



FIGURE 3.15 – Modèle 3D de la coque

3.4 Limites et problèmes rencontrés

Nous avons eu des soucis de compatibilité pour l'installation de ROS :

- sur Mac, il y avait un problème de dépendance python (graphviz) bien que la dépendance était installée
- sur Ubuntu 18.04, le framework ne fonctionnait pas (problème pour afficher une fenêtre graphique)

Pourtant d'après la documentation de ROS, le framework est censé pouvoir marcher sur ces OS et être cross-platform. On notera que l'on peut également utiliser une image Docker pour ROS 2.

Nous avons également dû utiliser l'IDE Platform IO (compatible Windows, Mac et Linux), une extension de VS Code, car impossible d'utiliser l'Arduino IDE car il manquait un time.h. PlatformIO est designé pour faire marcher énormément de cartes et il faut donc spécifier la carte sur laquelle on travaille. Il se charge ensuite de récupérer les librairies. Nous avons ensuite installé l'agent Micro-ROS et testé le code de l'araignée SEALK pour tester les mouvements mais le code fourni ne marchait pas sur la carte. En effet, nous avions le problème de compilation suivant :

```
src/armcontroller.h:52:13: error: there are no arguments to 'sei'
that depend on a template parameter, so a declaration of 'sei'
must be available [-fpermissive]
sei();
```

Cette instruction permet d'activer les interruptions de timer. Nous avons donc dû utiliser notre code que nous avons fait en cours avec les TP pour les mouvements de l'araignée.

Nous avons également dû utiliser la librairie suivante pour faire marcher correctement les servomoteurs : RP2040_ISR_Servo (https://github.com/khoih-prog/RP2040_ISR_Servo). En effet, les mouvements envoient une instruction en dehors de la loop principale.

Conclusion

En conclusion, ce projet d'exploration nous a permis de découvrir de nouvelles technologies, méthodes et outils dans le domaine de l'informatique. Nous avons évalué les différentes approches et technologies disponibles, et avons choisi celles qui nous semblaient les plus adaptées à notre projet. Nous avons mis en œuvre notre solution en utilisant les technologies et les outils sélectionnés, et avons obtenu des résultats satisfaisants.

En plus de nos résultats techniques, ce projet a également été bénéfique pour notre cursus. Nous avons eu l'occasion de mettre en pratique les connaissances et les compétences que nous avons acquises au cours de nos études, et d'en apprendre de nouvelles. Nous avons également développé des compétences en matière de travail en équipe, de communication, de gestion de projet et de résolution de problèmes. Ce projet a été une expérience précieuse pour notre formation et nous espérons qu'il nous aidera à poursuivre notre carrière dans ce domaine passionnant.

Nous avons également identifié les limites de notre solution et avons défini les directions futures pour la recherche et le développement dans ce domaine. En somme, ce projet d'exploration a été une expérience enrichissante et nous espérons qu'il contribuera à l'avancement de la recherche sur Micro-ROS.

GLOSSAIRE

- Ros2 : ROS2 (Robot Operating System 2) est un système d'exploitation open source pour les robots, basé sur une architecture modulaire et un modèle de communication basé sur les services et les requêtes, qui permet de développer et d'exécuter des applications de robotique de manière flexible et scalable.
- Micro-ROS : Micro-ROS est une implémentation légère de ROS2 (Robot Operating System 2) conçue pour être exécutée sur des périphériques embarqués tels que les microcontrôleurs et les cartes de développement.
- Esp32 : ESP32 est un microcontrôleur de la famille ESP de Espressif Systems, conçu pour être utilisé dans des applications embarquées telles que les objets connectés et les capteurs.
- Arduino Nano : Arduino Nano est une carte de développement microcontrôleur basée sur un microcontrôleur Atmel AVR. Elle est conçue pour être petite et portable, avec un format de circuit imprimé compact qui peut être facilement intégré dans des projets de robotique et d'objets connectés.
- OpenSCAD : OpenSCAD est un logiciel de modélisation 3D de type "constructif", qui permet de créer des modèles 3D en utilisant une syntaxe de script plutôt que par une interface graphique.

BIBLIOGRAPHIE

Documentation de la carte Arduino Nano RP2040 : <https://docs.arduino.cc/static/56034f29d9e2bd28f4fd3c90268d0557/ABX00053-datasheet.pdf>

Tuto et kit robot araignée :
<https://learn.sunfounder.com/category/quadruped-crawling-robot-v2-0-for-arduino/>

Site officiel de Micro ROS : <https://micro.ros.org/>

Répo Github du projet Micro ROS : https://github.com/micro-ROS/micro_ros_arduino

Répo Github du composant IDF de l'ESP32 : https://github.com/micro-ROS/micro_ros_esp32_idf_component

Page Arduino de la carte Nano RP2040 : <https://docs.arduino.cc/hardware/nano-rp2040-connect>

Page Arduino de la carte Portenta H7 : <https://store.arduino.cc/products/portenta-h7>

Page du blog de micro ROS évoquant le portage du projet sur l'ESP32 : <https://micro.ros.org/blog/2020/08/27/esp32/>

Tutoriel pour mettre Micro ROS sur l'ESP32 : <https://husarnet.com/blog/esp32-microros>

Autre tutoriel pour mettre Micro ROS sur l'ESP32 : <https://www.hadabot.com/setup-esp32-to-work-with-ros2.html>

Tutoriel d'installation de ROS 2 sur Linux : <https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html>

Tutoriel pour mettre Micro ROS sur l'Arduino Nano RP2040 : <https://gist.github.com/Redstone-RM/>

Exemple code micro ROS wifi : https://github.com/micro-ROS/micro_ros_arduino/blob/galactic/examples/micro-ros_publisher_wifi/micro-ros_publisher_wifi.ino