

PÔLE PROJET ROBOTIQUE

Robot Mobile Suiveur

Auteurs :

- Andrei Radlovic
- Kelian Tinen Toulac
- Joel Happi Kouemo
- Fernando Tovar Sanchez-Trujillo
- Hugo De Boscherre

Encadrants :

- Maria Makarova
- Morgan Roger

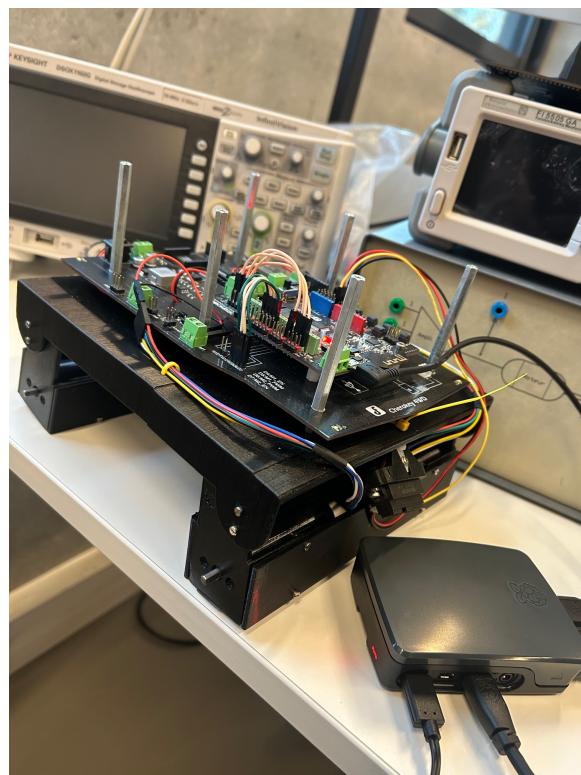
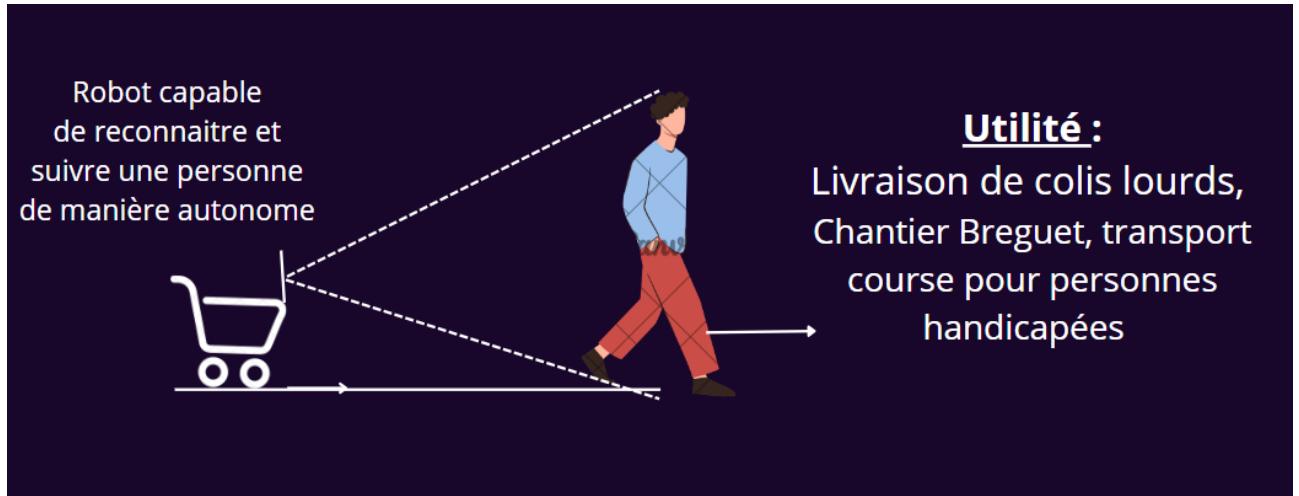


Table des matières

Introduction	2	
1	Notre organisation	4
1.1	Objectifs initiaux	4
1.2	Etat d'avancement du Projet à sa reprise par notre équipe	4
1.3	Situation Initiale et Problèmes rencontrés	5
1.4	Planning du Projet	6
2	Architecture du Robot	7
2.1	Présentation globale des connexions entre les éléments du robot	7
2.2	Présentation détaillée des connexions physiques entre les composants du Robot .	8
3	Les composants du robot	10
3.1	Les moteurs et les roues : 25GA 370 du Kit Omni-Wheel	10
3.2	Camera : HuskyLens	11
3.3	Capteurs infrarouge Shart 2Y0 A21	11
3.4	Gyroscope MPU ITG 6050	12
3.5	Version des logiciels (Raspberry Pi 3 Model B+ et ordinateur)	12
4	Explication des codes	13
4.1	Protocole de communication	13
4.2	Tests	16
4.2.1	Test des Moteurs	16
4.2.2	Test du Capteur Infrarouge	16
4.2.3	Test de la Caméra	16
4.2.4	Test du Servomoteur	17
4.2.5	Test du Gyroscope	17
5	Modes d'emplois	18
5.1	Contrôle à distance de la Raspberry	18
5.2	Premiers tests à faire pour prendre en main la Arduino UNO et l'interface Arduino	20
5.3	Connexions Arduino - Raspberry	20
5.3.1	Pour la Arduino Uno	20
5.3.2	Pour la Arduino Nano	20
5.3.3	Pour la Raspberry	20
5.4	Test composants	21
5.4.1	Test moteurs	21
5.4.2	Test capteur Infrarouge	21
5.4.3	Test Servo	21
5.4.4	Test caméra huskylib	22
5.4.5	Test gyro	22
5.5	Si l'un des tests ne marche pas	22
Conclusion et perspective	23	

Introduction

Le projet robot mobile suiveur a pour vision à long terme de concevoir un robot capable de suivre un opérateur. Il pourrait par exemple trouver son utilité dans la livraison de colis ou pour le transport des charges sur un chantier. Le robot serait capable de suivre un opérateur. Cela implique de pouvoir reconnaître l'opérateur, ne pas le perdre de vue, le distinguer des autres individus dans l'environnement, avancer à la même vitesse que lui et aussi d'éviter les obstacles de l'environnement de suivi.



1 Notre organisation

1.1 Objectifs initiaux

À la reprise du projet Robot mobile suiveur, nous avons discuté avec nos encadrants afin de fixer les objectifs du projet. Les objectifs du projet Robot Mobile sont de concevoir un Robot Mobile :

- Capable d'identifier une personne à suivre dans le cas où plusieurs sont présentes (Via un algorithme de reconnaissance par couleurs ou par formes)
- Robuste vis-à-vis des conditions extérieures quant au traitement d'image (luminosité, etc)
- Capable de détecter les obstacles et de les éviter
- Dont la commande est souple notamment pour le contournement d'obstacles
- Qui soit le plus rapide possible dans ses déplacements

1.2 Etat d'avancement du Projet à sa reprise par notre équipe

Pour amorcer notre projet, les encadrants nous ont remis un prototype du robot, un rapport récapitulatif de l'équipe qui a œuvré sur le projet le semestre précédent, ainsi que les codes nécessaires au fonctionnement du robot. Voici un aperçu des réalisations des précédents contributeurs :

Critère	Avancement
Solidité	Résistant à la manipulation
Autonomie	30 minutes
Vitesse de pointe	Pas de données
Distance à l'utilisateur	30 cm
Détection d'obstacles	Aucune Collision, Arrêt
Robustesse Logicielle	Perte de visuel < 10 % restauré en 0.5 s
Confinement à une zone précise	Non réalisé

TABLE 1 – Tableau récapitulatif de l'avancée des précédents contributeurs du projet

Etant donné que ce projet nous a été transmis par une équipe qui a travaillé sur ce projet le dernier semestre, nous avons précisé ces objectifs dans une perspective d'amélioration du travail mené par l'équipe précédente. Les voici présentés :

- Rendre résistants aux chocs puissants (Ajouts de protection sur la structure)
- Changer la batterie afin de faire passer l'autonomie de 30 min à 2h
- Améliorer la distance à l'utilisateur, ce qui pourrait impliquer un changement de caméra
- Améliorer vitesse de pointe, ce qui pourrait impliquer un changement des roues ou une modification de l'algorithme
- Confiner le robot à une zone précise par rapport à l'utilisateur qu'il suit

1.3 Situation Initiale et Problèmes rencontrés

Nous avons reçu un robot mobile comprenant : Une raspberry, une carte Arduino, une carte PCB, une Batterie pour la raspberry , 4 roues et 4 moteurs et un chassis. Nous possédions également les codes de l'équipe qui avait travaillé sur le robot l'année précédente.

