POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF8405 - Informatique mobile

Projet Robot

Présenté à :
Alejandro Quintero

Réalisé par :

Samir Boussaadi, 1623868
Mathieu Bussières, 1882012
Ouassim Ouali, 1958275
Laurianne Guindon, 2020972

23 avril 2023

Table des matières

1. Introduction

Pour ce projet, il nous était demandé de reprendre du code laissé par d'anciens étudiants ayant travaillé sur un projet robot et de le faire fonctionner. Il y avait une application mobile permettant de contrôler un robot, à l'aide d'un gyroscope. Néanmoins, aucune documentation ne permettait de le faire. Notre mandat principal était donc de faire fonctionner ce qui était déjà fourni ou de créer une nouvelle application. Nous devions contrôler un robot avec une application mobile sur un téléphone ou une tablette et ajuster le logiciel embarqué. De plus, il a été demandé de produire une documentation. Cette documentation se retrouve partiellement dans ce rapport et dans notre entrepôt GitHub[1] avec des README. Nous avons ajouté certaines fonctionnalités au projet telles qu'un joystick et des bras ludiques qui lèvent un chapeau et une canne.

Dans ce rapport, les outils et composantes utilisés seront détaillés. Ensuite, les scénarios du robot, les accomplissements, les limitations et les recommandations seront abordés. De plus, une démonstration du robot en action sera présentée. Bien sûr, un tutoriel de démarrage est aussi fourni (section 8). Les figures sont situées en annexe.

Nous avons nommé le robot Smol. Il est donc possible que vous trouviez des références à ce nom au travers de ce rapport ou dans la documentation sur GitHub. Néanmoins, Smol représente notre équipe. Le logo que nous avons produit nous est réservé. Il y a quelques variations du logo. Vous pourrez changer les images que vous utiliserez pour votre projet. Vous les verrez dans l'application mobile et dans quelques README. L'inspiration pour le nom de Smol est décrite sur notre GitHub. Nous vous invitons à faire une fork du dépôt et d'y travailler. Si vous voulez modifier le code déjà existant sur notre dépôt, contactez-nous. Sinon, vous êtes libre d'utiliser le code produit selon la licence MIT.

Ce projet a été réalisé sur environ huit semaines de cours pour un cours de trois crédits. La répartition du cours étant de 3 - 1.5 - 3, on peut estimer à environ 4.5 heures/semaine le temps consacré à la réalisation de ce projet, et ce, en incluant le temps de révision du cours.

2. Outils / composantes requis

Pour l'organisation de notre projet nous avons choisi d'utiliser GitHub comme gestionnaire de versions, nous permettant de vérifier et mettre en commun le code ajouté par chacun des membres. Puis, nous avons utilisé Trello pour nous servir de gestionnaire de tâches (*dashboard*). Bien sûr, nous avons aussi eu recours à plusieurs autres outils et composantes, pour le côté technique / pratique du projet.

[1] https://github.com/mathblin/inf8405-projet

2.1 Kotlin

Kotlin est un langage de programmation polyvalent, open-source, orienté objet et fonctionnel qui peut être utilisé pour développer une grande variété d'applications telles que des applications Web, de bureau, mobiles, scientifiques et d'entreprise. Kotlin a été développé par JetBrains et est devenu rapidement populaire pour le développement Android depuis que Google l'a annoncé comme langage officiel de développement Android en 2017.

Les applications Android sont des applications mobiles développées pour le système d'exploitation Android, qui est utilisé sur la plupart des smartphones, tablettes et autres appareils mobiles. Les applications Android peuvent être développées en utilisant différents langages de programmation tels que Java, Kotlin et C++. Kotlin est l'un des langages les plus populaires pour le développement Android car il est plus concis, sûr et facile à utiliser que Java.

Il y a plusieurs raisons pour lesquelles nous avons choisi (Kotlin) pour développer une application de contrôle de robot SMOL par WiFi :

- Compatibilité avec Android : Kotlin et Java sont deux des langages de programmation les plus couramment utilisés pour le développement d'applications Android.
- Facilité d'utilisation : Kotlin est souvent considéré comme plus facile à lire et à écrire que Java.
- Bibliothèques disponibles : Kotlin et Java ont tous deux une grande communauté de développeurs, ce qui signifie qu'il existe de nombreuses bibliothèques et frameworks disponibles pour simplifier le développement d'applications.
- Performances : bien que Kotlin soit plus lent que Java dans certains cas, il est généralement considéré comme plus rapide que Java dans les applications Android.

En ce qui concerne le contrôle de robot par Wi-Fi, Kotlin ou Java peuvent être de bons choix car ils offrent tous deux une grande flexibilité et une large gamme de bibliothèques pour la communication réseau et de l'interaction avec les périphériques. Cependant, il est important de noter que d'autres facteurs tels que la plate-forme matérielle du robot et les exigences de performances spécifiques peuvent également influencer le choix du langage de programmation.

2.2 Android Studio

Android Studio est un environnement de développement intégré (IDE) pour le développement d'applications mobiles Android. Il est développé par Google et est devenu l'IDE officiel pour le développement d'applications Android.
Android Studio est basé sur IntelliJ IDEA, un IDE Java populaire. Il est conçu pour faciliter le développement d'applications Android en offrant des fonctionnalités telles que l'éditeur de code, le débogage, la gestion de version, la compilation et le déploiement d'applications.

Les raison pour lesquelles nous avons choisi Android studio sont :

- Un éditeur de code intelligent avec une mise en surbrillance de syntaxe, une complétion de code, une refactorisation et une analyse de code en temps réel.
- Un débogueur puissant avec des fonctionnalités telles que la surveillance de variables, la mise en attente du code, la gestion des points d'arrêt et la visualisation des piles d'appels.
- Une interface de conception graphique pour la création d'interfaces utilisateur avec des outils de glisser-déposer et de prévisualisation en temps réel.

2.3 Arduino IDE

Le contrôle des moteurs et par extension celui des roues est fait par un Arduino. Le détail de la carte est dans la section sur les outils matériels. Le langage ou programme utilisé est par conséquent le Arduino Sketch qui est fortement inspiré par C++ (Arduino, 2023).

Le Sketch fourni avec le projet est repris de projets robots d'INF8405 précédents. Néanmoins, nous l'avons modifié. Par exemple, le projet sur lequel nous nous sommes inspirés avait déjà un contrôle des roues fait pour un mode gyroscope. Nous avons ajouté un mode pour contrôle d'angles et de puissance compatible avec un joystick.

Algorithme du gyroscope sur Arduino

Cet algorithme existait déjà lorsque nous avons repris le code initial. Nous l'avons simplement mis dans un fichier c++ afin de séparer les modes que recevait l'Arduino. Ces modes sont le gyroscope, le joystick et le servomoteur. Les commandes du mode gyroscope sont des caractères parmis les suivants: `WwSsadqQeEzZcCx`. Lorsque le robot reçoit un de ces caractères, il bouge d'une manière différente. Par exemple, `w` minuscule le fera avancer lentement et `W` majuscule le fera avancer plus rapidement, alors que `s` minuscule le fera reculer lentement et `S` majuscule le fera reculer plus rapidement. Par intuition, on peut se fier aux touches « asdw » du clavier pour connaître les directions, puisqu'elles représentent les flèches gauche, bas, droite et haut respectivement.

Cet algorithme est celui qui fonctionne le moins bien avec notre projet. La description de son comportement faite ci-dessus découle d'une analyse que nous avons faite du projet Arduino de base pour le gyroscope. Nous ne pouvons pas comparer avec ce qui a été fait avant puisque nous n'avons ni vidéo ni modèle de validation de cet algorithme.

Algorithme de joystick sur Arduino

Cet algorithme est décrit dans la section des accomplissements.

2.4 L'Arduino

Le modèle Arduino utilisé pour le projet robot est le DFRduino RoMeo V1.0. Dans le gestionnaire de cartes (Boards Manager), il s'agit du Atmel atmega328p Xplained mini (voir figure 2 en annexe). Rappelons que lors du projet intégrateur 1, nous utilisions un microcontroller d'Atmel, soit le ATmega324PA. Nous étions donc en territoire connu.

2.5 Bras, canne et chapeau

Pour la modélisation 3D des bras, de la canne et du chapeau, nous avons utilisé deux logiciels : Tinkercad et FreeCAD. Pour les ajustements avant impression, sur imprimante 3D, nous avons utilisé UltiMaker Cura. Ensuite, pour rendre ces membres mobiles, nous avons utilisé deux servomoteurs GS-9018 connectés à la carte Arduino (figure 8).

2.6 Wifi Local

Pour la carte Wifi utilisée, nous avons pensé à utiliser la carte WizFit210, mais dû à des limitations discutées dans la section accomplissement, nous avons dû la changer. La carte utilisée est de type ESP32, plus précisément une ESP32-D0WDQ6. La présence de deux cœurs de processeur permettent aussi la possibilité de gérer deux évènements/opérations en parallèle.

