

Chargeur de tôle motorisé

Reprise d'un projet :

Non

Partenaire :

Section CRCI du lycée

Étudiants chargés du projet :

Étudiant 1

Étudiant 2

Étudiant 3

Professeurs ou tuteurs responsables :

Pierre CLAVEL

Florent CHRÉTIEN

Patrick BONNAL

Sommaire :

Partie commune / page 2

Partie individuelle Théo MAURICE / page 11

Partie individuelle Mewen TREUSSART / page 35

Partie individuelle Iannis Carlin / page 57

Annexe / page 67

Expression du besoin :

Contexte

Le lycée Grandmont, situé à Tours, abrite plusieurs sections de Brevet de Technicien Supérieur (BTS) dans le domaine industriel, notamment la section Conception et Réalisation en Chaudronnerie Industrielle (CRCI). Cette formation prépare les étudiants à concevoir, fabriquer, maintenir et commercialiser des ouvrages en structure métallique. Les ateliers de cette section sont équipés de machines et d'équipements spécialisés, et les étudiants sont régulièrement amenés à manipuler et déplacer de grandes pièces de métal, notamment des tôles.

Problématique

L'année précédente, un groupe d'étudiants en BTS CRCI a conçu un chargeur de tôle pour transporter des tôles depuis le "Rack à tôles" jusqu'à la découpeuse laser située au fond de l'atelier. Bien que fonctionnel, ce chargeur présente plusieurs inconvénients :

- Déplacements manuels laborieux et peu ergonomiques.
- Problèmes de rigidité de la structure.
- Difficultés à positionner les tôles avec précision sur la découpeuse laser.

Matériel

Ce projet à besoin de beaucoup de matériels. La partie branchement et électrique du système a été fait en commun.

Le système est composé de :

- 4 roues
- 4 moteurs de roues
- 4 moteurs de directions des roues
- 1 raspberry pi 4
- 2 cartes ESP32 rev3 devkit
- 2 carte Arduino avec Motorshield
- 1 batterie
- 1 carte Arduino nano
- 1 interrupteur
- 1 répéteur
- 1 convertisseur 36-12V
- Divers câbles

Schéma du système électrique :

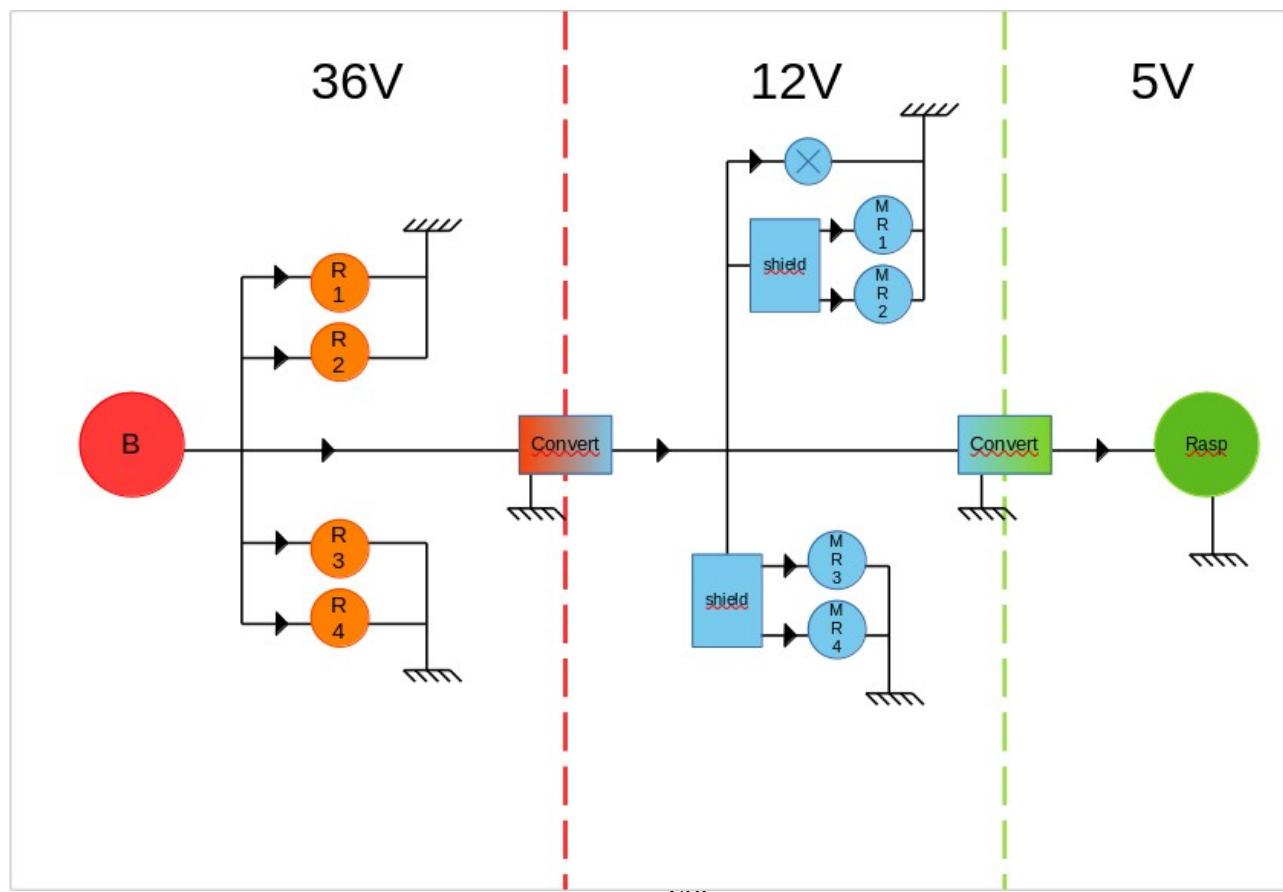
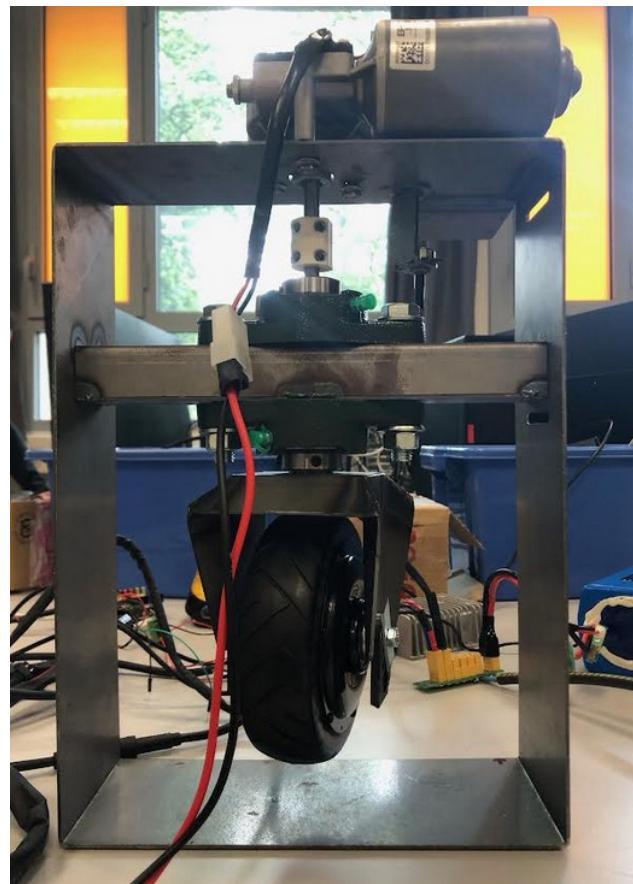
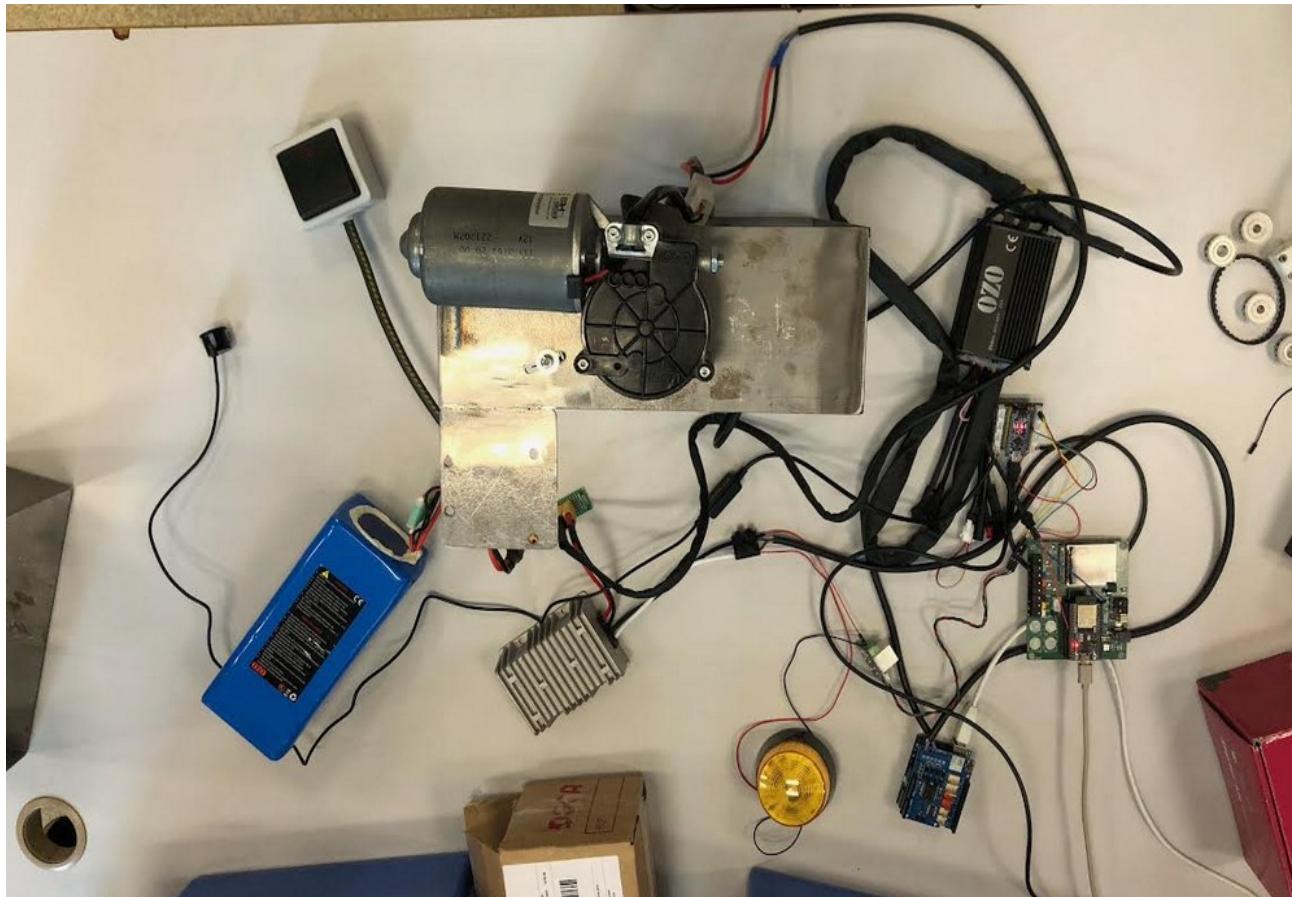