Le robot, dans sa version initiale délivrée par les encadrants en septembre, était censé remplir les objectifs énoncés précédemment. Malheureusement, nous ne pouvions pas lancer les codes fournis afin d'effectuer un premier suivi test ayant les performances "de base" escomptées. Ceci est sûrement dû à plusieurs facteurs : la raspberry présente sur le robot était disfonctionnelle, et les codes incomplets. C'est pourquoi lors de notre travail nous avons changé de carte raspberry et réécrit les codes de base donnant accès à la commande des capteurs, et que nous essaierons d'écrire ce rapport de la manière la plus concise possible vis-à-vis de la reprise.

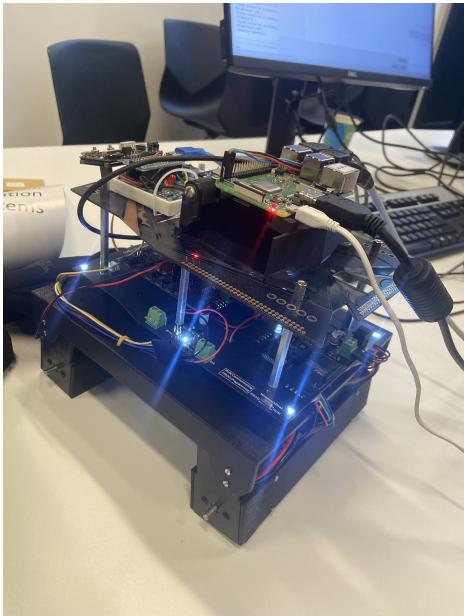


FIGURE 1 – Robot Monté sans ses Roues



FIGURE 2 – Roues du robot et autres accessoires

1.4 Planning du Projet

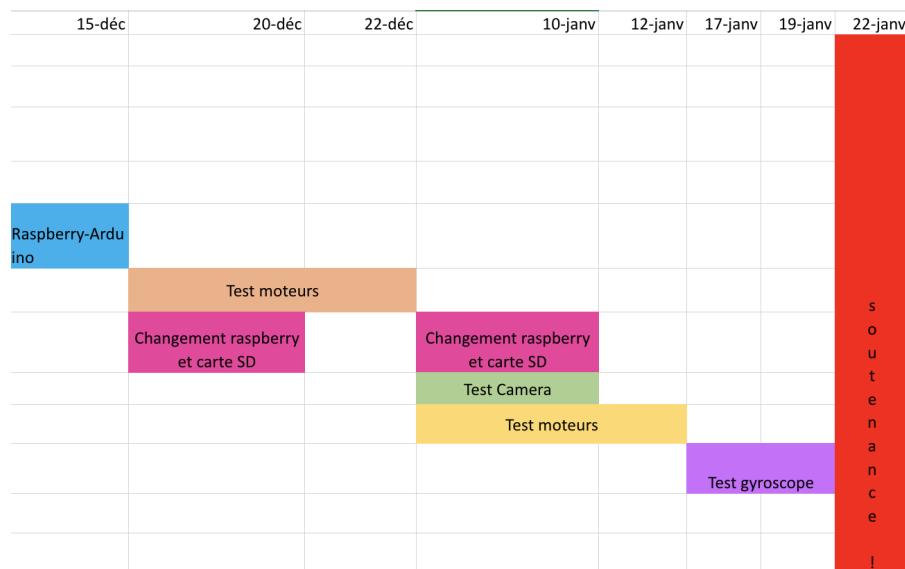


FIGURE 3 – Diagramme de Gantt du Projet

2 Architecture du Robot

2.1 Présentation globale des connexions entre les éléments du robot

Notre robot est constitué de trois cartes distinctes : une Raspberry Pi, une carte Arduino Uno et une carte Arduino Nano. En outre, nous disposons d'une variété de composants. En ce qui concerne les capteurs, nous utilisons un gyroscope, un capteur infrarouge et une caméra pour recueillir des données sur l'environnement. Du côté des actionneurs, nous avons quatre moteurs montés sur les roues, ainsi qu'un servomoteur responsable du déplacement de la caméra.

Pour traiter cette diversité d'informations et transmettre les commandes aux moteurs, nous avons mis en place les connexions logiques suivantes :

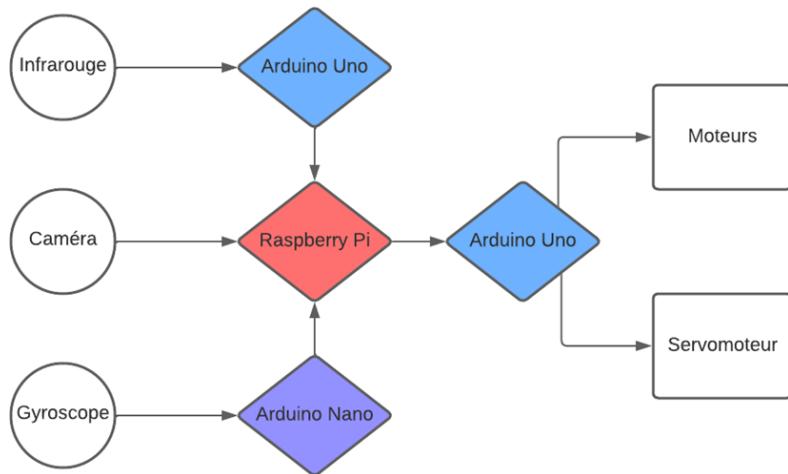


FIGURE 4 – Diagrammes d’interaction entre les différents systèmes utilisés

Il est essentiel de noter que ce schéma constitue une simplification des flux d'information au sein du robot. Son objectif est de permettre au lecteur de comprendre que le gyroscope est commandé et transmet ses données à la carte Arduino Nano, l'infrarouge à la carte Arduino Uno, et la caméra directement à la Raspberry. Après avoir reçu les informations de ses capteurs, les cartes Arduino envoient ces données à la Raspberry, où elles sont traitées. Enfin, la Raspberry envoie les commandes aux moteurs des roues et au servomoteur de la caméra via la carte Arduino Uno.

Effectivement, dans la réalité, étant donné que les codes sont exécutés en Python via la Raspberry, le processus commence avec la Raspberry émettant l'ordre de lire les capteurs aux cartes Arduino. Plus spécifiquement, la carte Arduino Nano est sollicitée pour récupérer les données du gyroscope, tandis que la carte Arduino Uno est chargée de collecter les informations de l'infrarouge. Une fois ces données acquises, le processus se poursuit conformément à la description précédente, avec les cartes Arduino transmettant les informations à la Raspberry, où elles sont traitées, et la Raspberry envoyant ensuite les commandes aux moteurs des roues et au servomoteur de la caméra par le biais de la carte Arduino Uno.