2.7 Écran LCD et autres ajouts

Pour faciliter le débogage et mieux comprendre les évènements se déroulant sur le robot, un écran LCD de 5 pouces a été ajouté au Raspberry PI. L'écran permet d'avoir un feedback visuel sans avoir à connecter le robot à un écran HDMI. Nous avons aussi ajouté une unité de refroidissement pour éviter le ralentissement inutile de l'application et aussi éviter d'endommager le Raspberry PI.

2.8 Tablette

Nous avons utilisé la tablette fournie par école pour installer notre application qui contrôle notre robot SMOL.
C'est une tablette Android Samsung Galaxy Tab A 10.1 de taille moyenne avec un écran tactile de 10,1 pouces, voici les principales caractéristiques de cette tablette :

- Écran : l'écran LCD de 10,1 pouces a une résolution de 1920 x 1200 pixels.
- Système d'exploitation : cette tablette utilise le système d'exploitation Android 9.0 (Pie).
- Processeur : cette tablette est équipée d'un processeur Exynos 7904 octa-core cadencé à 1,8 GHz.
- Mémoire vive : cette tablette dispose de 2 Go de RAM.

3. Scénarios

3.1 Perte de connexion

Pour la perte de connexion, un scénario est utilisé selon les différents environnements, soit sur l'application Android, sur le serveur Python et sur l'Arduino. Ces scénarios de perte de connexion ont été instaurés dans une optique d'assurance qualité.

3.1.1 Application Android

En cas de perte de connexion entre votre tablette et votre robot SMOL, votre application affichera un message d'erreur indiquant la perte de connexion. La caméra en direct de votre robot SMOL ne sera plus visible sur la troisième page de l'application.
Les commandes socket ne seront plus envoyées au robot SMOL et ce dernier ne répondra plus aux commandes et ne fera rien jusqu'à ce que la connexion soit rétablie. Pour résoudre le problème, vous devez d'abord vérifier votre connexion Wi-Fi pour vous assurer que votre tablette est toujours connectée au réseau Wi-Fi sur lequel est connecté le serveur. Si la connexion Wi-Fi est toujours activée, vérifiez les paramètres d'adresse IP que vous avez entrés pour vous assurer qu'elle est correcte. Si le problème persiste après avoir vérifié votre connexion Wi-Fi et les paramètres d'adresse IP, vous pouvez redémarrer l'application pour réinitialiser la connexion et recommencer le contrôle de votre robot.

3.1.2 Le serveur Python

Le serveur instancie un socket pour recevoir les demandes d'interaction avec l'application mobile. Si une tablette ou un téléphone tente de se connecter, le serveur recevra la demande et établira une connexion unique avec le premier dispositif qui s'y connecte. Il peut y avoir une perte de connexion avec ce socket. Le client (application Android) peut planter, se déconnecter, fermer l'application, etc. Lorsqu'un tel événement se produit, le serveur interrompt son socket et se met en attente d'une nouvelle connexion. Dans une telle situation, il enverra également la commande « stop » au Arduino afin que ce dernier arrête tous ses mouvements.

3.1.3 Arduino

Tel qu'indiqué à la section précédente, l'Arduino recevra la commande « stop » du serveur. Cette commande arrêtera les moteurs ainsi que le servomoteur pour le bras mécanique tenant une canne.

4. Accomplissements

4.1 Algorithme de joystick sur Arduino[2]

Nous avons implémenté nous-même l'algorithme du joystick, sans s'inspirer de travaux précédents puisque nos recherches là-dessus ont été infructueuses. La figure 3 démontre le comportement général qu'adopte le robot lorsque nous utilisons cet algorithme. L'algorithme reçoit un angle et une puissance. L'angle reflète la direction que prendra le robot. La puissance influence la vitesse des roues. Le cercle bleu représente le cercle trigonométrique. Les quatre cadrans sont séparés par une ligne horizontale et une ligne verticale, les deux en pointillés. Les cadrans mis en évidence par un chiffre bleu. Lorsque l'angle est dans le premier cadran, le robot va vers l'avant-droite puisque les roues gauches ont plus de puissance que les roues droites. Dans le deuxième cadran, le robot va vers l'avant-gauche. Puis, vers l'arrière-gauche et vers l'arrière-droite pour les cadrans 3 et 4 respectivement. L'angle et la puissance déterminent la différence de puissance PWM (pulse width modulation) ou vitesse entre les roues.

Donc, selon le cadran, les roues d'un côté sont toujours plus puissantes que celles de l'autre. Par exemple, toujours en se fiant à la figure 3, dans les 1er et 4^{e} cadrans, les roues gauches sont plus puissantes que celles de droite (flèches vertes plus longues), mais en sens inverse entre ces deux cadrans (flèches opposées). Dans les 2^{e} et 3^{e} cadrans, les roues droites sont plus puissantes que celles de gauche. Ceci permet une transition plus douce, soit un tournant progressif. La puissance reçue est considérée pour faire avancer les deux roues à une plus ou moins grande vitesse.

On remarque également deux lignes rouges pointillées. Elles séparent une zone différente dans les cadrans. Nous avons décidé de faire tourner le robot sur lui-même dans ces deux zones de 30^{o}. Le robot tourne sur lui-même à gauche lorsqu'il est entre 165^{o} et 195^{o} et sur lui-même à droite lorsqu'il est entre 345^{o} et 15^{o}. En d'autres termes, il effectue un virage de 360^{o} dans un sens ou dans l'autre.

Lorsque l'angle est de 90^{o} ou de 270^{o}, les roues ont la même puissance pour la droite et la gauche. Le robot roule vers l'avant pour un angle de 90^{o} et vers l'arrière pour un angle de 270^{o}. Il est important de considérer que les résultats réels peuvent varier de ce qui est décrit ici. Le matériel peut faire défaut et changer le comportement désiré. Les causes peuvent être l'usure, un bris matériel, une erreur de fabrication des pièces, une mauvaise connexion physique, etc.

[2] Il est à noter que cet algorithme est encore en phase expérimentale. Il n'est donc pas parfait.

4.2 Développement de l'application android

Notre application se compose de trois activités , alors cela signifie qu'il y a trois fichiers Java distincts qui étendent la classe AppCompatActivity et chacun d'entre eux représente une interface utilisateur différente de l'application.

L'activité "MainActivity" de ce code est destinée à afficher une interface utilisateur pour se connecter à un notre robot SMOL et à le calibrer. Le code commence par importer diverses classes Android, telles que "Intent", "ActionBar", "TextView", "EditText", "Button", etc. pour créer l'interface utilisateur.

Dans la méthode "onCreate", le code défini le contenu de l'activité à partir du fichier de mise en page "activity_main.xml". Il masque la barre d'action, définit un texte et un style personnalisé pour un TextView, récupère le texte entré dans un EditText et définit un OnClickListener pour un bouton.

Lorsque l'utilisateur clique sur le bouton "calibrateButton", une nouvelle activité appelée "CalibrationActivity" est lancée à l'aide d'un Intent. Le texte entré dans l'EditText est transmis en tant que paramètre à l'activité "CalibrationActivity" en utilisant la méthode "putExtra" de l'objet Intent.

En fin de compte, l'activité "MainActivity" fournit une interface utilisateur simple pour se connecter et passer au calibrage de robot.
En général, chaque activité de l'application est conçue pour fournir une fonctionnalité spécifique et possède son propre ensemble de méthodes, de vues et d'éléments de mise en page. Les activités communiquent entre elles en utilisant des Intent pour passer des données d'une activité à l'autre et lancer des activités supplémentaires

L'activité "CalibrationActivity" nous avons commencé par la déclaration des variables :
sensorManager: une instance de la classe SensorManager, qui est utilisée pour accéder aux capteurs du système.
sensorAccelerometer: une instance de la classe Sensor qui représente l'accéléromètre du dispositif mobile.
firstTime : une variable booléenne qui indique si c'est la première fois que l'on mesure une accélération.
isChecked: une variable booléenne pour vérifier le mode d'utilisation de l'application.
posXYZ: un tableau de 3 éléments pour stocker la position initiale du dispositif.
linear_acceleration: un tableau de 3 éléments pour stocker les accélérations linéaires.
relative_linear_acceleration: un tableau de 3 éléments pour stocker les accélérations linéaires relatives.
buttonMin, buttonMax, buttonStart, buttonReset: les boutons pour les actions de calibration.
max_acceleration, min_acceleration: les valeurs maximale et minimale pour les accélérations sur l'axe x.

textMin, textMax, textPosition: les TextViews qui affichent les valeurs mesurées.

Le code utilise les événements des boutons pour mettre à jour les valeurs min et max des accélérations et les valeurs de positions sur les axes x, y et z. Le code est également équipé d'un bouton de réinitialisation pour annuler toutes les valeurs de calibration et recommencer le processus.
Le code utilise également un Switch pour permettre à l'utilisateur de choisir entre le mode JOYSTICK ou le mode GYROSCOPE. Les valeurs de cet interrupteur sont transmises à l'activité vidéo de l'application Android qui utilise ces informations pour adapter la façon dont elle est utilisée.