Photo du système :



Analyse UML:

Cas d'utilisation

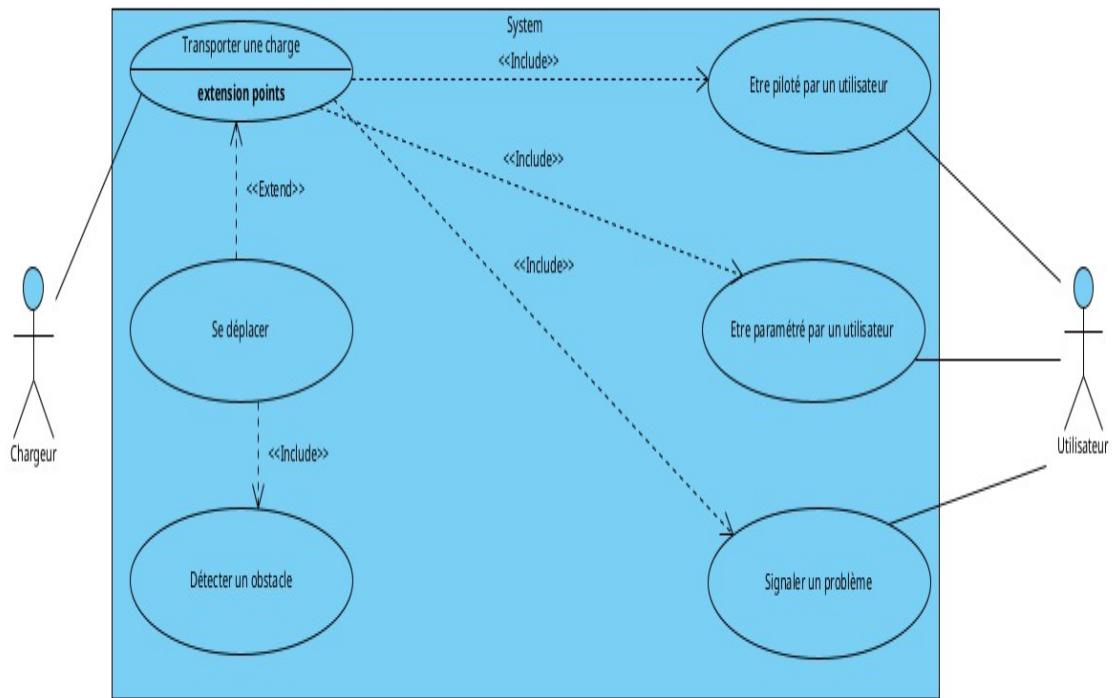


Diagramme d'exigence

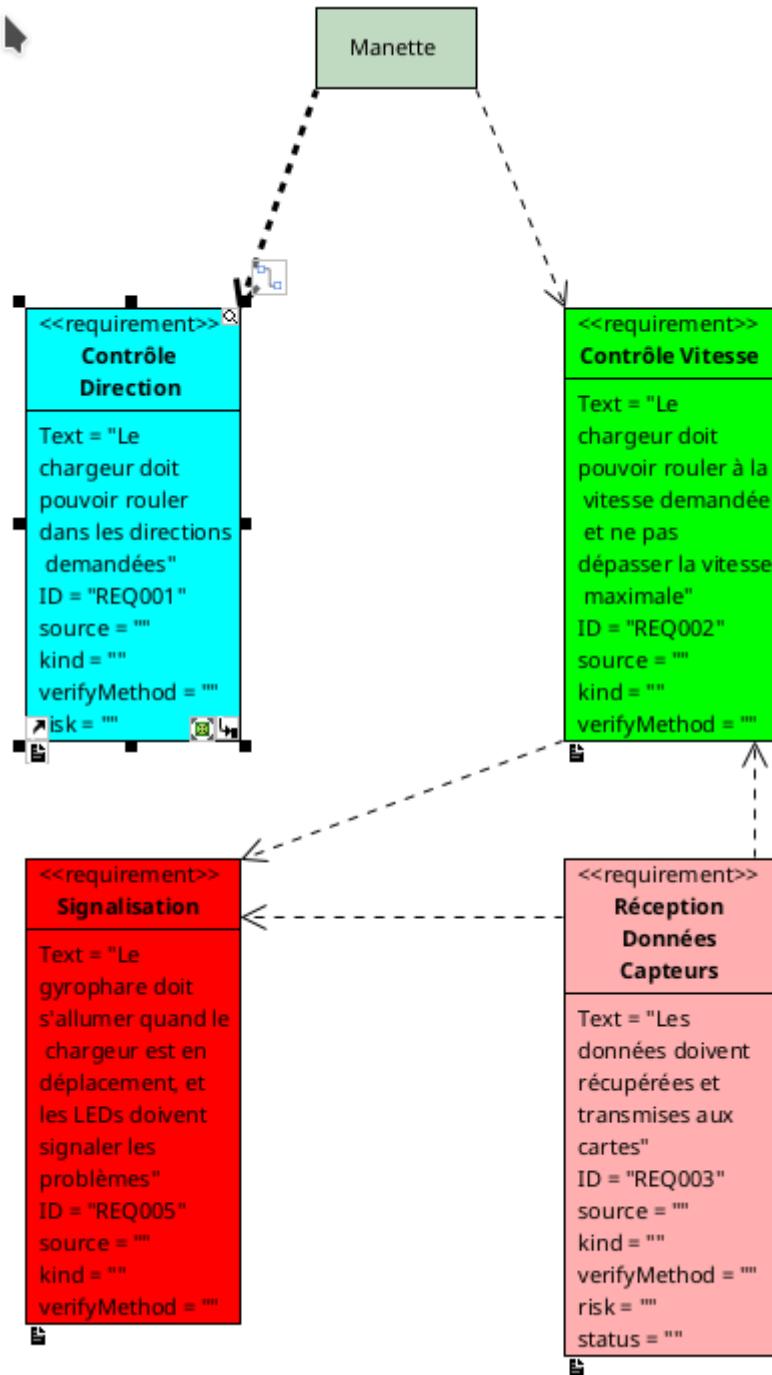


Diagramme de séquence Carte capteur

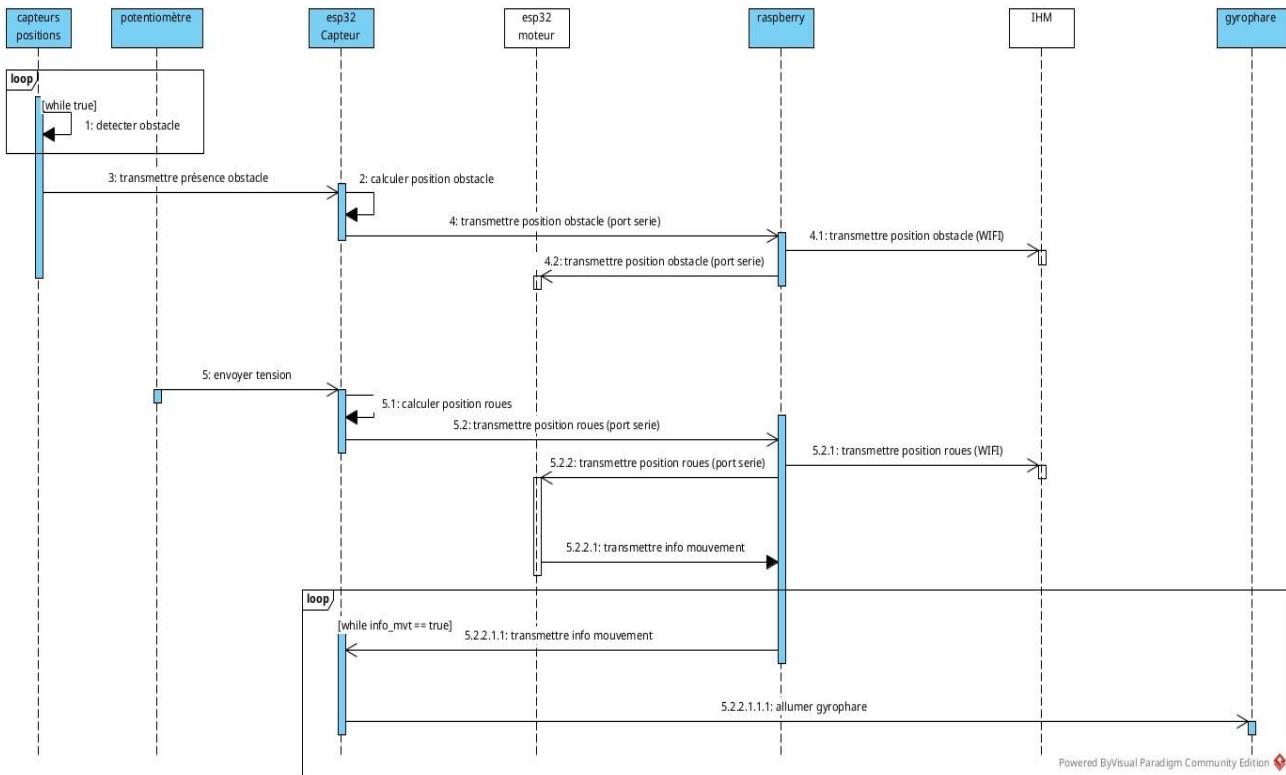


Diagramme de séquence Application gestion + capteur

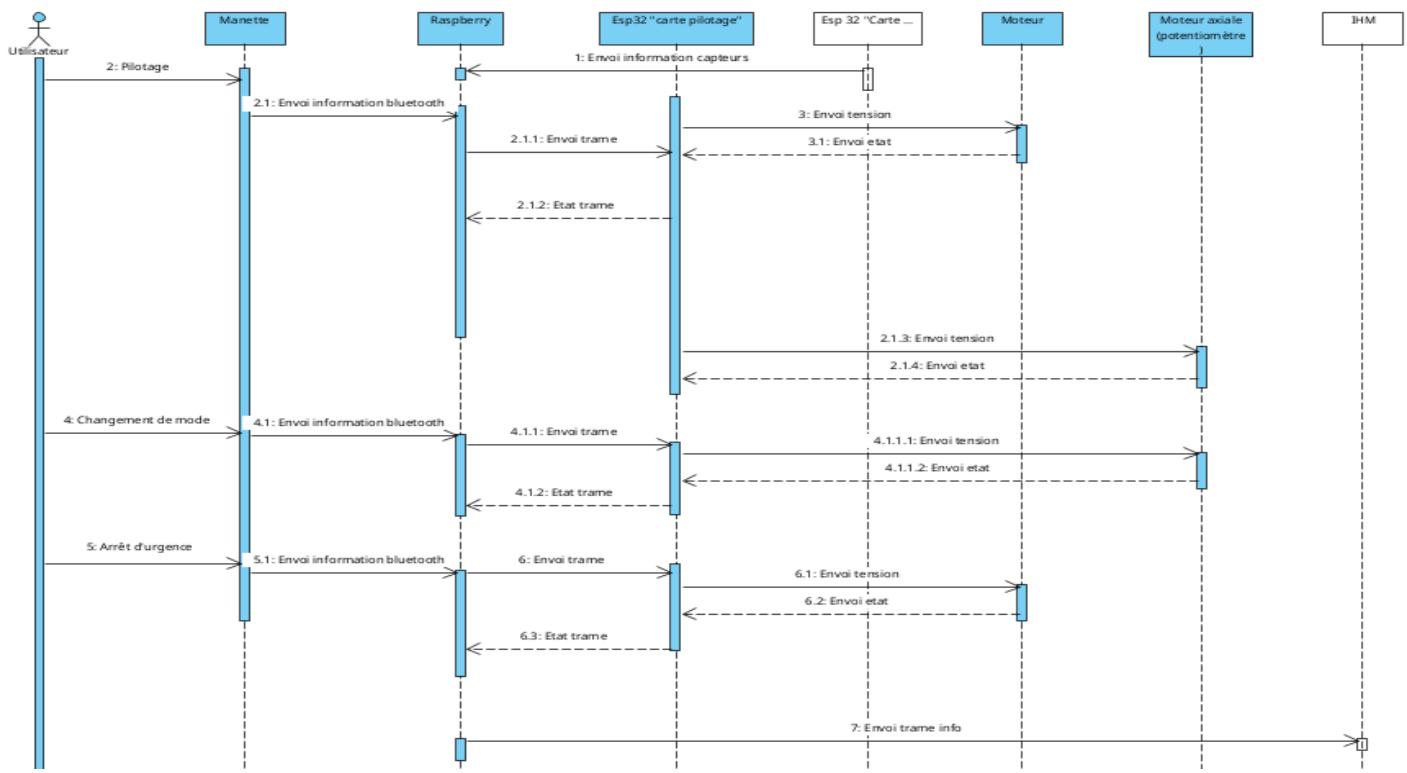


Diagramme de séquence IHM