2.2 Présentation détaillée des connexions physiques entre les composants du Robot

En pratique, les connexions physiques sont indiquées dans les figures qui suivent :

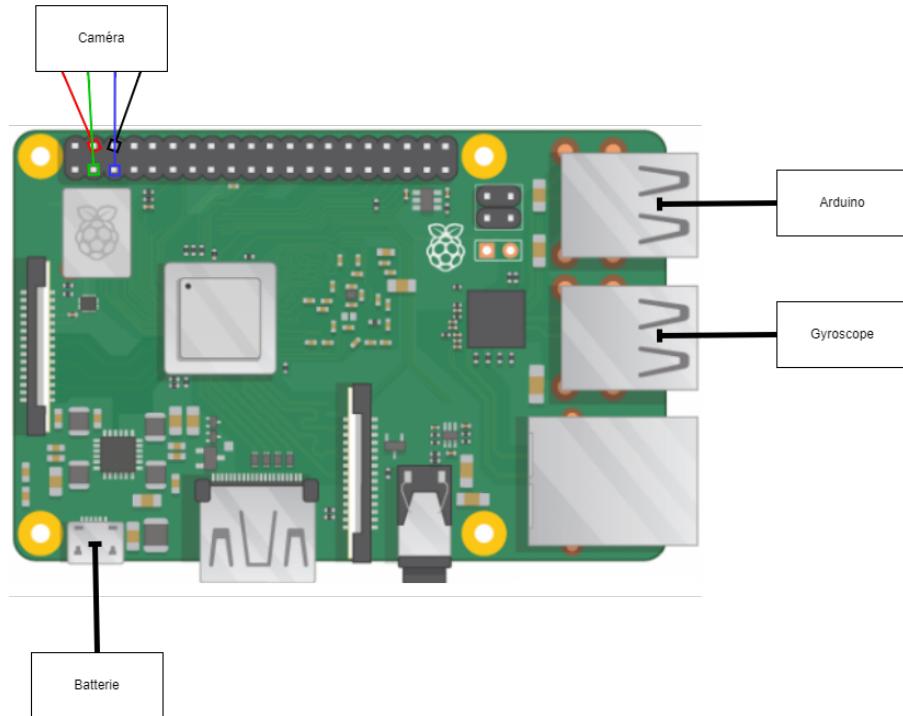


FIGURE 5 – Carte de connexions pour la carte Raspberry

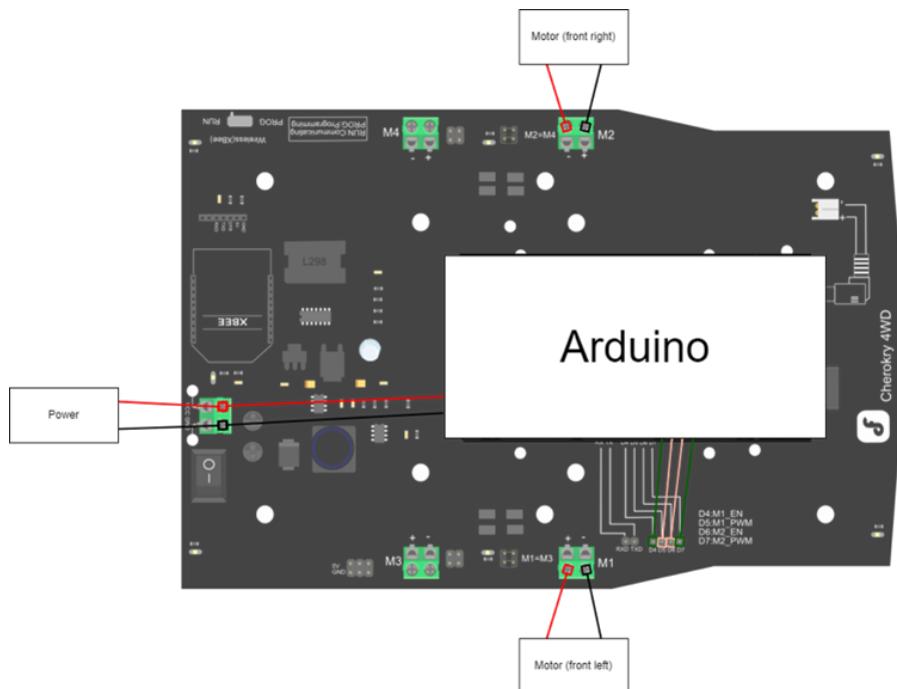


FIGURE 6 – Carte de connexions pour la carte Arduino

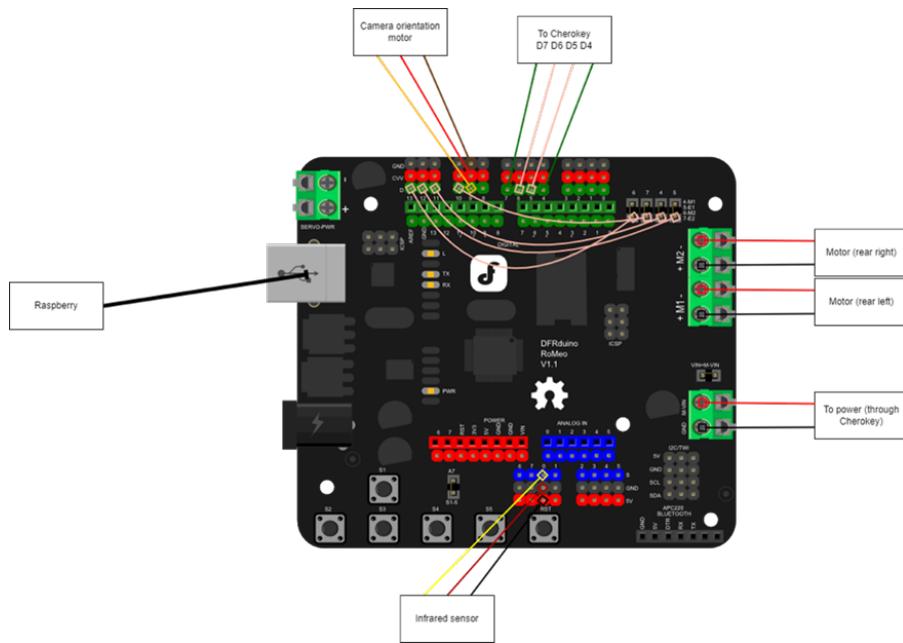


FIGURE 7 – Carte de connexions pour le circuit Cherokey

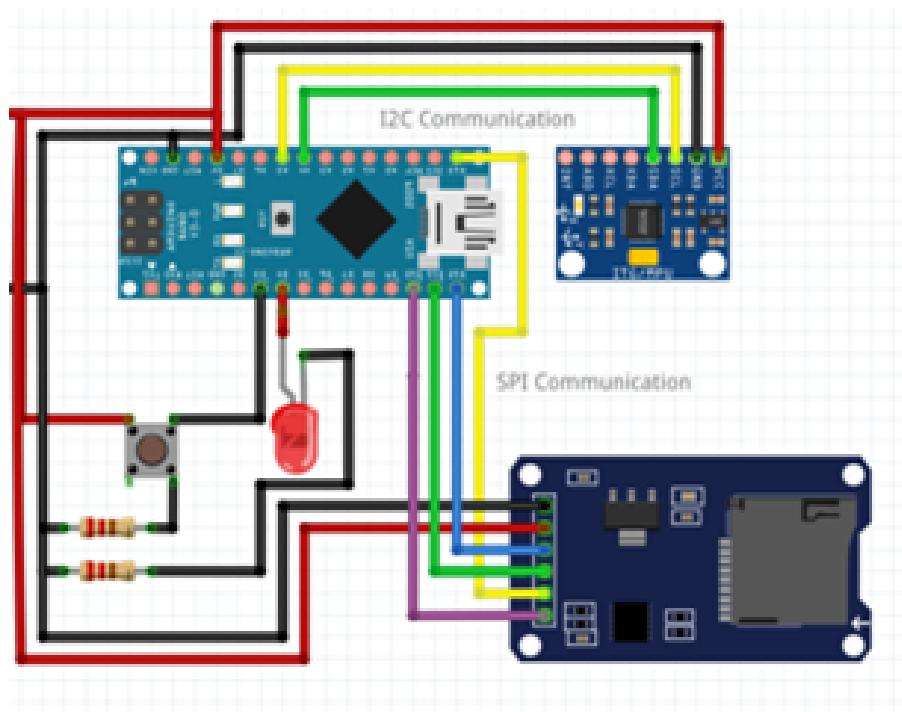


FIGURE 8 – Carte de connexions pour la carte Arduino reliée au gyroscope

La communication entre la Raspberry Pi et les cartes Arduino s'effectue en série via le port USB. L'Arduino Uno, connecté aux capteurs et moteurs, utilise un baud-rate de 115200, tandis que l'Arduino Nano, relié au gyroscope, fonctionne à un baud-rate de 9600. Cette configuration permet d'établir des échanges de données efficaces et adaptés à chaque composant du système.