Le code utilise l'accéléromètre du dispositif pour mesurer les accélérations linéaires en temps réel et stocke ces valeurs pour les utiliser dans l'activité vidéo de l'application Android. Enfin, le code redirige l'utilisateur vers l'activité vidéo après la calibration est terminée.
L'activité "VideoActivity" est l'activité principale qui contrôle le robot en utilisant soit le joystick soit le gyroscope de la tablette, et qui affiche le flux vidéo de la caméra du robot sur l'écran de la tablette . Nous avons commencé par définir une constante JOYSTICK_SCALE et une variable TAG. Ensuite, il initialise un VideoView, qui permet de lire des vidéos et un URL qui correspond à l'adresse du serveur qui héberge le système de contrôle à distance (le raspberry pi). Les variables mode, SERVERPORT et SERVER_IP sont également définies. ensuite nous avons e initialise un SensorManager, un Sensor de type accéléromètre et quelques autres variables qui stockent les valeurs d'accélération maximale et minimale, les valeurs de l'axe x (posXYZ), la variable isChecked (qui indique si le mode joystick ou gyroscope est sélectionné) et la variable switchMode (qui indique si le mode joystick est activé ou non).Le code utilise également un BroadcastReceiver pour détecter les changements de connectivité réseau et affiche un message de perte de connexion si nécessaire.
Le code initialise un thread client, un thread qui utilise une connexion TCP pour communiquer avec le serveur distant. Il initialise également un joystick pour le contrôle manuel du robot, et définit les propriétés de la vue, notamment sa taille et sa position. Le code initialise également une Switch pour passer du mode joystick au mode gyroscope, et un bouton Hello.
Ensuite, le code vérifie si le mode joystick est activé ou non et en fonction de cela, il affiche ou masque le joystick. Si le mode joystick est activé, il définit le texte de switchMode en "Mode JOYSTICK start" et s'il est désactivé, il le définit en "Mode GYROSCOPE start".
Enfin c'est l'implémentation d'un listener pour le capteur d'accéléromètre sur un appareil Android. L'objectif est d'utiliser les données de l'accéléromètre pour contrôler un robot à distance.
Dans la méthode onSensorChanged, les données d'accélération brutes sont récupérées à partir de l'événement du capteur. Ces données sont stockées dans un tableau linear_acceleration.
Ensuite, le code effectue un certain nombre de calculs pour déterminer la direction et l'intensité du mouvement du téléphone, puis envoie des commandes au robot distant en fonction de ces calculs. Par exemple, si le téléphone est incliné vers la droite, le robot est commandé pour tourner à droite. Les commandes sont envoyées à travers un thread de client en utilisant la méthode clientThread.sendMessage().

4.3 Bras avec canne et chapeau

Nous avons rajouté des bras pour rendre notre robot plus amical. Pour ce faire, nous avons utilisé deux servomoteurs. Les détails sur les étapes de configuration sont présentés plus bas, à la section 8.2 et les bras sont illustrés à la figure 9. Le bras positionné à la gauche du robot (lorsqu'on est à l'arrière de celui-ci) est celui qui tient une canne. Ce bras bouge dès que le robot est en mouvement, puis s'arrête dès que le robot s'immobilise. Il fait bouger la canne de haut en bas avec un angle de 45°. Lorsque le bras est arrêté (fonction *stop*), il se repositionne à l'angle de départ (spécifiée dans *init*). Le second bras, le bras de droite, tient le chapeau. Pour sa part, il effectue un mouvement de haut en bas, suivant une rotation de 90°, puis de bas en haut pour revenir à sa position initiale. Ce mouvement est activé à l'aide de l'application mobile. Sur la même page où l'on peut contrôler le joystick ou le gyroscope, on peut trouver le bouton *DIT BONJOUR* à droite de l'écran. C'est avec ce bouton qu'il est possible d'activer le bras et son chapeau.

4.4 Réseau Wifi privé

Au cours de la phase de planification du projet, nous avons envisagé d'assurer les communications en utilisant un réseau WiFi local. L'objectif était de permettre l'utilisation de l'application et du drone sur de plus grandes distances, quel que soit l'endroit. Pour ce faire, nous avons utilisé un module WiFi initial composé d'un Arduino Uno sur lequel était placée une carte WizFit210. Cette carte agissait comme un serveur DHCP, ce qui nous permettait de nous y connecter et de tester la connexion en utilisant le ping. Cependant, nous avons constaté que cette carte présentait des limitations, notamment l'absence de support pour les fonctionnalités plus récentes et l'impossibilité de communiquer facilement entre deux systèmes via ce point d'accès. Pour pallier ce problème, nous avons utilisé une carte ESP32, qui est plus récente et bénéficie d'une documentation et de projets abondants. La carte est programmée via le logiciel Arduino IDE. Le code écrit permet de créer un point d’accès Wifi sur lequel tous les appareils peuvent être connectés et communiquer entre eux. La limitation actuelle de cette configuration est que le point d’accès n’est actuellement pas connecté à internet et agit réellement plus comme un réseau local. Une possibilité est de l'utiliser comme un relais Wifi et donc de garder la possibilité d’être connecter à internet pour par exemple faire la diffusion en direct de la vidéo en temps réel.

5. Limitations et recommandations

5.1 Algorithme de joystick sur Arduino

Pendant la phase de développement, l’algorithme ne permettait pas de tourner de manière assez prononcée. En d’autres termes, il devait parcourir une certaine distance afin de faire un léger virage. À ce moment-là, la seule astuce à ce problème était de le faire s’arrêter, tourner à 360^{o} dans la direction souhaitée, puis de le faire avancer en ligne droite. Il a été possible de pallier ce problème en diminuant la puissance de la roue du côté où le robot tourne.

Néanmoins, elle doit conserver un minimum de puissance. Des travaux futurs pourraient être investis dans la résolution de ce problème.

Nous avons depuis résolu le problème en multipliant plusieurs fois la vitesse de la roue faible par le sinus de l'angle calculé. Ainsi, sa puissance est proportionnelle à l'angle, mais elle est également plus faible lorsque nous diminuons l'angle. Il faut considérer que 30º autour de 0º et autour de 180º sont réservés pour que le robot tourne sur lui-même (voir figure 3)[3]. Conséquemment, plus l'angle est faible (d'un côté comme de l'autre), moins le sinus permettra de tourner de manière prononcée puisque les valeurs proches de 0º et de 180º (0º à 15º, 345º à 360º et 165º à 195º) sont réservées pour les virages à 360º.

Une recommandation serait de changer le UI de l'application mobile afin que les zones de virage à 360º apparaissent sur le joystick délimitées par les lignes pointillées rouges de la figure 3. Ceci permettrait de savoir lorsqu'un utilisateur parvient à ces zones avec son doigt.

5.2 Suggestion

Pour pouvoir continuer à contrôler votre robot Smol lorsque vous perdez la connexion WiFi avec votre tablette Android, vous devez implémenter un mécanisme de commutation automatique vers une autre connexion disponible, telle que Bluetooth.

Pour ce faire, vous devez créer un système de gestion de connectivité qui surveille constamment les connexions disponibles (WiFi, Bluetooth, etc.) et qui peut basculer automatiquement vers une connexion alternative en cas de déconnexion. Vous pouvez également configurer le système pour vous avertir en cas de perte de connexion WiFi et vous donner la possibilité de basculer manuellement vers une autre connexion.

Voici les étapes générales pour implémenter un tel système :

- Déterminez quelles connexions alternatives sont disponibles pour votre robot Smol, telles que Bluetooth, NFC ou une connexion filaire. Vous devez vous assurer que le robot Smol est équipé de la technologie appropriée pour établir une connexion alternative.

- Ajoutez une logique de détection de connectivité à votre application Android existante pour surveiller la perte de connexion WiFi. Cette logique peut être implémentée en utilisant des BroadcastReceiver pour détecter les changements d'état de la connectivité.

- Lorsqu'une perte de connexion WiFi est détectée, votre application doit alors tenter de se connecter automatiquement à une connexion alternative disponible. Pour ce faire,

[3] Les angles de la figure 3 peuvent ne pas être proportionnels à la réalité, il ne s'agit que d'une esquisse pour permettre de visualiser le comportement défini par l'algorithme.

vous devez inclure une logique de connexion à chaque type de connexion alternative que vous avez identifié à l'étape 1.

- Lorsque vous avez réussi à établir une nouvelle connexion, vous pouvez reprendre le contrôle de votre robot Smol et continuer à lui envoyer des commandes via la nouvelle connexion.

- Si vous ne parvenez pas à établir une nouvelle connexion automatiquement, vous pouvez avertir l'utilisateur et lui donner la possibilité de basculer manuellement vers une connexion alternative.

En suivant ces étapes, vous pouvez mettre en place un système de commutation automatique de la connectivité qui vous permettra de continuer à contrôler votre robot Smol, même si vous perdez la connexion WiFi.

5.4 Communication bloquante

La communication avec l'Arduino est bloquante, c'est-à-dire que pour chaque commande envoyée, le serveur Python attend de recevoir la réponse du Arduino avant de poursuivre son exécution. Il y a deux raisons à cet aspect bloquant.