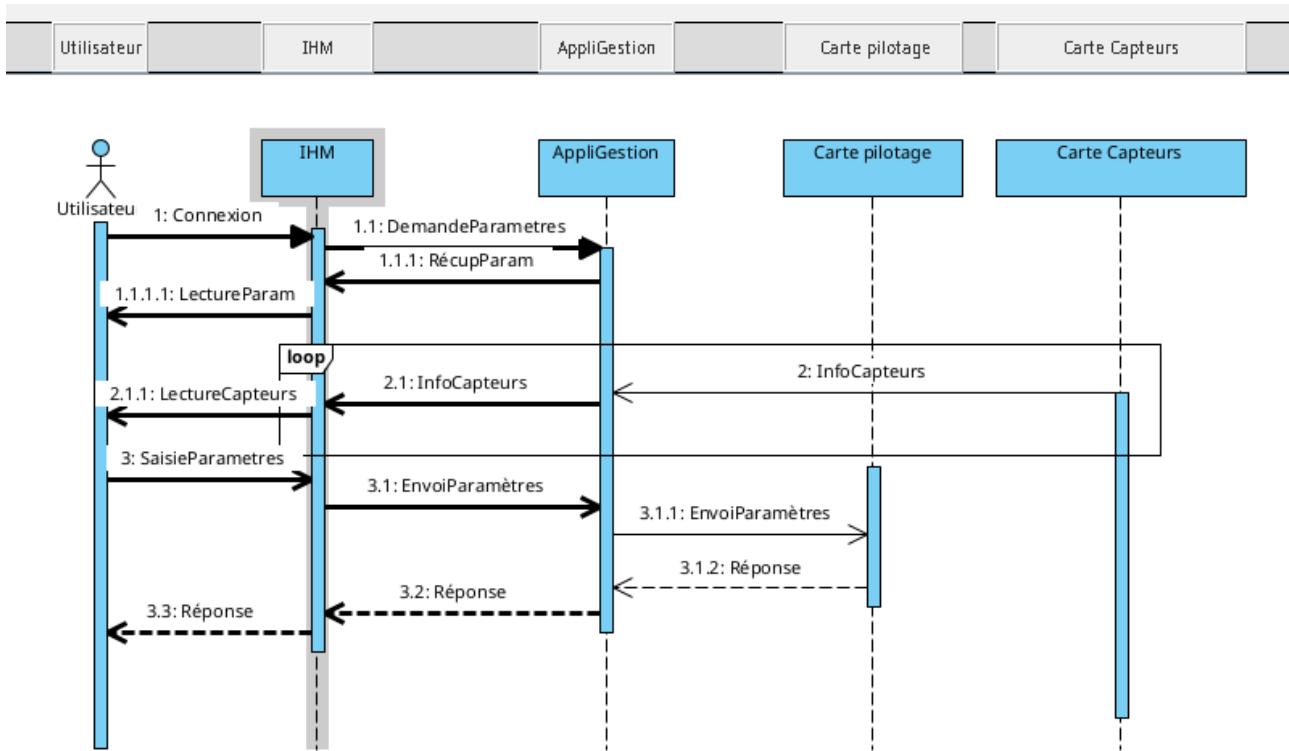
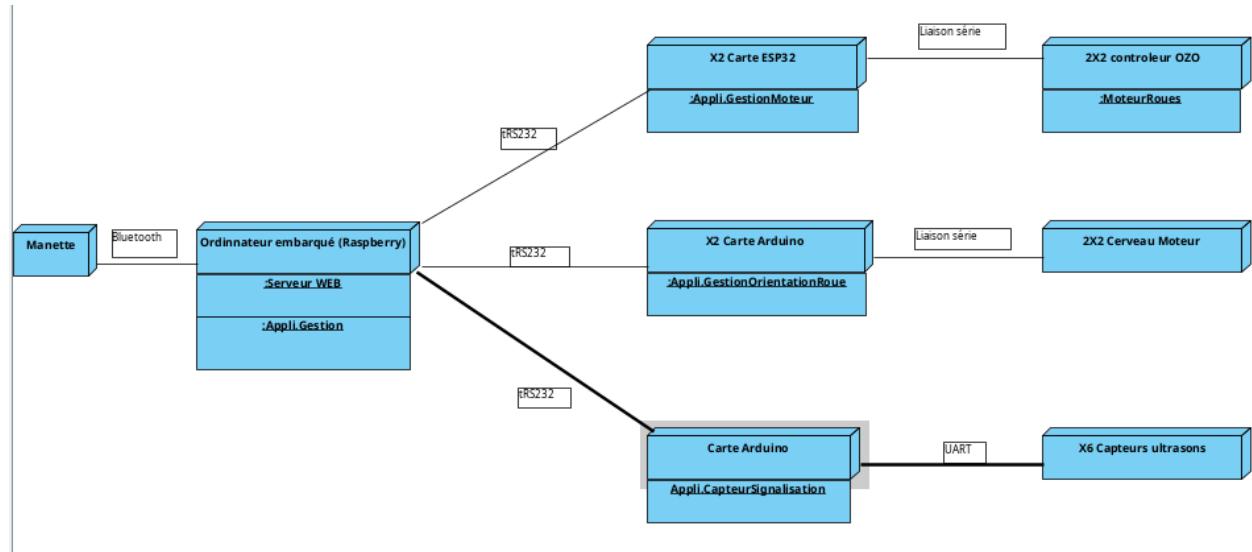


Diagramme de déploiement



Répartition des tâches et planning :

Iannis CARLIN	Mewen TREUSSART	Théo MAURICE
Carte capteur signalisation	Application IHM	Application Gestion
Application Capteur signalisation	Application Gestion Configuration micro ordinateur	Carte pilotage moteur Application pilotage moteur