3 Les composants du robot

Une grande partie de notre travail consistait à faire fonctionner les composants du robot en utilisant les codes de test fourni par les anciens. Certainement parce qu'on a dû changer de Raspberry, de carte SD et de version de logiciel par rapport aux anciens, aucun des codes de test fourni ne fonctionnait, nous avons du en recréer. Nous allons vous exposer dans cette partie les composants exacts que nous avons utilisé avec leur fiche technique et leurs entrées et sortie ainsi que les versions des logiciels. Cette présentation détaillée des composants a pour but de vous aider d'une part à reconstituer le robot au cas où un composant manquerait et d'autre part à gérer d'éventuels problèmes de compatibilité qui pourraient se poser.

3.1 Les moteurs et les roues : 25GA 370 du Kit Omni-Wheel

Les informations fournies dans cette partie sont issues du site : <https://joy-it.net/de/products/COM-Motor06>

Le robot est équipé de quatre moteurs issus du kit Omni-Wheel, fixés sur un châssis imprimé en 3D. Chacun des quatre moteurs entraîne une roue du robot mobile.

Le kit Omni-Wheel comprend un ensemble de quatre roues avec des moteurs intégrés, permettant un déplacement omnidirectionnel. Ces roues Omni sont dotées de rouleaux orientés à 45° par rapport à l'axe de rotation, ce qui facilite les déplacements latéraux sans altérer l'orientation du châssis. Adapté à la robotique, ce kit est compatible avec les SBCs(Single Board Computers) et les microcontrôleurs.

Spécifications du moteur (25GA370) :

- Rapport de transmission : 21,3
- Courant sans charge : 100mA
- Vitesse sans charge : 280 tr/min
- Couple nominal : 0,8 kg/cm
- Vitesse nominale : 200 tr/min
- Tension nominale : 6 Volts
- Courant nominal : 0,75 mA
- Couple de retenue max. : 4,5 kg/cm
- Courant de retenue : 1,3 A

Les détails du câblage, comprenant les couleurs correspondant aux polarités, sont fournis sur le site joy-it.net. Le kit (numéro d'article COM-Motor06) inclus 4 moteurs avec encodeurs, 4 roues Omni, des moyeux métalliques et plastiques, ainsi que les vis et boulons nécessaires. Des fiches techniques et des guides d'utilisation sont disponibles en téléchargement en allemand et en anglais sur le site.

NB : Nous avons changé les roues et opté pour des roues du même type mais de plus grand diamètre afin d'optimiser la vitesse de pointe du Robot.



FIGURE 9 – Moteur du modèle 25GA370

3.2 Camera : HuskyLens

Le Robot est équipé d'une camera Huskylens du fabricant DFRobot rattaché à un Servo Moteur permettant de modifier son inclinaison. HuskyLens est un capteur de vision artificielle AI qui offre des fonctionnalités telles que la reconnaissance faciale, le suivi d'objets et la détection de lignes. Compatible avec des cartes de contrôle populaires comme Arduino et Raspberry Pi via les ports UART/I2C. Doté d'un écran IPS de 2,0 pouces, il permet des réglages sans PC. La technologie de machine learning intégrée, avec la puce d'IA Kendryte K210, améliore la reconnaissance au fil du temps. Ses applications incluent le contrôle gestuel, les robots autonomes et les jouets interactifs. Les spécifications comprennent un processeur Kendryte K210, un capteur OV2640 de 2,0 mégapixels, une tension d'alimentation de 3,3 à 5,0V et des dimensions de 52mm * 44.5mm.

L'API Python disponible sur le répertoire git Huskylib permet de récupérer les informations de détection d'objet de la caméra directement dans un script Python. Les codes ont été copiés sur la mémoire de la Raspberry.



FIGURE 10 – Camera HuskyLens - Image extraite du site <https://www.dfrobot.com/>

3.3 Capteurs infrarouge Sharp 2Y0 A21

Le robot est équipé d'un capteur infrarouge du constructeur Sharp, permettant de mesurer des distances et donc d'évaluer la distance aux potentiels obstacles, afin que notre robot puisse les contourner. Les distances de détection couvertes sont de 10 à 80 cm, ce qui est largement suffisant pour l'utilisation pratique que nous souhaitons faire du robot. Il ne peut néanmoins pas détecter la couleur des objets ciblés par mesure de distance, c'est pourquoi il est couplé à la caméra HuskyLens pour que le robot ait toutes les fonctionnalités nécessaires.

Lien de la documentation technique : <https://docs.rs-online.com/6e24/A70000008623624.pdf>



FIGURE 11 – Capteur infrarouge Sharp

3.4 Gyroscope MPU ITG 6050

Le robot est équipé d'un module accéléromètre-gyroscope modèle MPU 6050. Ce modèle est composé d'un capteur (gyroscope) couplé avec une carte arduino Nano, qui communique avec les autres cartes du robot. Le capteur mesure l'inclinaison du robot autour de son axe z, afin de pouvoir effectuer des manœuvres de rotation et se positionner angulairement pour éviter les obstacles.

Lien de la documentation technique : <https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Components/General%20IC/PS-MPU-6000A.pdf>

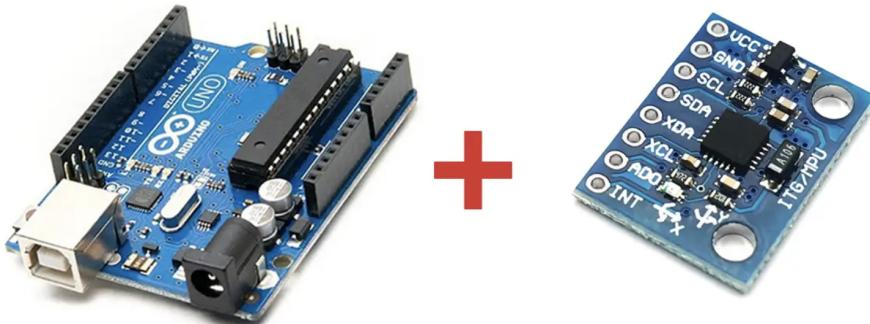


FIGURE 12 – Gyroscope + carte Arduino MPU 6050

3.5 Version des logiciels (Raspberry Pi 3 Model B+ et ordinateur)

Nous avons listé les versions des différents logiciels utilisés, avec ces versions vous êtes normalement garanti que ça fonctionne.

Logiciels Raspberry Pi 3 Model B+ :

- Arduino 1.8.19
- Thonny 3.3.14 (associé à python 3.9.2)
- Le VNC viewer est initialement sur la Raspberry sa version ne sera pas un problème

Logiciels sur l'ordinateur :

- RealVNCViewer 7.6.1

4 Explication des codes

Une grande partie de notre travail a consisté à assurer la communication correcte entre le Raspberry PI et les différents capteurs et composants du robot, ce qui est réalisé en utilisant les cartes Arduino comme intermédiaires. Compte tenu que le langage de programmation permettant de communiquer avec les cartes Arduino est le C++ et que nous avons choisi de programmer en Python via le Raspberry PI, nous trouverons des codes dans les deux langages.