La première raison est que l'Arduino avait un comportement étrange lorsqu'il recevait des commandes. Par exemple, une commande « avancer » était reçue par l'Arduino longtemps après l'avoir envoyée au serveur par l'application mobile. Attention, ici, il ne s'agit pas du délai de la communication bloquante, mais plutôt d'un délai de traitement par l'Arduino sur les commandes reçues. Ce délai était pire lorsque nous utilisions longtemps l'application. Il est possible que la communication sérielle ne permettait pas de bien séparer les envois. Néanmoins, un délai est également inséré par la communication bloquante afin d'attendre une réponse du Arduino. Ceci dit, les commandes envoyées étaient plus pertinentes. Cet aspect bloquant entraîna un nouveau défi, soit de filtrer le flux de commandes puisque l'application mobile envoie les commandes plus rapidement par socket que la communication sérielle. On peut voir sur la figure 5 que pour chaque commande envoyée (variable *command*), plusieurs commandes ont été reçues (variable *recvCommand*). Au-dessus de la ligne rouge, ce sont les commandes en mode gyroscope. En dessous, nous pouvons lire les commandes en mode joystick.
Dans les deux cas, trop de commandes sont reçues. Nous les filtrons en prenant la dernière commande envoyée par l'application mobile.

La deuxième raison est que nous voulions pouvoir déboguer le code qui s'exécute sur l'Arduino. Étant donné que les systèmes embarqués sont plus difficiles à déboguer, nous avons eu recours à la communication sérielle pour connaître l'état de certaines variables. La variable *Arduino's answer* nous permettait d'accéder à ces informations.

La recommandation que nous voulons faire concernant cette communication bloquante est d'utiliser une machine à états au lieu d'attendre constamment que le lien sériel soit prêt à envoyer et recevoir des commandes. Tel qu'il est, la communication opère avec un protocole de poignées de main. Une machine à état pourrait permettre d'éviter de bloquer le serveur et l'Arduino. Il pourrait y en avoir une pour le serveur et une pour l'Arduino. Cependant, il s'agit d'une suggestion et non d'une solution absolue. Il est libre à vous d'expérimenter.

6. Démonstration

Vidéo à venir…

7. Conclusion

Pour finir, ce projet complexe nous a permis de nous familiariser avec un code qui n'était pas le notre pour en tirer une base sur laquelle nous avons pu faire des optimisations et ajouter de nouvelles fonctionnalités. Il a également été nécessaire de se familiariser avec les nombreux outils et composantes que nous voulions utiliser. Malgré tout, nous sommes parvenus à produire une application intégrant les modes joystick (nouveau) et gyroscope pour manœuvrer notre petit robot Smol auquel nous avons rajouté des bras actifs. De plus, nous avons rajouté de la documentation sur GitHub afin de vous guider dans votre compréhension et utilisation de cette application.

Nous avons eu beaucoup de plaisir à travailler sur ce projet et nous espérons qu'il en sera de même pour les équipes à venir.

N'oubliez pas de continuer la lecture de ce rapport. Vous verrez un tutoriel pour vous aider à faire l'utilisation de cette application. Certaines notions sont répétées dans les *README.md* du répertoire *Git*[4]. Vous aurez quelques images pour appuyer ce tutoriel. Elles se retrouvent dans l'annexe.

[4] https://github.com/mathblin/inf8405-projet

8. Tutoriel

Dans cette partie, un tutoriel vous est présenté afin de vous guider dans la préparation et l'utilisation de votre projet du contrôleur de robot. Nous supposons que vous êtes déjà à l'aise avec certains outils comme Python 3 (Windows et Linux) et Arduino. Conséquemment, aucune explication ne sera donnée sur ces outils. Il est à noter que les images utilisées directement dans cette section proviennent de nos différents fichiers *README.md* de notre répertoire Git.

Nous vous montrerons comment fonctionne l'application mobile, le serveur Python sur Raspberry Pi et l'Arduino. Il est important de bien suivre les étapes et recommandations fournies. Certaines librairies devront être installées au cours de ce tutoriel, entre autres avec Python. Il se peut que vous deviez faire des ajustements selon votre environnement de travail (système d'exploitation, version du SE, etc.).

8.1 Application mobile

Voici un tutoriel étape par étape pour utiliser notre application Android :

- Télécharger l'application : https://github.com/mathblin/inf8405-projet/tree/main/MyApplicationINF8405.
- Installer et ouvrir l'application sur votre smartphone (Android) ou votre tablette Android.

La première page :

- Il s'agit de la page d'accueil de SMOL (Figure 10). Afin de lancer l'application, vous devez cliquer sur le bouton "Start". Vous serez redirigé vers la page de sélection de l'adresse IP et de calibrage.

La deuxième page :

- Appuyez sur le bouton « Calibrage » de la deuxième page de l'application pour calibrer le gyroscope de votre appareil (Figure 11).
- Choisissez l'adresse IP. Pour ce faire, vous pouvez prendre celles déjà disponibles ou en ajouter une. Pour en ajouter une, vous devrez ajouter une étiquette et une adresse ip, puis cliquer sur le bouton « Ajouter ». Afin d'utiliser l'adresse IP nouvellement ajoutée, vous devrez la sélectionner dans la liste d'adresse IP. Elle est située en haut des champs d'étiquette et d'adresse IP. Sur la figure 11, il s'agit de l'endroit où il y a un nom (ex: Poly).
- Par défaut, le mode choisi sera gyroscope.
- Cliquer sur « START ».
- Lorsque vous êtes en mode gyroscope (Figure 12), vous pouvez contrôler votre robot en inclinant votre appareil dans la direction que vous voulez aller. Si vous êtes dans le

mode joystick, vous verrez un joystick à l'écran (la page suivante) que vous pourrez utiliser pour contrôler votre robot (Figure 13).

La troisième page :

Si vous avez bien lancé le script Python de la caméra dans votre serveur (ex: sur Raspberry Pi), vous verrez un flux de caméra en direct de votre robot. Appuyez sur le bouton "Switch" pour basculer entre le mode gyroscope ou joystick.

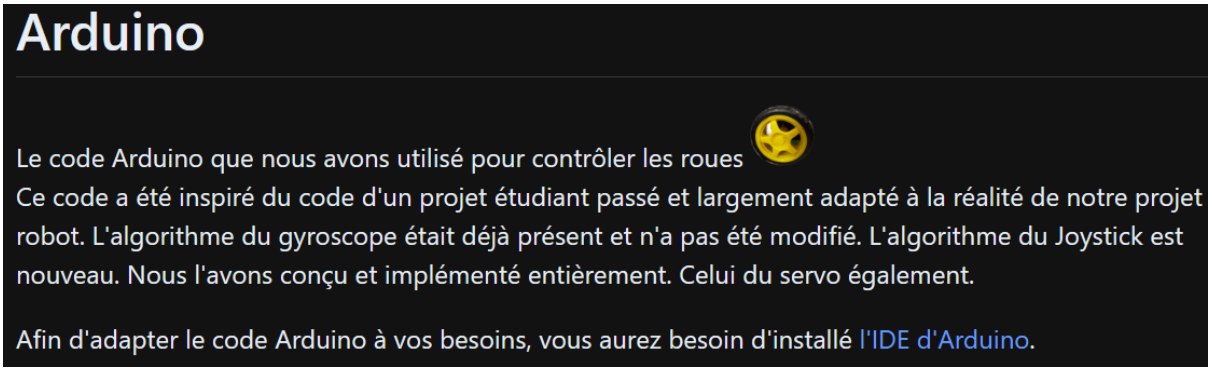
Utilisez votre appareil pour contrôler le robot. Dans le mode gyroscope, inclinez votre appareil dans la direction que vous voulez aller. Dans le mode joystick, utilisez le joystick à l'écran pour contrôler votre robot.

Et voilà ! Vous avez maintenant toutes les informations nécessaires pour utiliser notre application Android pour contrôler SMOL.

8.2 Arduino

Si vous utilisez l'Arduino de la figure 1 (voir annexe), vous devrez installer les gestionnaire de carte de la figure 2 comme sur la figure 4 dans l'IDE d'Arduino. Vous devrez ensuite brancher l'Arduino et charger le programme (upload) sur la carte par l'IDE.

Le code actuel est fait pour contrôler les roues et les deux bras que nous avons ajoutés. Il est possible que vous deviez l'adapter pour vos besoins.



Pour notre code Arduino qui utilise des threads, il est aussi important de télécharger la librairie *protothreads.h* dans le gestionnaire de librairie de l'IDE d'Arduino pour être en mesure de lancer notre code sans erreur.