Diagramme de Gantt prévisionnel

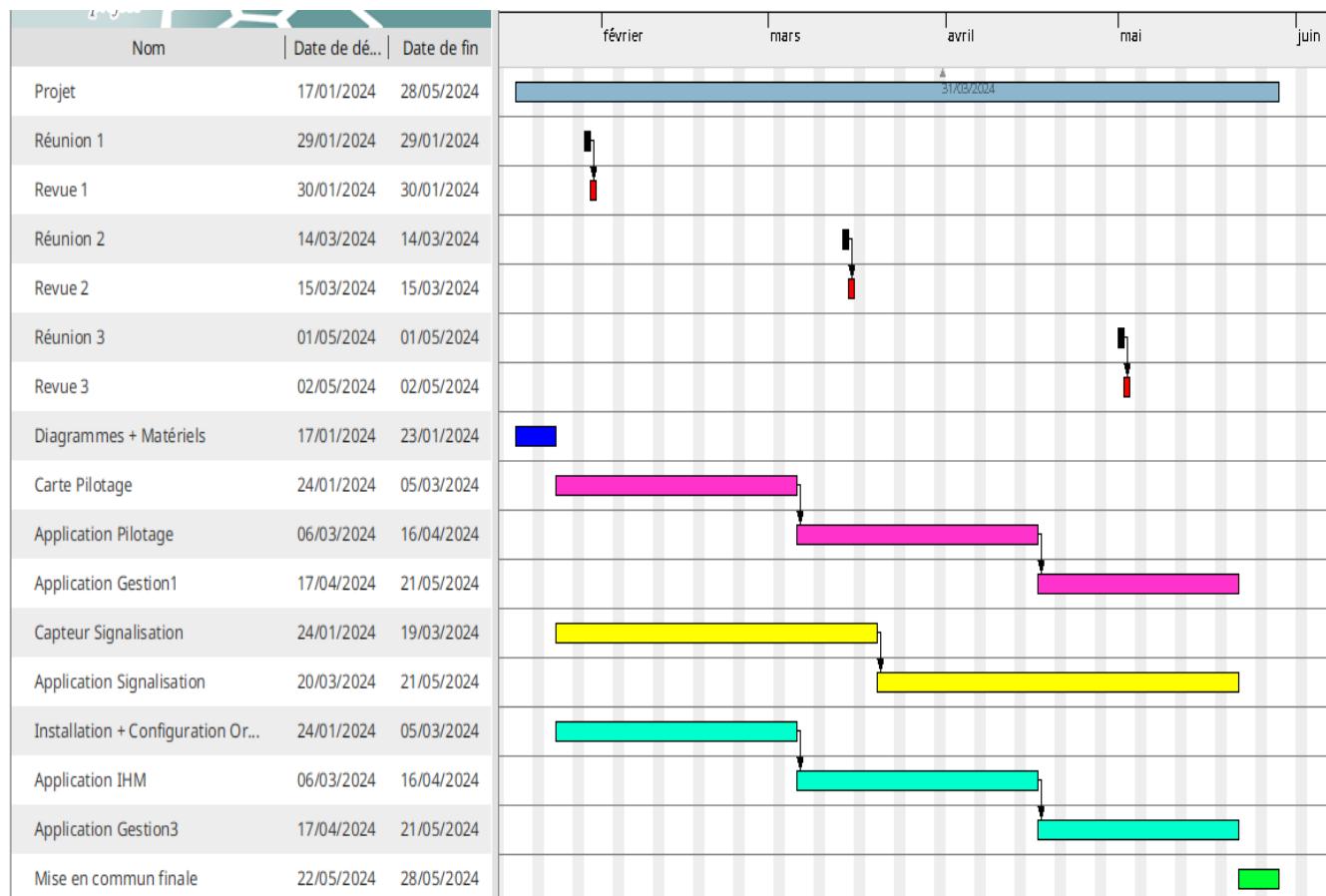
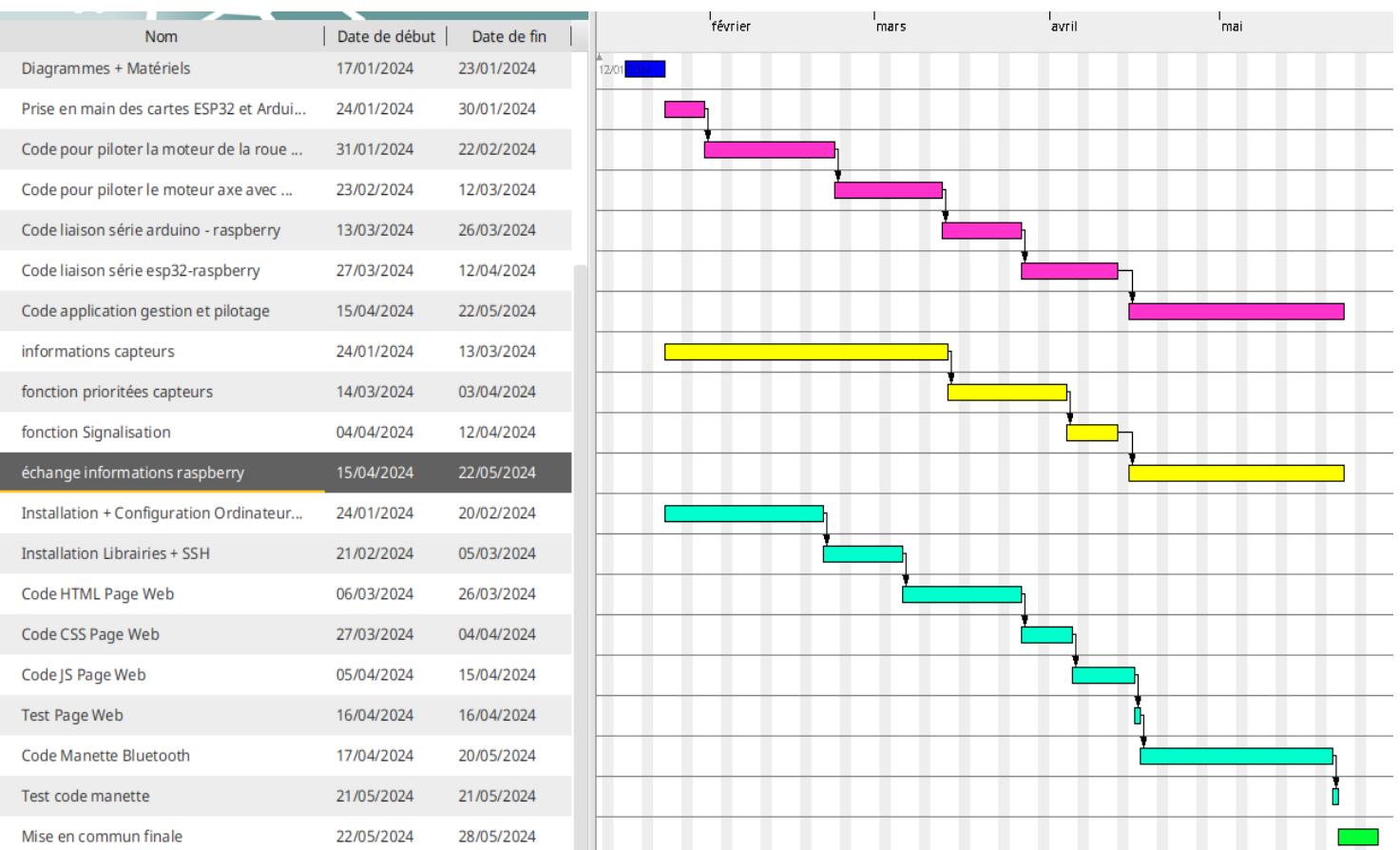


Diagramme de Gantt final



Partie individuelle : Théo MAURICE

Introduction :

Dans le cadre de mon projet en BTS Systèmes Numériques et Informatique et Réseaux (SNIR) au lycée Grandmont, j'ai eu l'opportunité de participer à un projet interdisciplinaire visant à concevoir et développer un chargeur de tôle motorisé pour la section Conception et Réalisation en Chaudronnerie Industrielle (CRCI). En effet L'année précédente, un groupe d'étudiants de BTS CRCI avait conçu un chargeur de tôle manuel pour transporter des tôles du "Rack à tôles" vers la "Découpeuse laser". Bien que fonctionnel, ce dispositif présentait plusieurs limitations, notamment des déplacements manuels peu pratiques, des problèmes de rigidité structurelle et des difficultés pour positionner les tôles de manière précise.

Pour améliorer ce système, il a été décidé de créer un chargeur de tôle motorisé et partiellement autonome, capable de suivre un parcours tracé au sol, de se positionner avec précision et de manipuler les tôles de manière plus efficace et sécurisée. Mon rôle dans ce projet était centré sur le développement de l'application « Gestion » et la carte « Pilotage Moteurs ».

Ma contribution au projet s'est articulée autour de trois axes principaux :

1. Application « Gestion » :

- Compléter les caractéristiques et les performances de l'application.
- Valider le fonctionnement envisagé par des tests de faisabilité.
- Développer et tester l'application.

Important : Une partie du code de cette application (communication avec IHM) sera développée par l'étudiant Mewen Treussart

2. Carte « Pilotage Moteur » :

- Sélectionner et assembler les éléments matériels.
- Valider les caractéristiques des composants.
- Effectuer les tests et les mesures permettant de valider le fonctionnement.

3. Application « Pilotage Moteurs » :

- Compléter les caractéristiques et le cahier des charges de l'application.
- Valider le fonctionnement envisagé, par des tests de faisabilité.
- Développer et tester l'application.

De plus, il m'a été demandé de produire la documentation nécessaire pour permettre à un utilisateur d'installer, configurer et utiliser l'ensemble des outils, ainsi que de rédiger les documents de recette.

A. Solutions multiples et envisagées :

1. Application « Gestion » :

Hardware : Pour héberger l'application gestion nous avions besoin d'un ordinateur embarqué. Pour pouvoir dupliquer ou faire évoluer le système le matériel devait être facile à trouver et si possible libre de droit. Le choix de la Raspberry Pi 4 pour l'ordinateur embarqué s'est imposé comme la solution la plus adaptée. Les raisons de ce choix sont les suivantes :

1. Performance et Flexibilité :

- Puissance de traitement : Le processeur quad-core de la Raspberry Pi 4 et jusqu'à 8 Go de RAM offrent une puissance de traitement suffisante pour gérer des applications exigeantes, incluant des interfaces graphiques complexes et la gestion de données en temps réel.

- Capacité Multitâche : Permet de gérer simultanément plusieurs processus, assurant une fluidité et une réactivité accrues pour l'application de gestion.

2. Connectivité et Interfaces :

- Wifi et Bluetooth : La connectivité sans fil intégrée facilite la communication avec les autres composants du système, notamment la manette.
- Interfaces multiples : Les ports USB, HDMI, et GPIO de la Raspberry Pi 4 offrent une grande flexibilité pour connecter des périphériques supplémentaires et des capteurs, et pour étendre les fonctionnalités futures.

3. Environnement de développement :

- Système d'exploitation et Outils : La possibilité d'utiliser Raspbian, un système d'exploitation basé sur Linux, permet de bénéficier d'un large éventail d'outils de développement et de bibliothèques. Cela facilite la programmation, le débogage et la maintenance de l'application.
- Communauté et Support : La large communauté de la Raspberry Pi offre un accès à une multitude de ressources, de guides et de support technique, réduisant le temps de développement et facilitant la résolution des problèmes.



Software : Pour le langage de développement de l'application Gestion j'ai d'abord voulu coder en C++ dans un souci d'uniformité puisque le code sur toutes les autres carte est du C++. Malheureusement le développement en C++ c'est révéler trop complexe sur Raspberry pi et avec trop peu de ressource. J'ai donc préféré partir sur le langage python qui est nativement supporté par la Raspberry Pi avec une large communauté qui développe et maintient des projets sur cette plateforme ce qui m'a facilité la tâche pour trouver des informations sur ce que je voulais faire.