En C++ :

- Main Arduino (dossier)
- Main Nano (dossier)
- MPU6050light121 (dossier)
- test_gyro.ino
- test_servo.ino

En Python :

- robust_serial (dossier)
- connexion.py
- test_connexion.py
- huskytest.py
- huskylib.py
- detecthusky.py
- test_moteur.py
- test_infrarouge.py
- test_servo.py
- test_gyro.py
- main_suivi.py

4.1 Protocole de communication

Nous commencerons par expliquer les codes qui gèrent la connexion du Raspberry aux cartes Arduino. Cette connexion utilise plusieurs fichiers pour être réalisée. Comme il s'agit d'un protocole de communication, la communication doit être gérée à la fois du côté du Raspberry et du côté des cartes Arduino.

Fichiers de communication côté Arduino :

En ce qui concerne les cartes Arduino, nous utilisons les fichiers que nous pouvons trouver dans les dossiers Main Arduino et Main Nano. Main Arduino contient quatre fichiers :

- Order.h : c'est un fichier qui contient les différentes ordres que la carte Arduino peut recevoir ou envoyer. Par exemple, l'ordre utilisé pendant le protocole de connexion est HELLO.
- Parameters.h : contient des paramètres généraux au robot
- Slave.h : contient une brève description des fonctions de base utilisées dans le main propres au Robust Arduino Serial Protocol
- Main_arduino.ino : Intègre les fichiers précédents et gère la communication du côté Arduino. Ce fichier doit être téléchargé sur la carte Arduino Uno puisqu'il explique à la carte comment lire les ordres reçus depuis le Raspberry Pi, comment les interpréter et comment répondre. C'est un fichier complexe qui contient de nombreuses variables et fonctions, certaines d'entre elles ne sont pas utilisées par le robot pour réaliser le suivi

souhaité. Cependant, nous vous conseillons de toucher le moins possible à ce fichier puisqu'il permet actuellement au robot de gérer correctement la communication et il est très facile de commettre une erreur qui la compromette. Les fonctions les plus importantes qu'il contient sont :

- **Get_messages_from_serial** (ligne 208) : lit l'ordre qui lui a été envoyé depuis le Raspberry Pi et agit en conséquence. En fonction de l'ordre reçu, il reconnaîtra avec quel composant nous essayons de communiquer et lancera d'autres fonctions spécifiques qui nous permettront de réaliser la tâche souhaitée. Par exemple, si nous nous concentrons sur l'ordre READSENSORS, cela déclenche l'appel à deux autres fonctions : Measure_IR, qui est chargée de lire la distance entre le capteur Infra Rouge et l'objet le plus proche se trouvant dans son champ de vision, et write_i8, qui envoie son argument au raspberry dans un format d'entier de 8 bits.
- **Update_motors_orders** (ligne 138) : en fonction des dernières vitesses lues par la fonction get_messages_from_serial (qui est capable de recevoir des messages sous forme entière provenant de la raspberry), cette fonction enverra aux moteurs des roues la puissance nécessaire pour atteindre ces vitesses. Notez que si les vitesses sont négatives alors les roues tourneront dans le sens inverse et que la vitesse maximale est de 100.

Le dossier **Main_Nano** contient également 4 fichiers le principal étant le fichier **main_nano.ino**. Celui-ci fonctionne à presque de la même manière que **main_arduino.ino**. Ce fichier doit être téléchargé sur la carte Arduino Nano pour gérer la communication et les ordres provenant du Raspberry PI. La fonction get_messages_from_serial de ce fichier contient uniquement 4 ordres différents. HELLO et ALREADY_CONNECTED fonctionnent de la même manière que dans le main_arduino.ino. Cependant, READ_GYRO est utilisé pour lire et envoyer l'angle de rotation du robot par rapport à l'axe z à travers un gyroscope ; et l'ordre RESET_GYRO vise à établir l'angle actuel du robot comme le nouvel origine (angle 0), mais actuellement cela ne fonctionne pas. Il est important de noter que ce fichier utilise une bibliothèque propre au modèle de gyroscope utilisé, MPU6050_light, qui doit être installée dans l'IDE Arduino. Sur la carte SD que nous utilisons, elle est déjà installée mais si vous changez de carte, vous devrez l'installer vous-même. Gardez à l'esprit que pour l'installation, la Raspberry PI doit être connectée à Internet, ce qui peut également générer des problèmes et une perte de temps au cas où il ne le serait pas. Cette bibliothèque apporte de nombreuses fonctions intégrées comme par exemple **mpu.getAngleZ**, qui est la fonction que nous utilisons pour obtenir l'angle de rotation dans l'ordre READ_GYRO, ou la fonction **mpu.update** qui se trouve à l'intérieur de la boucle et qui va actualiser cet angle continuellement pour prendre en compte de nouveaux mouvements du robot.

Fichiers de communication côté Raspberry Pi :

En ce qui concerne la gestion de la connexion depuis le Raspberry PI, la communication avec les cartes Arduino est gérée par les fichiers contenus dans le dossier **robust_serial** et par le fichier **connexion.py**. Avant de décrire le contenu de chacun de ces documents, il est important de spécifier que dans notre projet, un **serial_file** est une variable qui est associée à une des cartes Arduino et qui représente l'adresse à laquelle nous pouvons envoyer des données ou de laquelle nous pouvons en recevoir. Elle est généralement utilisée comme input des fonctions de base qui permettent d'envoyer ou de recevoir des informations des cartes Arduino.

Dans le dossier **robust_serial**, nous trouvons plusieurs fichiers, nous nous intéresserons uniquement à **robust_serial.py**. **Robust_serial.py** joue un rôle similaire avec le raspberry

à celui que les fichiers main jouent avec les cartes Arduino. Il définit les fonctions de base qui nous permettent de lire les messages provenant des Arduinos ainsi que de leur envoyer les ordres nécessaires. Les fonctions avec lesquelles nous devons nous familiariser sont :

- `Read_i8`, `read_i16` : nous permet de lire un entier de 8 ou 16 bits envoyé par une des cartes Arduino. Prenez en compte que en fonction de l'ordre que nous avons envoyé auparavant, nous attendrons de lire une grandeur différente. Par exemple, en envoyant à l'Arduino Nano `READ_GYRO`, j'espère recevoir une valeur entière de 8 bits qui représente l'angle de rotation du robot. Comme input, il prend le serial file avec lequel il doit communiquer.
- `Write_i8`, `write_i16` : envoie un entier de 8 ou 16 bits à la carte demandée. Comme input, ils prennent le serial file avec lequel ils doivent communiquer et l'entier qu'ils doivent envoyer.
- `Write_order` : envoie l'ordre spécifié au serial file demandé. Cela se fait à travers la fonction `write_i8`, puisqu'à chaque ordre est associé un entier spécifique que les cartes Arduino sont capables de reconnaître à travers leur respectif main.

Le fichier `Connexion.py` est le fichier dans lequel est gérée la première connexion entre le raspberry et la carte Arduino désirée. Nous y trouvons la fonction `connect_to_arduino`, qui prend comme input "`uno`" ou "`nano`" en fonction de la carte Arduino à laquelle nous voulons nous connecter. À l'intérieur de la fonction, il y a deux paramètres qui correspondent au port usb du Raspberry auquel chacune des cartes Arduino est connectée (`port_nano` et `port_uno`). À travers cette fonction et en utilisant ces paramètres, la première connexion avec la carte désirée est établie et le serial file correspondant à celle-ci est créé.

NB : Il est crucial de noter que le choix du port et du baudrate est tributaire du type de carte utilisée. Ainsi, il est essentiel de distinguer deux scénarios en fonction de la connexion à la Nano ou à la Uno. En cas de changement de carte Arduino, il sera nécessaire d'ajuster à la fois le port et le baudrate pour chaque carte. L'appel de la fonction `connect_to_arduino` représente la première étape incontournable pour établir la communication avec une carte Arduino. C'est pourquoi toutes les séquences de test des différents composants du robot débutent par cette opération fondamentale.