Les bras sont fixés aux servomoteurs à l'aide de fils et des hélices des servomoteurs (figure 9). Les servomoteurs doivent être connectés aux pins 8 et 9. La pin 8 est pour le bras de gauche, celui avec la canne, puis la pin 9 est pour le bras de droite, avec le chapeau. Si l'orientation des bras n'est pas adéquate pour vous, lors du lancement du code, il est possible

de les réorienter simplement en dévissant la vis de l'hélice, puis en la repositionnant avec la bonne orientation. Dans le code, sous Smol/Arduino/Smol_arduino/, on retrouve le fichier Smol_arduino.ino dans lequel se trouve l'initialisation des pins (fonction *init*). Si vous voulez changer les pins, libre à vous de le faire, en respectant les connexions. La variable correspondante au servomoteur avec le bras et la canne se nomme *cane*, puis celle pour le bras avec le chapeau se nomme *hat*. Lorsque le robot est en mouvement, la canne s'active. Elle fait un mouvement de haut en bas avec une rotation de 45°. Le mouvement de haut en bas et inversement dure une seconde. Pour le chapeau, le mouvement de haut en bas (et inversement) dure trois secondes. Le chapeau suit une rotation de 90°. Pour les deux bras, l'angle peut être changé lors de l'initialisation des pins (fonction *init*) et la durée du mouvement grâce à la fonction *start*.

L'Arduino lance des fils d'exécution dès le départ afin de permettre aux fonctionnalités de déplacement des roues et des bras d'opérer quasi simultanément. Sur la figure 8, vous pouvez voir que les fils d'exécution sont initialisés puis ordonnancés par les fonctions *setup* et *loop* respectivement.

L'Arduino et le serveur communiquent selon un protocole de poignées de main. Après avoir lancé ses fils d'exécution, celui responsable à la fois de la communication et de faire avancer les roues se met en mode attente. Sur la figure 6, nous pouvons voir que le fils d'exécution bloque en attendant de recevoir une communication sérielle de la part du serveur Python. Tel que mentionné dans la section sur les limitations et recommandations, cet aspect bloquant sert au débogage et à la validité des commandes.

Vous pouvez changer les informations que l'Arduino envoie au serveur lorsqu'il est en mode débogage. Nous pouvons voir un exemple d'envoi au serveur sur la figure 7. En mode joystick et débogage, le système embarqué envoie des informations sur le cosinus et le sinus de l'angle reçu du serveur, ainsi que sur l'état de d'autres variables. Vous pouvez ainsi vérifier les états des variables sur le système embarqué. Si la variable *debug* est nulle, l'Arduino envoie une simple *ok*.

8.3 Serveur Python sur Raspberry Pi

Quelle est son utilité?

L'objectif du serveur est de recevoir les commandes envoyées depuis l'application mobile via des sockets et les transmettre au Arduino via une communication sérielle. L'Arduino exécutera ensuite ces commandes. Il peut s'agir de commandes qui le font avancer, reculer, faire une rotation de 360^{o}, etc.

Le fichier d'entrée du serveur

Le fichier d'entrée que vous devez exécuter est `server.py`. Si vous désirez connaître les drapeaux/arguments (flags) possibles à passer au serveur, vous pouvez exécuter `python server.py -h` sur Windows ou `python3 server.py -h` sur linux. Vous pouvez également utiliser `--help`, ce qui revient au même.

Drapeaux/arguments

Les objectifs principaux des drapeaux sont changer l'adresse IP du serveur pour qu'elle corresponde à celle de l'ordinateur (portable, tour, Raspberry PI), déboguer avec des affichages de console, tester la communication sérielle, etc. Si vous avez toujours des doutes sur comment fonctionnent les drapeaux/arguments, suivez la section `Le fichier d'entrée du serveur`.

Plusieurs modes

Il est possible de tester la communication sérielle en ajoutant le drapeau `-serial` lorsque vous appellez le serveur. De cette manière, vous pouvez tester de nouvelles commandes Arduino. Conséquemment, vous devrez ajuster le code d'Arduino. Sans le drapeau `-serial`, le mode est avec socket. Dans ce mode, vous aurez besoin de l'application mobile installée sur un téléphone ou sur une tablette afin que le serveur reçoive des commandes.

L'adresse IP

L'adresse IP doit être configurée dans l'application mobile. Elle doit corresponde avec celle du serveur. Pour la partie serveur, you devrez soit changer l'adresse IP par défaut dans le code, soit utiliser le drapeau `-ip` suivi de l'adresse. Par exemple, en faisant ceci: `python server.py -ip 127.0.0.1`. Vous pouvez chercher la variable suivante dans le code:

```
DEFAULT_HOST
```

Si elle n'a pas été déplacée depuis, elle devrait se trouver dans le fichier `server_flags.py`. L'adresse par défaut que nous avons utilisée est celle que `Smol` avait à l'université Polytechnique Montréal. Il est fortement possible qu'elle ait changée lorsque vous lirez ceci. Vous aurez à communiquer avec le département informatique si vous y êtes étudiant(e)s.

Caméra

Le script `camera.py` est repris du code déjà fourni par le git de Bilal. Il permet d'activer la caméra du Raspberry Pi. Ensuite, vous pourrez vous connecter par l'application mobile. Nous avons utilisé le port `8000`.

Librairies utilisées

Un fichier de requis pour le serveur est fourni avec le dépôt. Toutes les librairies utilisées y sont présentes. Pour le moment, seul `pyserial` n'est pas une librairie intégrée dans Python. Nous l'utilisons pour le serveur. Vous pouvez l'installer avec le fichier de requis nommé `requirements.txt`. Il est dans le dossier `Server`. Vous pouvez utiliser la commande `pip install -r requirements.txt`. La librairie `socket` est également utilisée. Néanmoins, elle semble être intégrée dans Python, du moins pour certaines versions de Python. Selon la documentation de la librairie socket, vous devez avoir au moins la version 2.7 pour Python 2 ou 3.5 pour Python 3. Attention, il s'agit de `socket` sans `s`, donc pas `sockets`. `socket` est une interface réseau de bas niveau.

Les librairies précédentes doivent être installées pour le bon fonctionnement du serveur[5].

Tel que mentionné dans la section Arduino de ce tutoriel, l'Arduino envoie des informations lorsqu'il est en mode débogage et un simple *ok* sinon. Il est intéressant de constater que ce mode est indiqué par la variable *debug* dans le micrologiciel. Cette variable est en fait envoyée par le serveur Python. Dans la figure 6, vous pouvez constater que l'Arduino reçoit un message (*msg*) par communication sérielle. De ce message, il extrait la valeur de la variable *debug*. Le drapeau *-debug* active ce mode lors du lancement du serveur.

8.4 Téléchargement du dépôt Git sur un Raspberry Pi

Nous ne couvrons qu'un cas spécial ici, soit le téléchargement du dépôt git sur un Raspberry Pi. Nous supposons que vous savez déjà vous servir d'un dépôt git. Dans le cadre du projet, nous n'étions pas en mesure de télécharger le dépôt avec un identifiant git normal. Nous avons donc créé un courrier électronique pour le robot, puis un compte GitHub pour le courrier électronique. Le courriel de Smol ne vous est pas fourni. Vous aurrez à en créer un vous-même.

Afin de pouvoir télécharger le dépôt de GitHub, nous avons ensuite créé un GitHub token. Lorsque vous tenterez de vous connecter, il est possible que vous receviez un message qui vous indique que les mots de passe ne sont plus utilisez. Néanmoins, vous pourrez entrer votre courriel comme identifiant ainsi que votre GitHub token comme mot de passe.

[5] Voir la section des références pour les liens.