2. Cartes et applications « Pilotages Moteurs » :

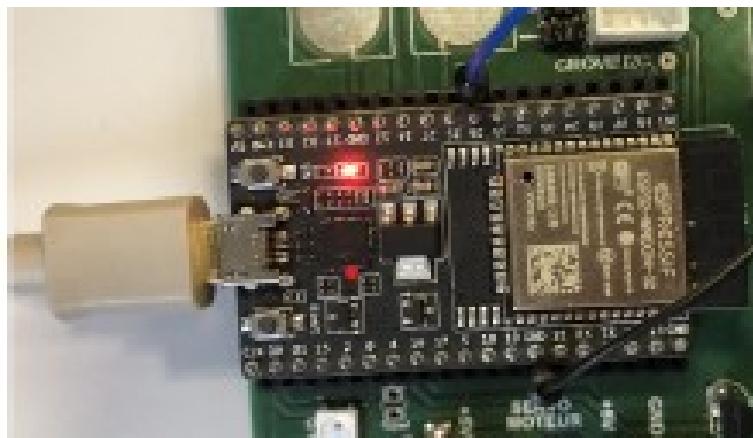
Hardware : Pour piloter les différents moteurs des roues (moteurs de directions des roues et moteur des roues) nous avions besoin de plateformes de pilotage qui répondraient aux exigences de performances, de flexibilité et de facilité d'intégration. Après avoir envisagé plusieurs options, et en fonction du matériel déjà disponible, les choix se sont portés sur une carte ESP32 rev3 devkit pour le pilotage des moteurs des roues et une carte Arduino avec Motorshield pour le moteur de direction des roues. Les raisons de ces choix sont les suivantes :

ESP32 rev3 devkit :

- Performance et connectivité : Le choix de l'ESP32 pour le pilotage des moteurs des roues a été motivé par sa capacité à gérer des tâches complexes en temps réel grâce à son processeur double cœur.

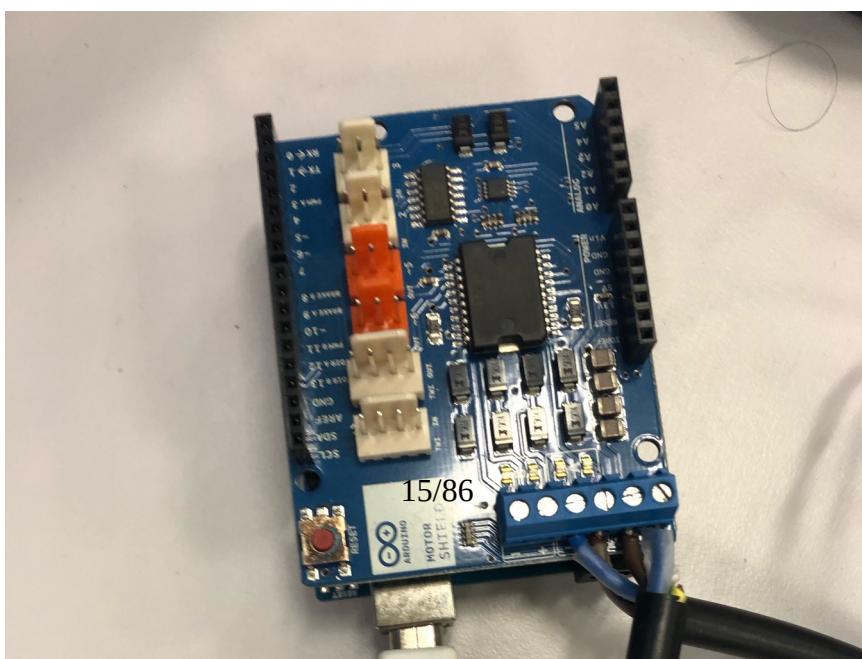
- Polyvalence : L'ESP32 rev 3 devkit offre de nombreuses broches GPIO (notamment le 25 et 26 qui permettent l'envoi d'une tension), permettant de connecter plusieurs moteurs et capteurs supplémentaires si nécessaire. Sa flexibilité en fait un choix idéal pour des systèmes évolutifs et modifiables.

- Faible coût.
- Nous avions déjà un ESP32 de disponible j'ai donc pu directement travailler dessus.



II. Arduino avec Motorshield :

- Simplicité d'utilisation : L'Arduino, associé au Motorshield, offre une solution simple et rapide à mettre en œuvre pour le contrôle du moteur de direction des roues. Le Motorshield permet un branchement direct des moteurs et un contrôle facile, réduisant ainsi le temps de développement et les risques d'erreurs de câblage. Sur le motorshield on a 2 sorties pour contrôler 2 moteurs.
- Fiabilité et documentation : L'Arduino est une plateforme bien connue et éprouvée, avec une vaste documentation et une large communauté de développeurs. Cela facilite la résolution des problèmes et l'optimisation du code. De plus, le Motorshield fournit une interface fiable pour le contrôle des moteurs, adaptée aux besoins de ce projet.



Software : Pour ces applications j'ai choisi le langage C++ car c'est le langage de base des environnements de développement Arduino et ESP32. Pour développer j'ai choisi d'utiliser Visual Studio Code car je maîtrisais cet éditeur de code qui permet de mener des projets Arduino ou ESP32.

B. Conception, configuration, réalisation :

I. Carte et Application pilotage moteur de direction des roues

1. Connectique :

Pour piloter le moteur de direction des roues avec une carte Arduino et une carte motorshield il faut d'abord que la carte motorshield soit correctement installé sur la carte arduino.

Il faut ensuite que la carte motorshield soit alimenté avec une tension de 12V.

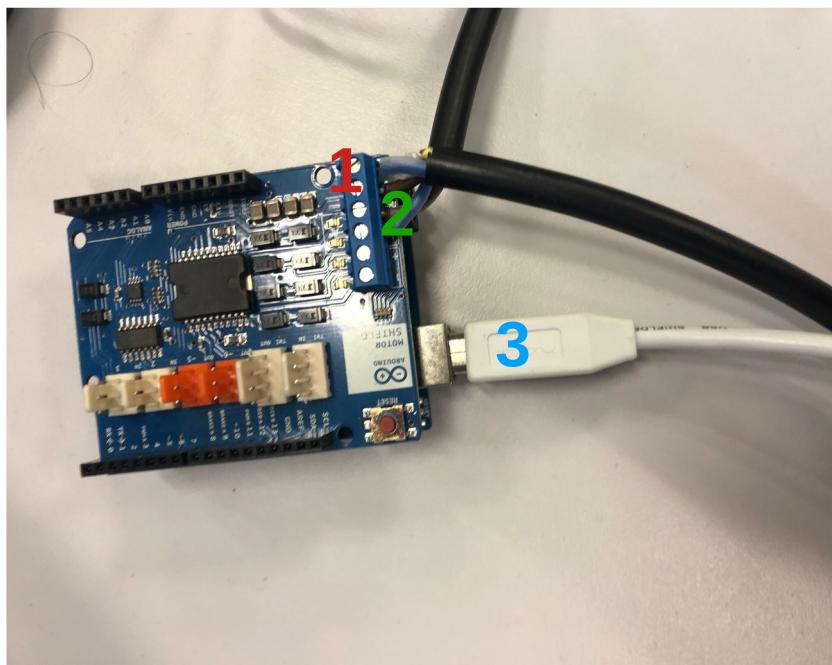
De plus il faut que les sorties de tensions de la carte motorshield soient connectées aux moteurs de directions des roues.

Il faut aussi que la carte Arduino soit capable de communiquer avec la raspberry pi. Pour cela j'ai choisi la liaison série avec un câble USB.

1 : Câble d'alimentation de la carte Motorshield.

2 : Sortie de tension

3 : Liaison série USB



2. Code :

La carte doit être capable de piloter le moteur des roues en fonction de l'information envoyée par la raspberry.

Pour contrôler le moteur de direction des roues il faut d'abord savoir quels sont les pins qui sont utilisés par la carte motorshield. Ces pins sont les suivants :

- Sortie A :

- Direction = 12
- PWM = 3
- Brake = 9

Sortie B :

- Direction = 13
- PWM = 11
- Brake = 8

J'ai donc choisi dans un souci de simplicité de définir des variables pour ces pins.

Extrait de code pour la sortie A :

```
int directionPin = 12;  
int pwmPin = 3;  
int brakePin = 9;
```

Une autre variable que j'ai défini est celle qui stocke l'information de la direction envoyer par la raspberry.

```
int direction_roue = 0;
```

Cette variable va changer de valeur en fonction de si la raspberry dit qu'il faut tourner à droite ou à gauche.

Il faut ensuite démarrer la communication entre la raspberry et l'arduino tout en activant les pins en mode « output ». La communication se fait à 9600 bauds car c'est ce qui est conseillé pour l'arduino.

Extrait de code pour la sortie A :

```
void setup()  
{  
    Serial.begin(9600); // Démarre la communication série à 9600 bauds  
    pinMode(directionPin, OUTPUT);  
    pinMode(pwmPin, OUTPUT);  
    pinMode(brakePin, OUTPUT);  
}
```

Pour contrôler le moteur il faut donc utiliser ces 3 pins qui contrôles la direction, la tension envoyée et le frein.

Pour contrôler la direction et le frein on utilise la fonction **digitalWrite(pin, value)** qui prend en paramètre le pin que l'on souhaite contrôler et la valeur que l'on veut envoyer (HIGH ou LOW).

Si le pin de frein reçoit HIGH alors le frein est activé, s'il reçoit LOW alors le frein est désactivé.

Si le pin de direction reçoit HIGH alors le moteur tournera à droite, s'il reçoit LOW il tournera à gauche.

Pour la tension envoyer qui va faire tourner le moteur il faut utiliser la fonction **analogWrite(pin, value)** qui prend en paramètre le pin que l'on souhaite contrôler et la valeur que l'on veut envoyer.

Les fonctions analogWrite et digitalWrite sont présentes dans la librairie Arduino de base.

Ces deux fonctions m'ont permis de coder 3 fonctions : une qui fait tourner le moteur à droite, une à gauche et une qui actionne le frein.

```
void tourner_droite()
{
    digitalWrite(directionPin, LOW);
    digitalWrite(brakePin, LOW);
    analogWrite(pwmPin, 255);
}

void tourner_gauche()
{
    digitalWrite(directionPin, HIGH);
    digitalWrite(brakePin, LOW);
    analogWrite(pwmPin, 255);
}

void frein()
{
    digitalWrite(brakePin, HIGH);
    analogWrite(pwmPin, 0);
}
```

Il faut pour savoir quand le programme doit utiliser ses fonctions que la variable direction roue soit modifiée en fonction de ce que la raspberry envoie.

Pour cela j'ai utilisé la librairie Serial présente de base dans un projet Arduino.

Cette librairie donne accès aux fonctions Serial.available() qui permet de savoir si des données sont envoyées par le port série, Serial.read() qui permet de lire les données envoyées et Serial.println() qui permet d'envoyer des informations par le port série.