4.2 Tests

4.2.1 Test des Moteurs

Comme indiqué précédemment, la phase initiale de tous les tests consiste à établir une connexion avec la carte Arduino appropriée. Pour les moteurs, il s'agit de la carte Arduino Uno, et cette opération est réalisée en utilisant la fonction `connect_to_arduino` avec "uno" comme argument. Une fois la connexion établie, le test englobe deux fonctions distinctes.

Move C'est une fonction qui garantit l'établissement de la vitesse de chaque roue de manière indépendante. Cette fonction utilise l'ordre MOTOR.

Stop Elle établit la vitesse des quatre roues à zéro, c'est-à-dire qu'elle arrête le mouvement du moteur. Elle utilise l'ordre STOP.

4.2.2 Test du Capteur Infrarouge

Le capteur infrarouge a pour objectif de mesurer la distance entre le robot et tout obstacle potentiel sur son trajet pendant le suivi de la personne. Par conséquent, le test a pour but d'acquérir cette distance. Dans cette optique, la première étape consiste à établir une connexion avec la carte Arduino Uno. Ensuite, la fonction `lecture_IR` est initiée, utilisant la commande READSENSORS pour récupérer une valeur entière de 8 bits représentant la distance en centimètres de l'objet le plus proche du capteur.

La distance maximale est de 80 centimètres et la distance minimale est de 10 cm. Cependant, il est possible que le capteur nécessite une calibration et qu'une distance réelle de x cm ne corresponde pas à une valeur de x renournée par le programme du capteur. Néanmoins, le test montre que les valeurs reçues sont cohérentes car si nous éloignons la main, la distance renournée augmente et fournit des valeurs significatives, bien que nous n'ayons pas pu mesurer leur précision.

4.2.3 Test de la Caméra

L'objectif de la caméra est de déterminer les coordonnées du barycentre de la personne que le robot doit suivre. Une fois ces coordonnées obtenues, le suivi de personne se traduit par la minimisation de la différence entre la coordonnée x du barycentre et la coordonnée du centre de l'image. Cette minimisation est mise en oeuvre par une activation des moteurs. Le test vise à recueillir ces coordonnées en utilisant la bibliothèque husky, spécialement conçue pour la caméra, qui intègre un algorithme de reconnaissance de personnes ainsi que plusieurs fonctions, c. L'exécution du test se fait dans le fichier `detecthusky.py`, où la fonction principale, `get_xy`, renvoie les coordonnées souhaitées.

Il est important de noter que les distances sont mesurées en pixels et que, pour le moment, nous ne connaissons pas les dimensions de l'image complète ni la position de l'origine des coordonnées dans l'image. C'est pourquoi une calibration et une recherche approfondie de la bibliothèque seront nécessaires. Le test permet d'observer que la caméra renvoie des coordonnées cohérentes car elles varient avec le mouvement d'une personne dans l'image. Les fichiers `huskylib.py` et `huskytest.py` intègrent de nombreuses fonctions que nous n'avons pas utilisées.

4.2.4 Test du Servomoteur

Le servomoteur est le moteur qui se charge de déplacer la caméra verticalement, ce qui peut être utile en cas de perte de vue de l'objectif de suivi. Comme l'ordre de mouvement est effectué via la Raspberry, qui communique avec l'Arduino Uno, et que c'est ce dernier qui déplace le servomoteur, nous avons décidé de réaliser deux tests. Un premier directement en C++, `test_servo.ino`, qui garantit que la carte Arduino est capable de déplacer le moteur. Un deuxième test en Python, `test_servo.py`, qui assure que la communication entre la Raspberry, l'Arduino et le moteur est correcte et que nous sommes donc capables de déplacer le moteur via un programme lancé sur la Raspberry.

4.2.5 Test du Gyroscope

Le gyroscope permet d'obtenir l'angle de rotation sur l'axe z du robot par rapport à un angle initial. Cette information peut être très intéressante pour éviter les obstacles et localiser le robot dans l'espace. Le fonctionnement de ce capteur diffère de celui des autres, car sa communication avec la Raspberry est gérée par la carte Arduino Nano et non l'Arduino Uno. Suivant la même idée que pour le servomoteur, nous avons réalisé deux tests différents.

Test direct en C++ Le premier, dans le fichier `test_gyro.ino`, nous permet d'obtenir l'angle de rotation du robot directement via la carte Arduino Nano. Ce fichier utilise des fonctions propres au modèle du gyroscope utilisé via la bibliothèque `MPU6050_light` que nous avons dû installer dans l'IDE Arduino, et de la bibliothèque `Wire` (qui était déjà incorporée).

Test en Python Le deuxième test, et le plus important, permet de garantir l'obtention correcte de l'angle de rotation via la Raspberry Pi. Il se trouve dans le fichier `test_gyro.py`. Tout d'abord, nous nous connectons à la carte Arduino NANO à l'aide de la fonction `connect_to_arduino`, à laquelle nous envoyons "nano" comme entrée. Ensuite, nous appelons la fonction `get_gyro`, qui nous renvoie l'angle de rotation via l'ordre `READ_GYRO`. Ce test inclut une deuxième fonction, `reset_gyro`, qui n'a pas été implémentée mais qui sera probablement nécessaire pour le bon fonctionnement du robot. Cette deuxième fonction permet de réinitialiser l'angle de référence du robot, c'est-à-dire de remettre l'angle à zéro. Enfin, soulignons l'importance de l'installation de la bibliothèque `MPU6050_light` dans l'IDE Arduino, car sans elle, le gyroscope ne fonctionnera pas correctement.

5 Modes d'emplois

5.1 Contrôle à distance de la Raspberry

Dans cette partie nous allons vous apprendre quelque chose de primordial sur ce travail : le contrôle à distance. Il est très utile car en contrôlant la Raspberry à distance, vous pourrez lancer des codes sur le robot à distance depuis votre ordinateur, et vous pourrez aussi transférer des fichiers de votre ordinateur à la raspberry. Nous mettrons aussi dans le dossier de rendu "le document unique", une version plus détaillée de ce que nous allons vous présenter maintenant qui explique aussi comment faire si ce n'est pas assez clair ici. Pour faire cela vous aurez besoin de l'application RealVNC Viewer sur votre ordinateur, l'application est déjà installée sur la raspberry.

Récupération de l'adresse IP de la Raspberry

Lorsque vous ouvrez RealVNC Viewer vous allez avant tout devoir créer un compte et vous y connecter, notez bien vos id et mot de passe.

Une fois que vous êtes connectés vous devriez avoir accès à l'Accueil VNC Viewer, une interface semblable à celle de la Figure 13 : Faites un clic droit, puis dans le menu déroulant cliquez sur "nouvelle connexion". L'adresse IP de la Raspberry vous est demandé : Les prochains paragraphes vous expliquent comment avoir l'adresse IP de Votre Raspberry Pi.