9. Annexe

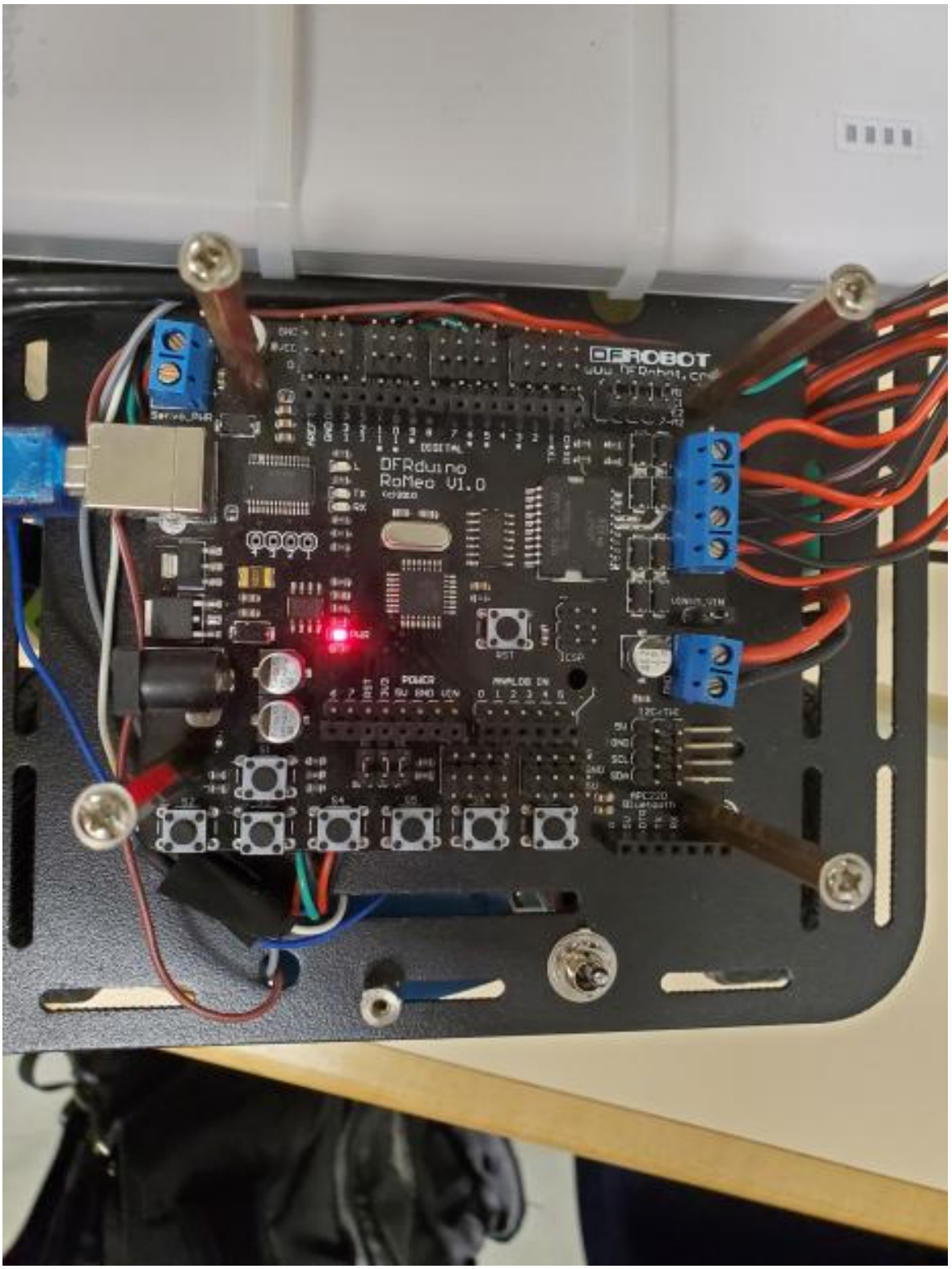


Figure 1: Arduino utilisé pour Smol: DFRduino RoMeo V1.0

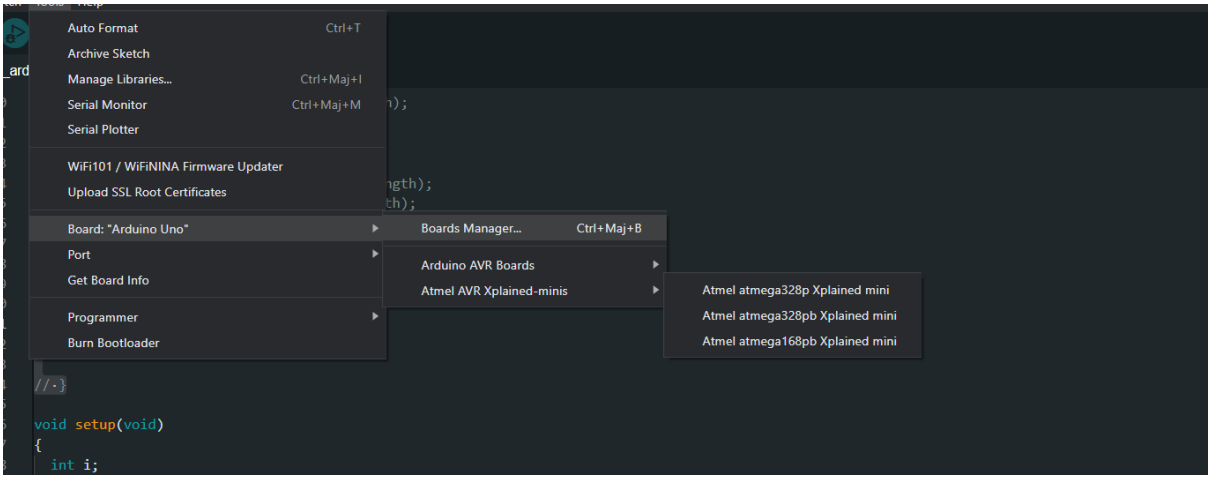


Figure 2: gestionnaire de cartes (Boards Manager): Atmel atmega328p Xplained mini.