Grace à cette librairie j'ai pu coder une fonction recevoir qui test s'il y a des données envoyées pour pouvoir stocker ces données dans la variable de direction des roues et qui ensuite envoie à la raspberry la variable de direction des roues pour savoir si l'arduino a correctement reçu les données de la raspberry.

```
void recevoir()
{
    if (Serial.available() > 0)
    {
        direction_roue = Serial.read();

        Serial.println(direction_roue);

    }
    delay(10);
}
```

Maintenant que la variable de direction des roues est changée en fonction de ce qu'envoie la raspberry j'ai pu coder une fonction qui en fonction de l'information reçue tourne à droite, à gauche ou freine.

```

void output()
{
    if (direction_roue == 49)
    {
        tourner_droite();
    }
    else if (direction_roue == 50)
    {
        tourner_gauche();
    }
    else
    {
        frein();
    }
    delay(40);
    direction_roue = 0;
}

```

Pour finir il faut que le programme tourne en boucle sur l'arduino j'ai alors placé mon code dans la fonction loop().

```

void loop()
{
    recevoir();
    output();
}

```

Conclusion : Contrôler le moteur de direction des roues a été assez simple car la carte motorshield est très bien documenté sur le site officiel d'Arduino.

II. Carte pilotage et Application moteur des roues

1. Connectique :

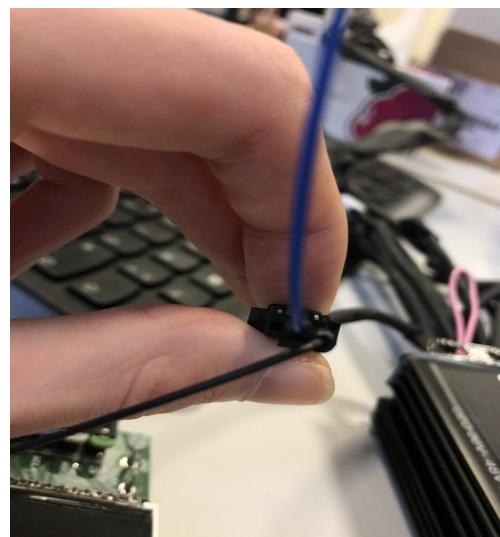
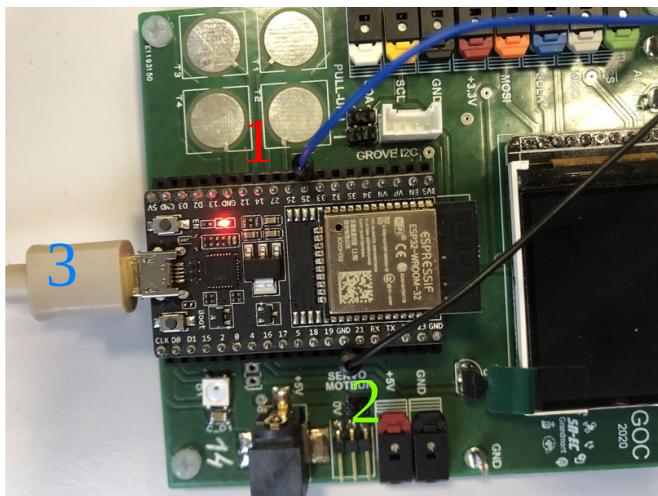
Pour piloter le moteur des roues avec une carte ESP32 rev3 devkit il faut que la carte soit alimentée via un câble USB-micro USB branché sur la raspberry, ce câble sert aussi de liaison série.

De plus il faut que le moteur soit connecter au pin qui envoie la tension (pin 25 ou 26) et à la masse (pin GND).

1 : Câble connecté au GPIO25

2 : Câble connecté à la masse

3 : Câble USB connecté à la raspberry



2. Code :

La carte doit être capable de recevoir les informations de la raspberry (si le moteur doit être activé et si la vitesse doit diminuer ou augmenter) et piloter le moteur en fonction de ces informations. Comme la réception et l'envoie doivent être faits en simultanée il est nécessaire de faire du multitâche à l'aide de thread.

Il faut définir les différents paramètres requis pour permettre à la liaison série et au pin de fonctionner. Pour cela j'ai utilisé les librairies driver/uart.h et driver/gpio.h

```
#define TEST_TXD (GPIO_NUM_1)
#define TEST_RXD (GPIO_NUM_3)
#define TEST_RTS (UART_PIN_NO_CHANGE)
#define TEST_CTS (UART_PIN_NO_CHANGE)

#define UART_PORT_NUM (0)
#define UART_BAUD_RATE (115200)
#define TASK_STACK_SIZE (2048)
#define BUF_SIZE (1024)
```

J'ai défini 2 variables dans mon code : une qui permet de savoir si le moteur doit être activé ou pas, une qui contrôle la vitesse.

```
bool etat_roue = false;
int vitesse = 120;
```

J'ai ensuite codé une fonction changement_vitesse() qui permet de changer la variable vitesse en fonction de l'information reçue dans la trame.

```
void vitesse_roue(uint8_t *trame)
{
    if (trame[1] == '1' && vitesse <= 253)
    {
        vitesse += 2;
    }
    else if (trame[1] == '2' && vitesse >= 122)
    {
        vitesse -= 2;
    }
}
```

Pour que la variable etat_roue change il faut créer une tache qui reçoit les informations de la liaison série. Dans mon code la trame qu'envoie la raspberry est stockée dans une variable data. Si le caractère de cette trame qui correspond au moteur des roues est 1 alors le moteur doit être en marche alors que s'il est de 0 il doit être arrêté. Ensuite j'envoie les informations stockées dans data à la raspberry pour vérifier que le transfert de données c'est bien déroulé.

```

static void tache_recevoir()
{
    uart_config_t uart_config =
    {
        .baud_rate = UART_BAUD_RATE,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .source_clk = UART_SCLK_DEFAULT,
    };
    int intr_alloc_flags = 0;

#if CONFIG_UART_ISR_IN_IRAM
    intr_alloc_flags = ESP_INTR_FLAG_IRAM;
#endif

    ESP_ERROR_CHECK(uart_driver_install(UART_PORT_NUM, BUF_SIZE * 2, 0, 0, N
    ESP_ERROR_CHECK(uart_param_config(UART_PORT_NUM, &uart_config));
    ESP_ERROR_CHECK(uart_set_pin(UART_PORT_NUM, TEST_TXD, TEST_RXD, TEST_RTS

    // Configure un buffer pour les données qui arrivent
    uint8_t *data = (uint8_t *) malloc(BUF_SIZE);

    while (1)
    {
        // Lis les données UART
        int len = uart_read_bytes(UART_PORT_NUM, data, (BUF_SIZE - 1), 20 /
        // Change l'état des roues en fonction du message reçu
        if(data[0] == '1')
        {
            etat_roue = true;
        }
        else
        {
            etat_roue = false;
        }
        // Renvoie les données UART
        uart_write_bytes(UART_PORT_NUM, (const char *) data, len);
        vTaskDelay(10);
    }
}

```

Maintenant que la variable `etat_roue` est changée en fonction des données de la raspberry j'ai pu coder les tâches qui vont s'occuper de faire fonctionner les moteurs des roues.

Grace à la fonction **`dac_output_voltage`**(pin, value) qui envoie une tension entre 0V et 5V (dans cette fonction la valeur est comprise entre 0 et 255 avec 0 = 0V et 255 = 5V) dans le pin voulu.

Si ma variable `etat_roue` est vrai alors j'envoie dans les pins la valeur de ma variable vitesse sinon j'envoie 0.

Une fois mes

```
static void tache_output_roue1()
{
    while(1)
    {
        if (etat_roue)
        {
            dac_output_voltage(DAC_CHAN_0, vitesse);
        }
        else
        {
            dac_output_voltage(DAC_CHAN_0, 0);
        }
        vTaskDelay(10);
    }
}

static void tache_output_roue2()
{
    while(1)
    {
        if (etat_roue)
        {
            dac_output_voltage(DAC_CHAN_1, vitesse);
        }
        else
        {
            dac_output_voltage(DAC_CHAN_1, 0);
        }
        vTaskDelay(10);
    }
}
```

différentes tâches codées j'ai pu les intégrer dans le main.

Dans le main j'ai d'abord activé les pins qui envoient la tension et ensuite créer mes threads.

```
void app_main(void)
{
    dac_output_enable(DAC_CHAN_0);
    dac_output_enable(DAC_CHAN_1);
    xTaskCreate(tache_recevoir, "tache_recevoir", TASK_STACK_SIZE, NULL, 10, NULL);
    xTaskCreate(tache_output_roue1, "tache_output_roue1", 2048, NULL, 10, NULL);
    xTaskCreate(tache_output_roue2, "tache_output_roue2", 2048, NULL, 10, NULL);
}
```

III. Application Gestion

L'application gestion est l'application qui fait le lien entre toutes les cartes du système.

1. Connectique :

L'application Gestion se trouve sur la raspberry. Nous avons choisi de faire communiquer les différentes cartes avec la raspberry par liaison série USB.

La raspberry doit être connectée au 5 cartes (2 cartes moteurs roues, 2 cartes moteurs directions roues et la carte capteur) via câble USB.

2. Code :

Pour communiquer avec les différentes cartes j'ai utilisé la librairie serial. Avec cette librairie j'ai créé différentes variables ser qui stockent le port et le baudrate où doit être envoyé l'information.

```
import serial
ser=serial.Serial(port='/dev/ttyACM0',baudrate=9600, timeout=0.05)
ser2=serial.Serial(port='/dev/ttyUSB0',baudrate=115200, timeout=0.1)
ser_capt=serial.Serial(port='/dev/ttyUSB1',baudrate=115200, timeout=0.1)
```

Ensuite grâce à mes variables ser j'ai utilisé la fonction write(données) qui permet d'envoyer les informations via la liaison série.

```
command=str(y)
command2=str(y)
ser.write(command.encode('utf-8'))
ser2.write(command2.encode('utf-8'))
```

J'ai ensuite créé des variables line qui stockent les informations envoyées par les différentes cartes du système.

```
line=ser.readline().decode('utf-8').rstrip()
line2=ser2.readline().decode('utf-8').rstrip()
print(line)
print(line2)
```

Au moment au j'ai codé l'application Gestion les informations de la manette n'était pas encore récupéré. J'ai donc testé mon programme avec des entrées clavier. En fonction des entrées clavier les variables command changent. Ces variables sont ensuite envoyées via les ports séries.

```
1 import tkinter as tk
2 import serial
3 ser=serial.Serial(port='/dev/ttyACM0',baudrate=9600, timeout=0.2)
4 ser2=serial.Serial(port='/dev/ttyUSB1',baudrate=115200, timeout=0.2)
5 root=tk.Tk()
6 root.geometry("500x500+450+100")
7 canvas1=tk.Canvas(root, width=500, height=500)
8 |canvas1.pack()
9 |command='0'
10 while(True):
11     def keypressed(k):
12         x=0
13         y=0
14         global command
15         global command2
16         if k.keysym=='Up':
17             y=2
18         elif k.keysym=='Down':
19             y=1
20         elif k.keysym=='Left':
21             x=2
22         elif k.keysym=='Right':
23             x=1
24         else:
25             y=0
26         command=str(y)
27         command2=str(y)
28         ser.write(command.encode('utf-8'))
29         ser2.write(command2.encode('utf-8'))
30         line=ser.readline().decode('utf-8').rstrip()
31         line2=ser2.readline().decode('utf-8').rstrip()
32         print(line)
33         print(line2)
34     root.bind('<KeyPress>', keypressed)
```