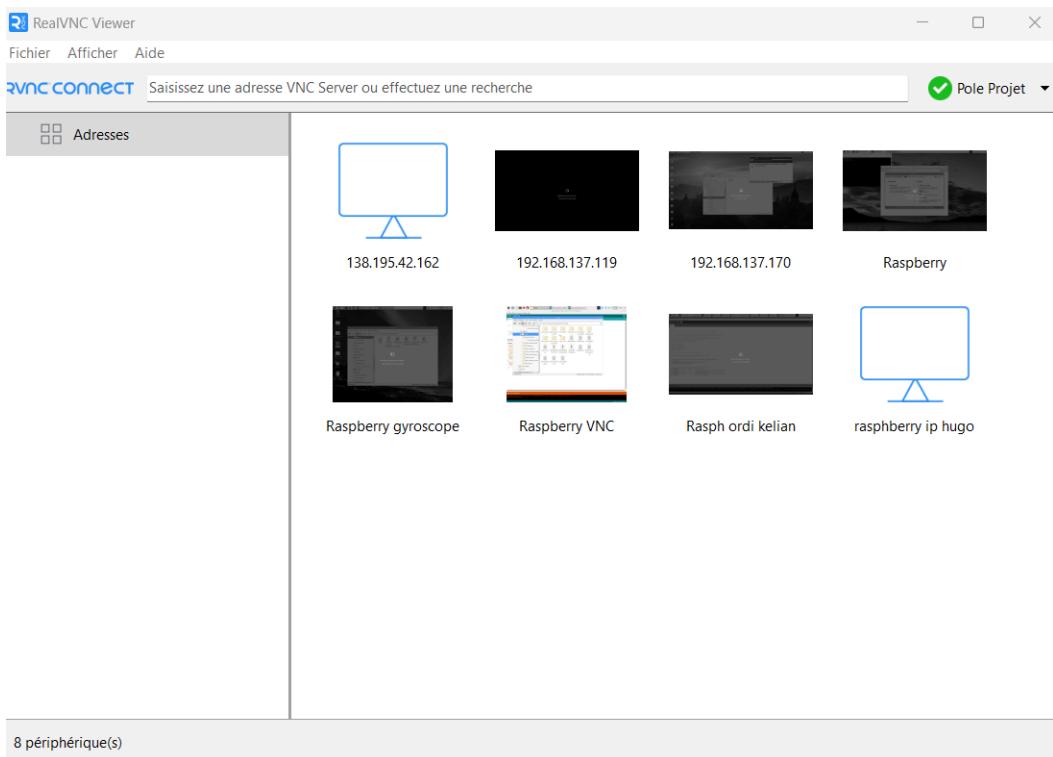


FIGURE 13 – Accueil VNC Viewer

Quelque chose d'important à savoir est que l'adresse IP de la Raspberry change en fonction du wifi auquel la Raspberry est connectée.

Dans les faits pour la connexion à distance il faudra toujours connecter la raspberry à un point d'accès généré par votre ordi (aller dans paramètre internet et activer point d'accès sans fil mobile). Nous allons voir 2 méthodes pour contrôler la Raspberry pour se connecter à ce point d'accès et récupérer l'adresse IP afin de pouvoir faire la connexion à distance. Lisez les deux méthodes !

Bien sûr pour des travaux ne nécessitant pas de travail à distance (coder sur la raspberry), vous pouvez juste vous connecter à la raspberry et travailler dessus sans utiliser de point d'accès.

Par cable VNC :

Cette méthode est pratique car elle requiert juste un ordi avec port VNC, un câble VNC et le robot.

1. Brancher l'ordi et la raspberry
2. Créer la nouvelle connexion et mettre l'IP suivant (on parle d'IP VNC) 169.254.125.234.
3. Double-cliquer dessus et rentrer l'identifiant : pi et le mot de passe : raspberry. (Attention à ne pas confondre ce login (celui du compte VNC Server) et le login au compte RealVNC Viewer !)

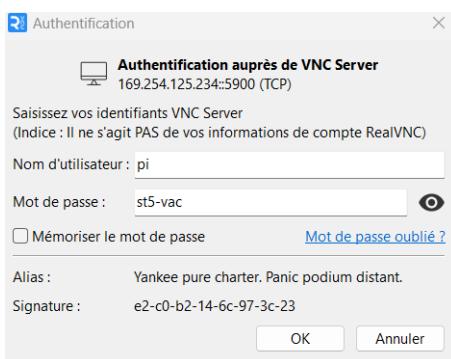


FIGURE 14 – Accueil RealVNC Viewer

Maintenant que vous êtes sur l'écran de la raspberry, il suffit de vous connecter à votre réseau comme sur un vrai ordinateur. Une fois que cela est fait, passez votre souris sur l'icone wifi et vous pourrez lire l'adresse IP de votre Raspberry.

Utiliser directement la Raspberry (nécessite un écran et une souris) :

Pour cette méthode il suffit de brancher un écran et une souris à la Raspberry et encore une fois, de la connecter au point d'accès mobile et de récupérer l'adresse IP comme indiqué ci haut dans le cas de la méthode VNC.

Maintenant que vous avez récupéré l'adresse IP de la Raspberry connectée à votre point d'accès mobile, vous pouvez établir la connection à distance. Pour ce faire, il suffit de reprendre ce qu'il y a écrit plus haut dans la partie "Récupération de l'adresse IP de la Raspberry" : Il faut créer une nouvelle connexion sur l'interface de Real VNC Viewer (Figure 13) en entrant cette fois, l'adresse IP que vous venez de récupérer, voilà tout ! Vous contrôlez désormais la Raspberry à distance.

5.2 Premiers tests à faire pour prendre en main la Arduino UNO et l'interface Arduino

Une fois que vous avez réussi à vous connecter à la Raspberry, il est temps de commencer à faire fonctionner la carte Arduino UNO. Pour ce faire, ouvrez l'Arduino IDE (version 1.8.19) (petite framboise (= logo Raspberry) en haut à gauche / Programmation / Arduino IDE). Commencez par faire clignoter une LED pour vérifier que la Raspberry marche toujours (Fichier/Exemples/Basics/Blink). Vérifiez que les paramètres sont les bons pour téléverser (= cliquer sur la flèche en haut à gauche) le code sur la carte Arduino. Il y a trois choses à vérifier dans l'onglet Outils : Type de Carte doit être sur Arduino Uno. Le port doit être "/dev/ttyACM0 (Arduino Uno)". Le Programmateur doit être "AVRISP mkII". Une fois que c'est bon cliquez sur téléverser. Si la LED clignote, passez à la suite

5.3 Connexions Arduino - Raspberry

Nous allons débuter par examiner le protocole de connexion entre l'Arduino et la Raspberry. Comprendre cette composante est essentiel pour l'ensemble du travail. Il est crucial de noter que notre robot est équipé d'une Raspberry qui établit des communications avec une carte Arduino Uno et une carte Arduino Nano.

Nous allons maintenant tester individuellement chaque code pour nous assurer de leur bon fonctionnement. Dans cette étape, nous testons le code `connexion.py` situé dans le répertoire `Robot2024/connexion.py`.

5.3.1 Pour la Arduino Uno

Avant d'exécuter le code, veuillez téléverser le code `main_arduino.ino` situé dans le répertoire `Robot2024/main_arduino/` en veillant à utiliser les paramètres appropriés. Le type de carte doit être défini sur "Arduino Uno", le port doit être configuré sur "/dev/ttyACM0 (Arduino Uno)", et le programmateur doit être sélectionné comme "AVRISP mkII".

5.3.2 Pour la Arduino Nano

Avant d'exécuter le code, assurez-vous de téléverser le code `main_nano.ino` situé dans le répertoire `Robot2024/main_nano/` avec les paramètres appropriés. Le type de carte doit être réglé sur "Arduino Nano", le port doit être configuré sur "/dev/ttyUSB0", le processeur doit être défini sur "ATmega328P (Old Bootloader)", et le programmateur doit être sélectionné comme "AVRISP mkII". Il est essentiel de s'assurer que tous ces paramètres sont corrects, sinon le code ne fonctionnera pas comme prévu.

5.3.3 Pour la Raspberry

Ouvrez le code avec le logiciel Thonny. Comme expliqué précédemment, la carte peut se connecter à la carte Arduino Uno ainsi qu'à la carte Arduino Nano. On peut se connecter aux deux cartes en simultané. Pour se connecter à la carte Arduino Uno décommentez la ligne correspondante. Essayez de vous connecter aux deux Arduino séparément et simultanément. Si tout fonctionne, après quelques tentatives de connections, vous devriez voir apparaître la phrase "connexion successful". Si c'est le cas passez à la suite.

5.4 Test composants

5.4.1 Test moteurs

Passons maintenant aux tests concrets des composants. Le premier test à effectuer est le test des moteurs, qui est également le plus simple. Ce test ne concerne pas la carte Arduino Nano, car ce sont la carte Arduino Uno et le PCB (le support de circuits imprimés noirs, qui est en fait aussi une carte Arduino Romeo BLE) qui actionnent les moteurs. Deux des moteurs sont connectés à la carte PCB, et deux autres à la Arduino Uno. En cas de problème avec les moteurs, cela pourrait également être lié à la batterie, en particulier si les moteurs qui ne fonctionnent pas (ou mal) sont ceux branchés sur le PCB. La batterie est fixée sous le robot.