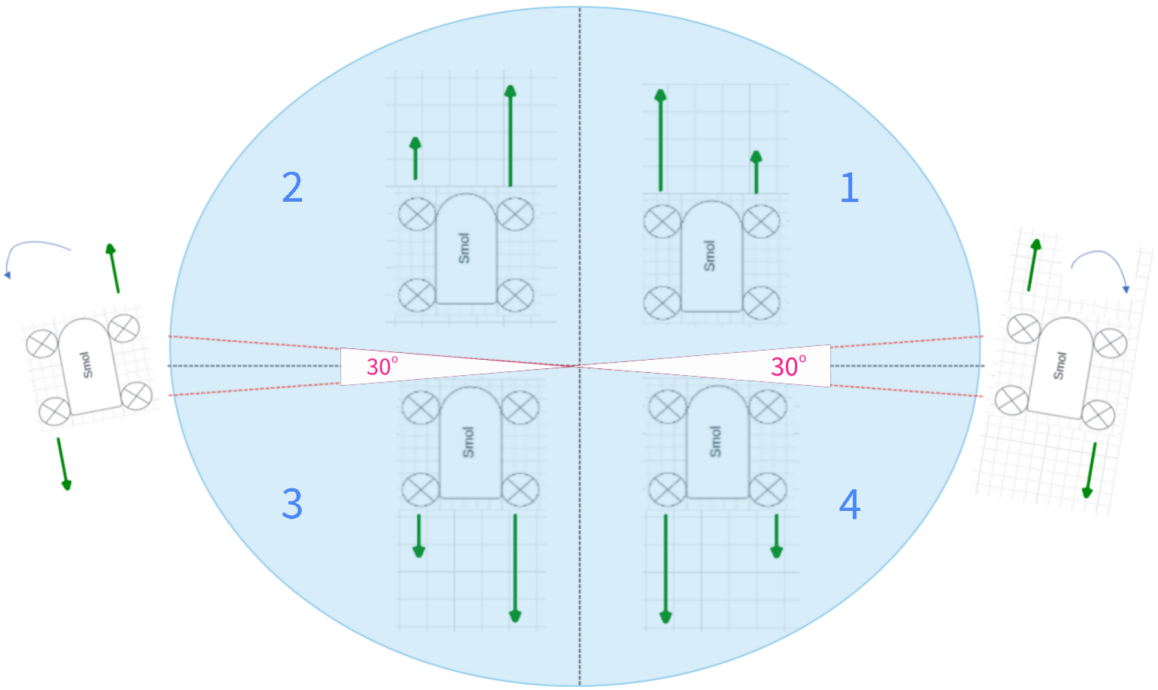


Figure 3: Algorithme de joystick sur Arduino

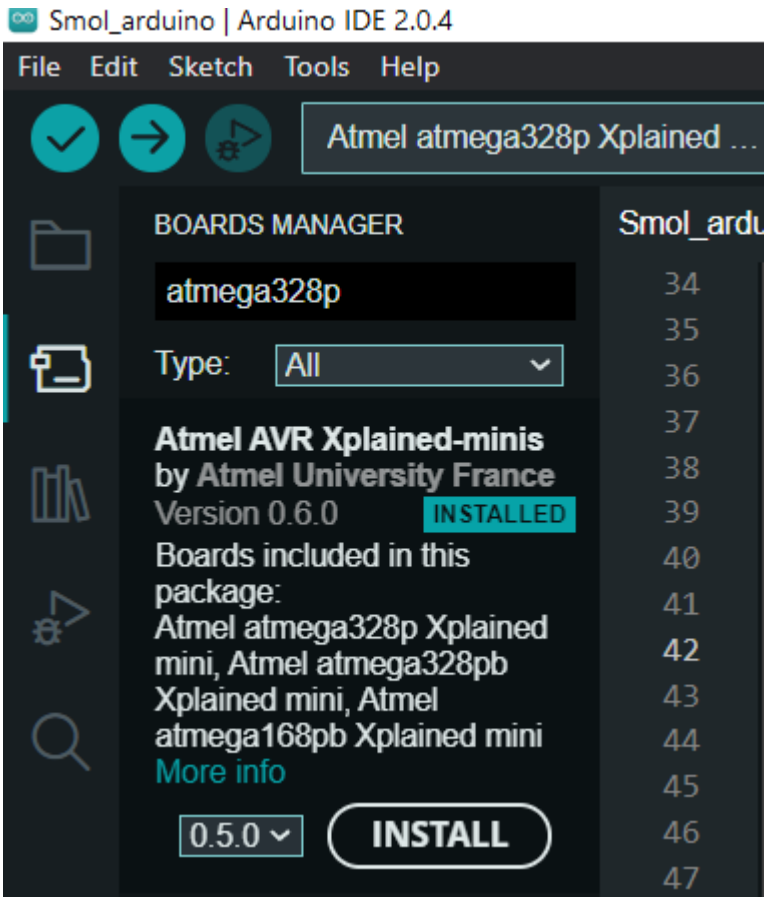


Figure 4: Installation du gestionnaire de carte utilisé: Atmel atmega328p Xplained mini

```
recvCommand:  1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;
Arduino's answer:  command: s
command: 1,s
recvCommand:  1,s;1,q;1,q;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;
Arduino's answer:  command: s
command: 1,s
recvCommand:  1,q;1,q;1,q;1,q;1,s;1,s;1,s;1,s;1,s;1,q;1,q;
Arduino's answer:  command: q
command: 1,q
recvCommand:  1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;1,s;
Arduino's answer:  command: s
command: 1,s
recvCommand:  0,124,68;
Arduino's answer: sin:0.83 cos:-0.56 str:68.00 cad:1 <--: 147 -->: 199
command: 0,124,68
recvCommand:  0,120,76;0,107,99;0,99,100;0,93,99;0,84,99;0,77,100;0,64,100;0,38,100;0,16,99;0,353,100;0,336,91;0,305,51;0,237,39;0,203,42;0,159,48;0,122,70;0,96,100;0,74,100;0,53,99;0,38,100;0,31,100;0,31,70;0,55,24;0,188,42;0,209,64;0,246,55;0,315,100;0,331,100;
Arduino's answer: sin:-0.48 cos:0.87 str:100.00 cad:3 <--: 255 -->: 99
command: 0,331,100
recvCommand:  0,342,100;0,356,100;0,15,99;0,29,99;0,45,99;0,79,99;0,108,99;0,158,78;0,194,86;0,229,93;0,255,84;0,281,83;0,317,97;0,337,100;0,359,99;0,16,99;0,33,100;0,51,99;0,76,99;0,109,100;0,135,100;0,157,72;0,165,20;0,356,55;0,1,100;0,13,99;0,38,100;0,64,100;0,100,99;0,123,100;
Arduino's answer: sin:0.84 cos:-0.54 str:100.00 cad:1 <--: 183 -->: 255
command: 0,123,100
recvCommand:  0,142,100;0,161,96;0,189,34;0,318,50;0,335,98;0,342,100;0,343,100;0,342,99;0,344,100;0,354,100;0,13,100;0,54,100;0,78,99;0,98,100;0,107,100;0,118,99;0,145,85;0,195,46;0,272,47;0,302,70;0,320,100;0,337,99;0,345,100;0,358,99;0,27,99;0,47,100;0,67,99;0,81,100;0,98,99;
Arduino's answer: sin:0.99 cos:-0.14 str:99.00 cad:1 <--: 248 -->: 253
command: 0,98,99
recvCommand:  0,109,99;0,122,99;0,148,54;0,211,27;0,291,53;0,318,97;0,335,100;0,359,100;0,21,99;0,58,100;0,92,100;0,118,100;0,170,87;0,192,87;0,0,0;
Arduino's answer: sin:0.00 cos:1.00 str:0.00 cad:5 <--: 0 -->: 0
command: 0,0,0
Arduino's answer:  command: x
exceded timout

Exited while being in debugger mode
PS C:\Users\Mathblin\Documents\INF8405 - Projets\inf8405-projet\Smol\Server>
```

Figure 5: Affichage du serveur en mode debug avec socket.

```
int move(struct pt* pt) {
  PT_BEGIN(pt);

  while (true) {
    bool debug = false;
    String msg = "";
    int mode = 1000;

    if (!Serial.available()) {
      return;
    }

    msg = Serial.readString();
    mode = getValue(msg, ',', MODE_POSITION).toInt();
    debug = getValue(msg, ',', DEBUG_POSITION).toInt();

    manage_stop_servo(&msg, &debug);

    switch (mode) {
      case MODE_JOYSTICK:
        mode_joystick(&msg, &debug);
        break;
      case MODE_GYROSCOPE:
        mode_gyroscope(&msg, &debug);
        break;
      case MODE_SECOND_SERVO:
        hat.activate(&msg, &debug);
        break;
    }
  }

  PT_END(pt);
}
```

Figure 6: Communication sérielle bloquante par le fils d'exécution *move*.

```
void mode_joystick(String* msg, bool* debug) {
  advance_joystick(msg, &angle_rad, &sin_result, &cos_result, &strength, &cad);

  if (*debug && Serial.availableForWrite() > 30) {

    Serial.print("sin:" + String(sin_result) + " cos:" + String(cos_result) + " str:" + String(strength) + " cad:" + String(cad)
                 + " <--: " + String(left_strength) + " -->: " + String(right_strength));
  }
  else if (Serial.availableForWrite() > 30) { // Tells the server that Arduino is ready to receive a command
    Serial.print("ok");
  }
}
```

Figure 7: Exemple de communication sérielle en mode joystick avec envoi de données.

```
void setup(void) {
  PT_INIT(&ptCane);
  PT_INIT(&ptHat);
  PT_INIT(&ptMove);

  cane.init(8, 45, 90);
  hat.init(9, 0, 90);
  int i;
  for (i = 4; i <= 7; i++)
    pinMode(i, OUTPUT);
  Serial.begin(9600);  //Set Baud Rate
}

void loop(void) {
  PT_SCHEDULE(caneServo(&ptCane));
  PT_SCHEDULE(hatServo(&ptHat));
  PT_SCHEDULE(move(&ptMove));
}
```

Figure 8: Initialisation et ordonnancement des fils d'exécution.