C. Cahier de recette

Fiche de test

<input type="checkbox"/> Recette globale : d'un cas d'utilisation et/ou d'une fonctionnalité globale				
<input checked="" type="checkbox"/> Test de faisabilité				
<input type="checkbox"/> Test unitaire				
<input type="checkbox"/> Test d'intégration				
Nom de l'étudiant : MAURICE Théo				
Date : 08/02/2024				
Objectif du test : Vérifier le pilotage du moteur via ESP32				
Conditions de réalisation				
<ul style="list-style-type: none"> Besoins matériels <ul style="list-style-type: none"> Carte ESP32 Moteur Câble Câble USB Alimentation Ordinateur Besoins logiciels (nom et version) <ul style="list-style-type: none"> Visual Studio Code Noms des fichiers utilisés <ul style="list-style-type: none"> Tp_test_output_ESP32 				
Scénario				
ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status OK / NOK
1	Connecter le pc et l'ESP32 via câble USB		ESP32 connecté au pc	OK
2	Téléverser le programme Tp_test_output_ESP32 avec VS Code		Le programme est téléchargé dans l'ESP32	OK
3	Brancher les câbles sur GPIO25 et GND au moteur (voir annexe 1)		Les câbles sont branchés	OK
4	Brancher l'alimentation au moteur (voir annexe 2)		L'alimentation est branchée	OK
5	Observer le fonctionnement du moteur		Le moteur fonctionne	OK

Annexes (captures d'écran et schémas)



Annexe 1

Fiche de test

<p><input type="checkbox"/> Recette globale : d'un cas d'utilisation et/ou d'une fonctionnalité globale <input checked="" type="checkbox"/> Test de faisabilité <input type="checkbox"/> Test unitaire <input type="checkbox"/> Test d'intégration</p>				
<p>Nom de l'étudiant : MAURICE Théo Date : 27/03/2024 Objectif du test : Vérifier le contrôle de l'esp32 via des entrées clavier raspberry pi</p>				
<p>Conditions de réalisation</p> <ul style="list-style-type: none"> • Besoins matériels <ul style="list-style-type: none"> Carte esp32 Raspberry pi Câble USB Ordinateur Clavier • Besoins logiciels (nom et version) <ul style="list-style-type: none"> Visual Studio Code Thonny Python IDE • Noms des fichiers utilisés <ul style="list-style-type: none"> Tp_test_esp32_output_liaison_rasp Test_clavier 				
Scénario				
ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status OK / NOK
1	Connecter le pc et l'esp32 via câble USB		L'esp32 est connecté au pc	OK
2	Téléverser le programme TP_test_esp32_output_liaison_rasp		Le programme est téléchargé dans l'esp32	OK
3	Connecter l'esp32 à la raspberry via câble USB		L'esp32 est connecté à la raspberry	OK
4	Faire le montage du système (voir annexe 1)		Tous les composants sont alimentés	OK
5	Lancer le programme Test_clavier sur la raspberry avec Thonny Python IDE		Le programme se lance	OK
6	Tester avec les flèches directionnelles si la roue tourne		Le moteur des roues est contrôlé par les flèches du clavier	OK

Annexes (captures d'écran et schémas)

Fiche de test

<input type="checkbox"/> Recette globale : d'un cas d'utilisation et/ou d'une fonctionnalité globale				
<input checked="" type="checkbox"/> Test de faisabilité				
<input type="checkbox"/> Test unitaire				
<input type="checkbox"/> Test d'intégration				
Nom de l'étudiant : MAURICE Théo				
Date : 21/03/2024				
Objectif du test : Vérifier l'échange d'information entre l'Arduino et la raspberry				
Conditions de réalisation				
<ul style="list-style-type: none"> • Besoins matériels <ul style="list-style-type: none"> Carte Arduino Uno Raspberry pi Câble USB Ordinateur • Besoins logiciels (nom et version) <ul style="list-style-type: none"> Visual Studio Code Thonny Python IDE • Noms des fichiers utilisés <ul style="list-style-type: none"> Tp_test_arduino_liaison_serie Test_liaison_série_rasp 				
Scénario				
ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status OK / NOK
1	Connecter le pc et l'arduino via câble USB		L'arduino est connecté au pc	OK
2	Téléverser le programme TP_test_arduino_liaison_serie		Le programme est téléchargé dans l'arduino	OK
3	Connecter l'arduino à la raspberry via câble USB		L'arduino est connecté à la raspberry	OK
4	Lancer le programme Test_liaison_série_rasp sur la raspberry avec Thonny Python IDE		Le programme se lance	OK
5	Observer l'échange d'information entre les 2 cartes		Les informations s'échangent (voir Annexe 1)	OK

Annexes (captures d'écran et schémas)

Annexe 1

Commentaires Le test fonctionne parfaitement	
Approbation : Oui	Signature du chef

Fiche de test

- | |
|--|
| <input type="checkbox"/> Recette globale : d'un cas d'utilisation et/ou d'une fonctionnalité globale
<input checked="" type="checkbox"/> Test de faisabilité
<input type="checkbox"/> Test unitaire
<input type="checkbox"/> Test d'intégration |
|--|

Nom de l'étudiant : MAURICE Théo

Date : 08/02/2024

Objectif du test : Vérifier le pilotage du moteur direction via Arduino Uno et motorshield

Conditions de réalisation

- Besoins matériels
 - Carte Arduino Uno
 - Carte motor shield rev3
 - Moteur
 - Câble
 - Câble USB
 - Générateur de tension
 - Ordinateur
- Besoins logiciels (nom et version)
 - Visual Studio Code
- Noms des fichiers utilisés
 - Tp_test_motorshield

Scénario

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status OK / NOK
1	Connecter l'arduino uno et la carte motorshield		Les cartes sont connectées	OK
2	Connecter le pc et l'arduino uno via câble USB		ESP32 connecté au pc	OK
3	Téléverser le programme Tp_test_motorshield avec VS Code		Le programme est téléchargé dans l'ESP32	OK
4	Brancher la sortie A au moteur (voir annexe 1)		Les câbles sont branchés	OK
5	Brancher le générateur au motorshield et le régler sur 12V et moins d'1 A (voir annexe 1)		Le motorshield est branché	OK
6	Observer le fonctionnement du moteur		Le moteur fonctionne	OK

Annexes (captures d'écran et schémas)

Fiche de test

<input type="checkbox"/> Recette globale : d'un cas d'utilisation et/ou d'une fonctionnalité globale <input checked="" type="checkbox"/> Test de faisabilité <input type="checkbox"/> Test unitaire <input type="checkbox"/> Test d'intégration				
Nom de l'étudiant : MAURICE Théo				
Date : 27/03/2024				
Objectif du test : Vérifier le contrôle de l'arduino via des entrées clavier raspberry pi				
Conditions de réalisation				
<ul style="list-style-type: none"> • Besoins matériels <ul style="list-style-type: none"> Carte arduino Carte motorshield Raspberry pi Câble USB Ordinateur Clavier • Besoins logiciels (nom et version) <ul style="list-style-type: none"> Visual Studio Code Thonny Python IDE • Noms des fichiers utilisés <ul style="list-style-type: none"> Tp_test_arduino_output_liason_rasp Test_clavier 				
Scénario				
ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status OK / NOK
1	Connecter le pc et l'arduino via câble USB		L'arduino est connecté au pc	OK
2	Téléverser le programme TP_test_arduino_output_liaison_rasp		Le programme est téléchargé dans l'arduino	OK
3	Connecter l'arduino à la raspberry via câble USB		L'arduino est connecté à la raspberry	OK
4	Faire le montage du système (voir annexe 1)		Tous les composants sont alimentés	OK
5	Lancer le programme Test_clavier sur la raspberry avec Thonny Python IDE		Le programme se lance	OK
6	Tester avec les flèches directionnelles si le moteur axial tourne dans les deux directions		Le moteur axial des roues est contrôlé par les flèches du clavier	OK

Annexes (captures d'écran et schémas)

D. Problèmes rencontrés

La plus grande difficulté rencontrée lors de ce projet a été la communication avec le pôle CRCI qui avait pour mission de construire le chariot. La maquette sur laquelle on devait travailler lors de ce projet nous a été livré avec 2 mois de retard. De plus nous n'avons jamais eu le chariot complet.

J'ai eu des difficultés sur le code de la carte qui pilote le moteur des roues. Le sujet n'était pas très documenté, j'ai du cherché longuement avant de trouver une solution qui marchait.

E. Conclusion personnelle

Participer au développement du chargeur de tôle motorisé pour le lycée Grandmont a été une expérience enrichissante et formatrice. Ce projet interdisciplinaire m'a permis de mettre en pratique les compétences acquises durant ma formation en BTS Systèmes Numériques et Informatique et Réseaux, tout en travaillant en étroite collaboration avec la section Conception et Réalisation en Chaudronnerie Industrielle.

L'utilisation de l'ESP32 et de l'Arduino pour le contrôle des moteurs a mis en évidence l'importance de choisir des composants matériels adaptés et de les programmer de manière optimale pour répondre aux besoins spécifiques du projet.

En conclusion, ce projet a été une formidable opportunité de développer des compétences techniques, mais aussi de travailler en équipe et de comprendre les enjeux pratiques d'un projet technologique complexe.

Partie individuelle : Treussart Mewen

I°) Introduction

Le lycée Grandmont se distingue par ses multiples sections de BTS industriels, offrant des formations spécialisées qui répondent aux besoins du secteur. Parmi ces sections, le BTS CRCI (Conception et Réalisation en Chaudronnerie Industrielle) prépare les étudiants aux métiers techniques de la conception, fabrication, maintenance et commercialisation de structures métalliques. Les ateliers dédiés à ces formations se trouvent dans un bâtiment principalement destiné aux disciplines industrielles.

La formation CRCI implique fréquemment la manipulation et le déplacement de grandes pièces métalliques. L'année dernière, des étudiants de cette section ont conçu un chargeur de tôle pour transporter les tôles depuis le "Rack à tôles" jusqu'à la "Découpeuse laser". Bien que fonctionnel et actuellement en usage, ce chargeur présente plusieurs limitations : il nécessite des déplacements manuels, souffre de problèmes de rigidité et complique le positionnement des tôles.