Il est nécessaire de téléverser à nouveau le fichier `main_arduino.ino`, en veillant à utiliser les bons paramètres, comme décrit dans la section [5.2](#) intitulée "Premiers tests à faire pour prendre en main la Arduino UNO et l'interface Arduino". Allumez ensuite la carte PCB (les LEDs doivent s'allumer, l'interrupteur se trouve vers l'arrière du robot). Ensuite, exécutez le fichier `test_moteur.py` (situé dans `Robot2024/test_moteur.py`). Décommentez les lignes 24 et 25 (la fonction `move` doit être exécutée) et commentez la ligne 30 (la fonction `stop` sert à arrêter les moteurs).

Vous pouvez modifier les valeurs des 4 `write_i8` dans la fonction `move` (elles peuvent aller de -100 à 100) pour vérifier si tous les moteurs tournent correctement dans les deux sens. Il est recommandé de tester avec les valeurs 100, 0 et -100, car le couple résistant peut parfois empêcher la roue de tourner si les valeurs données sont trop faibles. Si cette étape fonctionne, vous pouvez passer à la suite.

5.4.2 Test capteur Infrarouge

Cette section est dédiée au capteur infrarouge, qui est également géré par la carte Arduino Uno.

Il est nécessaire de téléverser une fois de plus le fichier `main_arduino.ino`, en vous assurant d'utiliser les bons paramètres (cf section "Premiers tests à faire pour prendre en main la Arduino Uno et l'interface Arduino"). Ensuite, exécutez le fichier `test_infrarouge.py` (situé dans `Robot2024/test_infrarouge.py`).

Normalement, vous allez observer plusieurs erreurs de structure qui s'afficheront dans la console (c'est normal et probablement dû à un problème de synchronisation d'horloges). De temps en temps, vous devriez également voir un nombre compris entre 10 et 80 s'afficher. Ce nombre représente la distance que calcule le capteur infrarouge entre lui et l'obstacle. Vous pouvez simuler un obstacle en plaçant votre main devant le capteur. Si tout fonctionne correctement, plus votre main s'éloigne, plus les nombres devraient augmenter.

Si cette étape fonctionne, vous pouvez passer à la suite.

5.4.3 Test Servo

Cette section est dédié au servomoteur de la caméra, qui est aussi géré par la carte Arduino Uno

Cette fois-ci, il y a deux tests à effectuer.

Le premier teste le servomoteur directement avec le code téléchargé sur la carte Arduino Uno. Il n'y a donc pas de fichier python impliqué dans ce test. Il faut donc téléverser le fichier `test_servo.ino` (`Robot2024/test_servo/test_servo.ino`). Une fois ce code téléchargé, tu peux t'amuser à changer l'angle (en degrés) dans la commande `motor.write`, pour changer

l'angle de la caméra. A chaque fois que tu changes l'angle, il faut téléverser le code à nouveau (l'arduino exécute en boucle la loop du dernier code téléchargé).

Si tu as réussi à faire bouger l'angle de la caméra, tu peux passer au test suivant.

Pour le second test, il faut encore une fois téléverser le fichier `main_arduino.ino` en s'assurant d'avoir les bons paramètres (cf section [5.2 "Premiers tests à faire pour prendre en main la Arduino Uno et l'interface Arduino"](#)).

Ensuite il faut exécuter le fichier `test_servo.py`(Robot2024/test_servo.py).

Il faut ensuite exécuter le fichier en changeant l'angle dans la fonction main. Normalement la caméra devrait bouger. Si ce n'est pas le cas, passer à la section suivante.

5.4.4 Test caméra huskylib

Cette section est dédiée à l'utilisation de la caméra HuskyLens qui utilise sa propre librairie `huskylib` (si vous avez besoin de documentation, n'hésitez pas à aller voir la page [github dédiée ici](#)).

Il suffit d'exécuter le code `detecthusky.py` dont le chemin d'accès est `Robot2024/detecthusky.py`. Cet algorithme renvoie le barycentre de l'objet qui est reconnu. Vous pouvez tester qu'il fonctionne bien en vous mettant face à la caméra. En vous déplaçant légèrement (les coordonnées renvoyées changent en fonction de votre déplacement).

5.4.5 Test gyro

Le test qui nous a pris le plus de temps. Le gyroscope est géré par la carte Arduino Nano. Il y a encore une fois deux tests.

Le premier est contenu dans le fichier `test_gyro.ino` de chemin d'accès

`Robot2024/test_gyro/test_gyro.ino`. Il suffit de téléverser ce code sur la Arduino Nano en s'assurant d'avoir les bons paramètres activés (cf section "Connexions Arduino - Raspberry" / "Pour la Arduino Nano"). Le gyroscope a besoin que la bibliothèque `MPU6050light121` soit installée pour fonctionner. Ca devrait déjà être le cas sur la Raspberry fournie, mais si ce n'est pas le cas, il suffit d'aller dans Croquis/Importer une bibliothèque/Gérer les bibliothèques et de chercher la bibliothèque `MPU6050_light`. Pour cela néanmoins, il faut avoir une version de l'IDE Arduino ≥ 1.8 . Ensuite il faut voir ce qui est affiché dans le fichier Serial (on peut y accéder avec le raccourci clavier Ctrl + Shift + M ou en allant dans Outils/Moniteur série). Si quand vous tournez le robot, les chiffres changent, passez à la suite.

Pour le deuxième test, il faut téléverser le programme

`main_nano.ino` (du dossier `Robot2024/main_nano/ main_nano.ino`) sur la carte Arduino Nano. Ensuite exécutez le programme `test_gyro.py` (du dossier `Robot2024/test_gyro.py`) en commandant la ligne 38 qui appelle la fonction `reset_gyro` (cette fonction qui devrait redéfinir le zéro du gyroscope à l'angle courant n'est pas encore fonctionnelle). Si quand vous tournez le robot, les chiffres changent, vous avez réussi à effectuer tous les tests, ce qui signifie que tous les composants fonctionnent ! Super !

5.5 Si l'un des tests ne marche pas

Si un test ne fonctionne pas, vérifiez bien que tous les bons paramètres sont activés. Cela peut causer bien des erreurs. Vérifiez aussi que les logiciels que vous utilisez ont la bonne version et que la carte SD que vous utilisez est celle qui vous a été fournie. En effet, nous avons eu des problèmes lorsque nous avons essayé d'utiliser une autre carte SD, plus rapide, mais qui posait des problèmes d'OS. Il faut aussi vérifier que les baudrate sont bien les mêmes dans le code Arduino et dans le code python. De plus, si tu te sers du Moniteur Série pour afficher des informations directement en Arduino, le baudrate doit aussi être le bon (tu peux le régler en bas

à droite). Sinon tu peux toujours me contacter à l'adresse mail : hugo.debosschere@student-cs.fr et j'essaierai de répondre le plus vite possible et au meilleur de mon habilité.

Conclusion et perspective

Pour Conclure nous avons eu beaucoup de mal à démarrer car nous avons du retester tous les composants et retrouver les bons paramètres en passant par un long processus de trial and error.

Voici les choses que vous devez faire, et dans l'ordre que vous devez les faire, maintenant que vous avez fini de lire consciencieusement le rapport :

1. Réaliser tous les tests
2. Calibrer tous les capteurs (tester/découvrir les marges d'erreurs des capteurs / les incertitudes)
3. Intégrer les différents capteurs dans une fonction python unique qui permet de tous les appeler. Vous pourrez pour cela vous servir de la fonction main_suivi.py (Robot2024/main_suivi.py) en tant que base.
4. Implémenter le suivi de personnes en autonomie.
5. Implémenter l'évitement des objets pour poursuivre le suivi