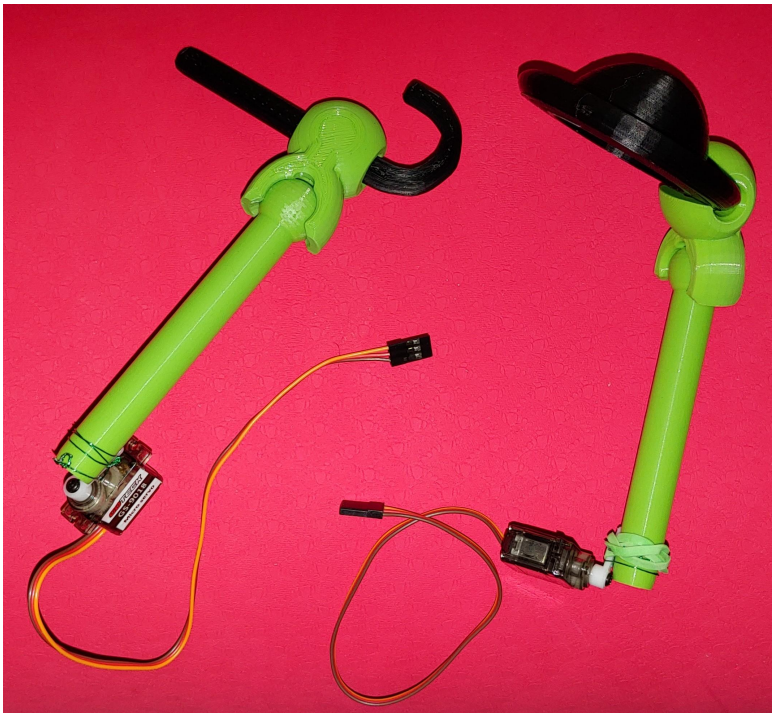


Figure 9: Bras avec canne, chapeau et servomoteurs

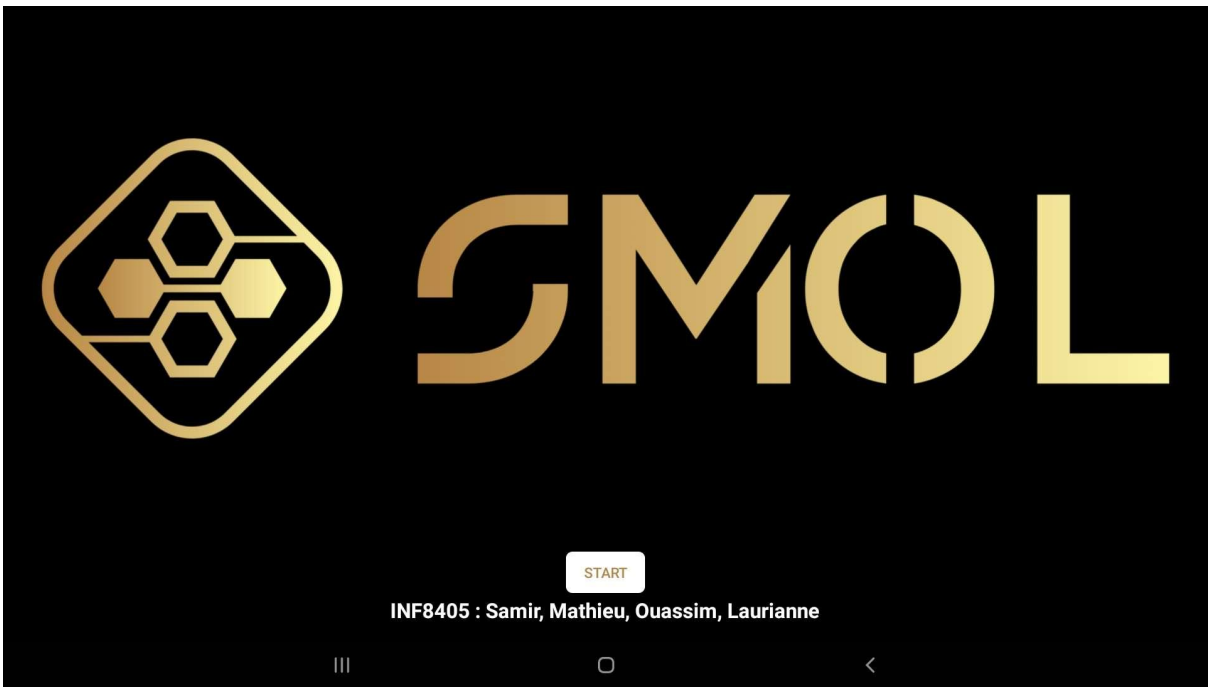


Figure 10: La première page: page d'accueil avec les noms des programmeurs

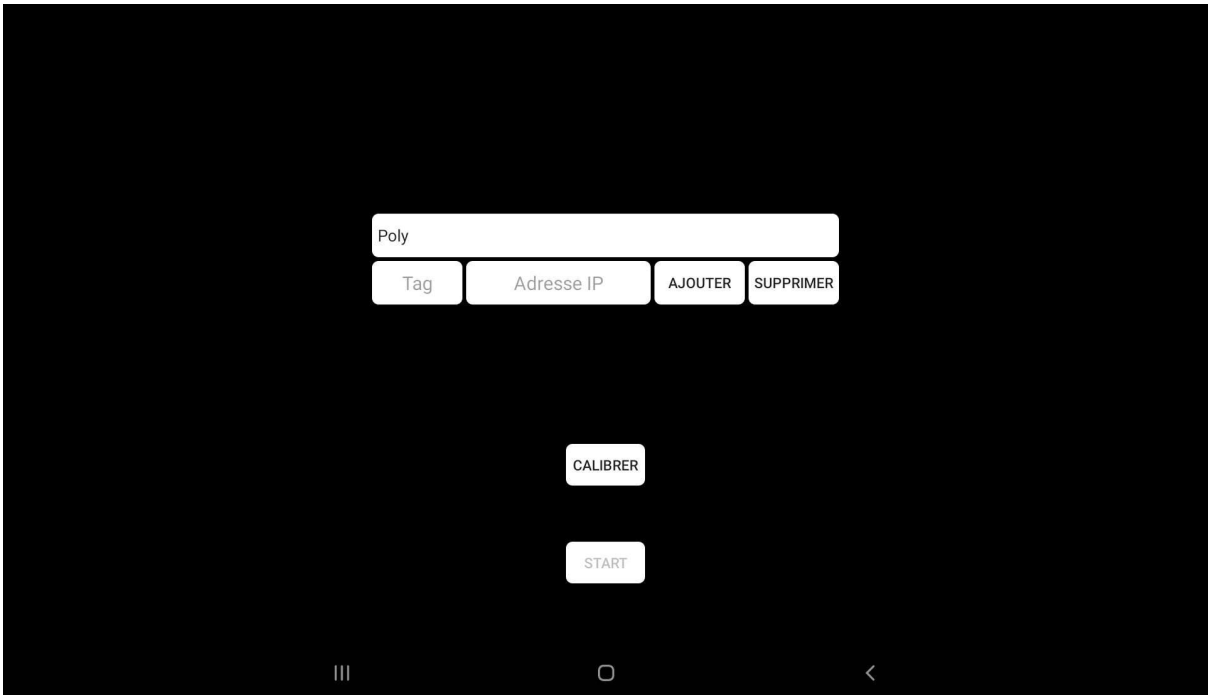


Figure 11: La deuxième page: la sélection d'adresse IP et le calibrage

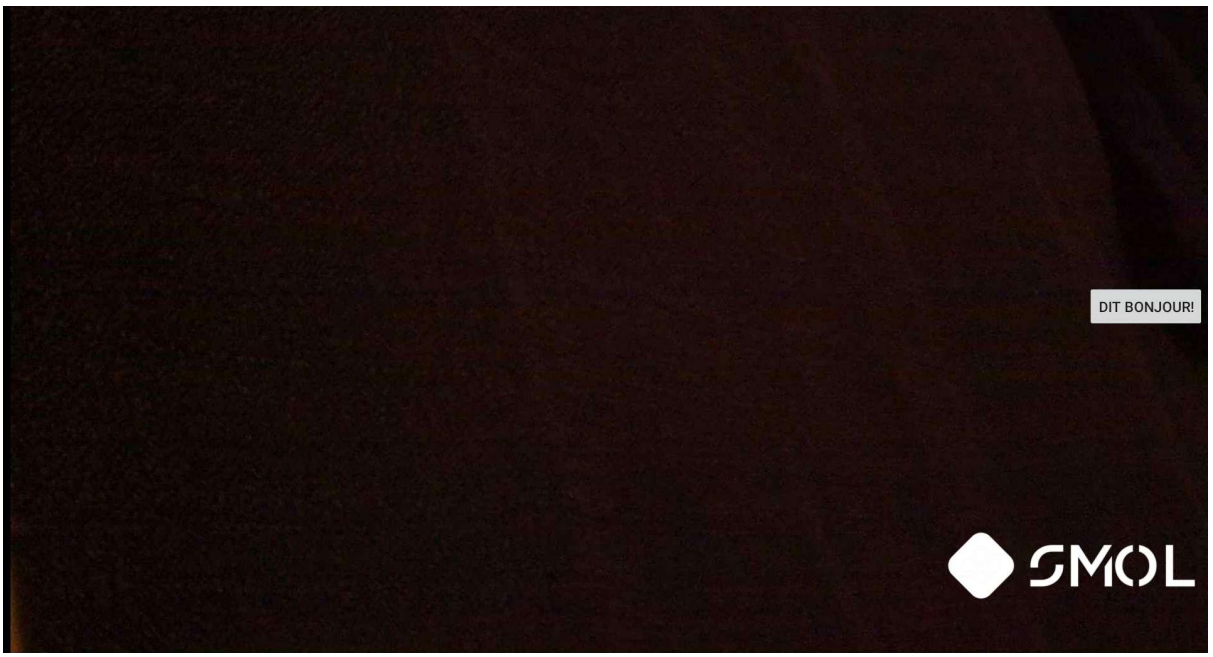


Figure 12: Le mode Gyroscope

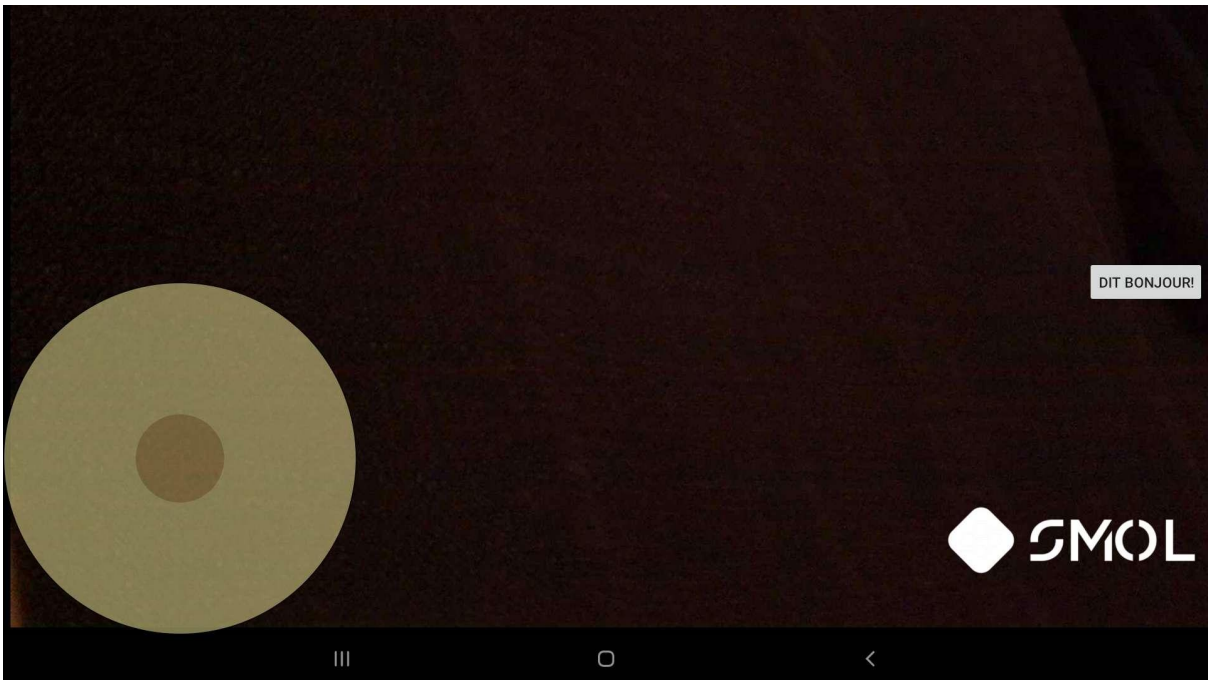


Figure 13: Le mode joystick

10. Références

Arduino. (2023). *Sketch*. https://www.arduino.cc/en/Tutorial/Sketch

pypi. (2023). *pyserial*. https://pypi.org/project/pyserial/

Python. (23 avril 2023). *Socket*. https://docs.python.org/3/library/socket.html