Face à ces défis, un nouveau projet a été lancé, mobilisant les compétences des deux sections de BTS. L'objectif est de concevoir un chargeur de tôle motorisé, pilotable et partiellement autonome. Les étudiants de la section CRCI seront responsables de la conception et de la réalisation de la partie mécanique, incluant le châssis et toutes les pièces de fixation et de liaison. Parallèlement, les étudiants de la section SNIR (Systèmes Numériques, option Informatique et Réseaux) prendront en charge la partie électronique, qui devra permettre le pilotage des déplacements du chariot, ainsi que la gestion des échanges de données, des

capteurs et le contrôle des différents actionneurs (moteurs, vérins, avertisseurs, etc.).

Ce projet interdisciplinaire représente une opportunité enrichissante pour les étudiants des deux sections de mettre en pratique leurs compétences techniques et de collaborer à la réalisation d'un dispositif innovant et performant.

Dans le cadre de ce projet, ma contribution se focalise sur la conception des interfaces homme-machine, visant à garantir une interaction fluide et efficace pour l'utilisateur. Le chariot sera contrôlable via une télécommande sans fil, pensée pour être ergonomique, intuitive et facile à utiliser.



Cette télécommande offre plusieurs possibilités à savoir :

- Régler la vitesse du chargeur.
- Orienter les roues.
- Démarrer ou arrêter les déplacements.

En parallèle, une application web servira d'interface homme-machine (IHM) pour le pilotage du chariot. Cette application proposera plusieurs fonctionnalités :

- Pilotage du chariot : contrôle à distance du chariot par l'utilisateur.

- Modification des paramètres : ajustement de paramètres tels que la vitesse maximale.

Chaque fonctionnalité (pilotage, paramétrage) sera accessible à partir d'onglets distincts pour une utilisation claire et organisée. De plus, cette application, tout comme L'IHM physique à savoir la manette, devront communiquer avec l'application de Gestion pour assurer une intégration complète et une coordination optimale du système.

En parallèle de cette conception des IHM, j'ai également installé et configurer l'ordinateur embarqué du système, à savoir un Raspberry Pi 4, sur lequel j'ai dû installer l'OS mais également plusieurs librairies nécessaires à différentes parties du projet.

II°)Faisabilités

Le premier test a réalisé était celui de la manette, celle-ci étant déjà disponible au niveau de la section, suite à un projet des années précédentes. Les différents points à tester étaient la connectivité en bluetooth entre la manette et l'ordinateur embarqué, mais également la fonctionnalité de chaque bouton ainsi que les réponses envoyées lors de l'activation de ces derniers. Dans ce but, j'ai établi la procédure que vous pouvez voir sur la fiche de test qui suit.

Fiche de test

- Recette globale : d'un cas d'utilisation et/ou d'une fonctionnalité globale**
- Test de faisabilité**
- Test unitaire**
- Test d'intégration**

Nom de l'étudiant : Treussart Mewen

Date : 12/02/2024

Objectif du test : Vérifier la connectivité bluetooth de la manette sur l'ordinateur embarqué.

Conditions de réalisation

- Besoins matériels
 - Ordinateur embarqué (Raspberry 4)
 - Écran
 - Câble Ethernet RJ45
 - Câble HDMI
 - Manette Bluetooth
- Besoins logiciels (nom et version)
 - Navigateur web
 - Site web de test : <https://hardwaretester.com/gamepad>
- Noms des fichiers utilisés

Scénario

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status OK / NOK
1	Connecter l'ordinateur embarqué au réseau via le cable RJ 45, puis à un écran via le câble HDMI. Mettre sous tension l'ordinateur embarqué.		Branchements corrects, l'ordinateur embarqué se lance bien, et le bureau s'affiche correctement sur l'écran.	OK
2	Vérifier si le Bluetooth est activé, sinon l'activé via l'icône située dans l'angle supérieur gauche du bureau. (annexe 1)		Le Bluetooth est activé.	OK
3	Allumer la manette via le bouton central. (annexe 2)		La manette est allumée, un voyant s'allume et clignote.	OK
4	On connecte le PC à la manette via le menu Bluetooth. (annexe 3)		La manette est connectée, le voyant est allumé et ne clignote plus, la manette apparaît cochée dans le menu Bluetooth. (annexe 4)	OK

5	On lance le navigateur pour se rendre sur la page web : https://hardwaretester.com/gamepad		La page web s'affiche bien.	OK
6	On effectue des tests en appuyant sur les différents boutons, joysticks.		Les valeurs passent de 0 à 1 en fonction des pressions sur les boutons et des mouvements de joysticks. (annexe 5)	OK

Annexes (captures d'écran et schémas)



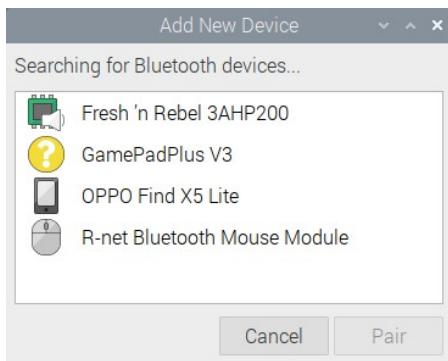
annexe 1



Voyants lumineux

Bouton central

annexe 2



annexe 3 4

annexe

GamePadPlus V3 (Vendor: 1949 Product: 0402)

INDEX	CONNECTED	MAPPING	TIMESTAMP									
1	Yes	n/a	49500.40000									
Pose	HapticActuators	Hand	DisplayId	Vibration								
n/a	n/a	n/a	n/a	n/a								
B0 1.00 B13 0.00	B1 0.00 B14 0.00	B2 0.00 B15 0.00	B3 0.00 B16 0.00	B4 1.00 B17 0.00	B5 0.00 B18 0.00	B6 0.00 B19 0.00	B7 0.00 B20 0.00	B8 0.00 B21 0.00	B9 0.00 B22 0.00	B10 0.00 B23 0.00	B11 0.00 B24 0.00	B12 0.00 B25 0.00
AXIS 0 -1.00000	AXIS 1 -0.59798	AXIS 2 0.00000	AXIS 3 0.00000	AXIS 4 0.00000	AXIS 5 0.00000	AXIS 6 0.00000	AXIS 7 0.00000					

annexe 5

Commentaires

Le test fonctionne parfaitement

Approbation : Oui

Signature du chef

Le test de faisabilité de ce modèle de manette ayant porté ses fruits, j'ai donc pris la décision de travailler avec celui-ci et non d'en commandé un autre comme cela a pu être envisagé lors des premières discussions autour du projet.

III°) Solutions multiples

Lors des premiers jours du projet, nous avons décidé de l'ordinateur embarqué que nous allions utilisés, et notamment sa version. En effet, le choix de s'orienter vers un Raspberry Pi était assez évident puisque nous avions déjà travaillé sur ce genre de machine lors de nos deux années, mais nous devions alors choisir la version que nous allions utilisé, on a donc comparé la version 3

et 4 de la machine, et opté pour la version 4 à cause des avantages suivants :

RAM : 4 Go contre 0,5 Go, soit 3,5 Go de plus.

Processeur : 28,57 % plus rapide avec une fréquence de 1,8 GHz contre 1,4 GHz.

Ports USB : 4 ports contre 1, soit 3 de plus.

Ports HDMI : 2 ports contre 1.

Version OpenGL ES : 3.1 contre 2, offrant une version plus récente.

Ports USB de Type-C : inclus dans le Raspberry Pi 4.

Version Bluetooth : 5 contre 4.2, une version plus récente.

Support Ethernet : disponible sur le Raspberry Pi 4.

Ici, les points clés sont la version du Bluetooth, la présence de 4 ports USB qui est intéressante dû au nombre de cartes qui seront branchées à l'ordinateur embarqué à terme, ainsi que le support Ethernet facilitant les tests lors de l'évolution du projet.

Pour la page Web, j'ai décidé de coder en HTML, JavaScript et CSS, puisqu'il s'agit des langages Web sur lesquels j'avais des bases, et qui me permettrait de faire ce qui était demandé dans le projet.

Pour le codage applicatif, donc le code de l'IHM manette et de l'application Gestion, j'ai décidé, de concert avec les autres membres de l'équipe, d'utiliser le Python. Ce choix a été orienté par la présence de librairies intéressantes notamment au niveau du traitement de la communication avec les autres cartes mais également au niveau de la gestion de la manette et de ses informations en Bluetooth. Un autre choix, un temps évoqué, était le C++ car c'est le langage sur lequel nous avons le plus travaillé lors de notre formation. Toutefois, apprendre le Python à travers le projet était quelque chose qui ne nous effrayait pas.

Lors de l'avancement du projet, je me suis toutefois rendu compte que la réalisation de l'IHM Web ainsi que l'IHM physique était un peu trop ambitieux en terme de durée de projet. En accord avec les membres du groupe mais également les membres de la section CRCI, je me suis alors concentré sur la partie concernant la manette, qui était aussi la plus parlante pour eux car palpable.

Je vous présenterai tout de même le travail réalisé sur la page Web, le projet étant ambitieux et prévu sur plusieurs années, ce travail pourra être une base pour les années suivantes.

IV°) Conception, configuration et réalisation

Dans un premier temps, je vais vous présenter le travail réalisé au niveau de l'IHM Web. L'idée était d'avoir une page de paramétrage et une page de pilotage, en pouvant choisir entre les deux à l'aide du menu.

La page Paramètre doit permettre à l'utilisateur de sélectionner la vitesse maximale du chariot, ainsi que la distance des capteurs à ultrasons, mais aussi le mode de fonctionnement. En effet, la section CRCI souhaite avoir 3 modes de fonctionnement, à savoir :

- Libre (Permet de rouler librement sans contrainte)
- Crabe (Permet un déplacement uniquement latéral)
- Rotation (Permet uniquement la rotation sur lui même du chariot)

Voici quelques extraits de code pour ces paramètres :

```
<div id="parametres" >
  <label for="vmax"> Vitesse Max de 1 à 5 </label>
  <input type="number" id="vmax" name="VitesseMax" min="1" max="5">
  <form action="">
    <label for="dcapt"> Distance Capteur de 30 à 400 en cm </label>
    <input type="number" id="dcapt" name="DistanceCapteur" min="30" max="400">
```

Ici, on déclare les paramètres pour la vitesse, de type nombre, allant de 1 à 5, puis la distance capteur, aussi un nombre, allant de 30 à 400.

```
<legend>Modes de fonctionnement:</legend>

<div>
  <input type="radio" id="Libre" name="modes" value="mode1" checked />
  <label for="Libre">Libre</label>
</div>

<div>
  <input type="radio" id="Crabe" name="modes" value="mode2" />
  <label for="Crabe">Crabe</label>
</div>

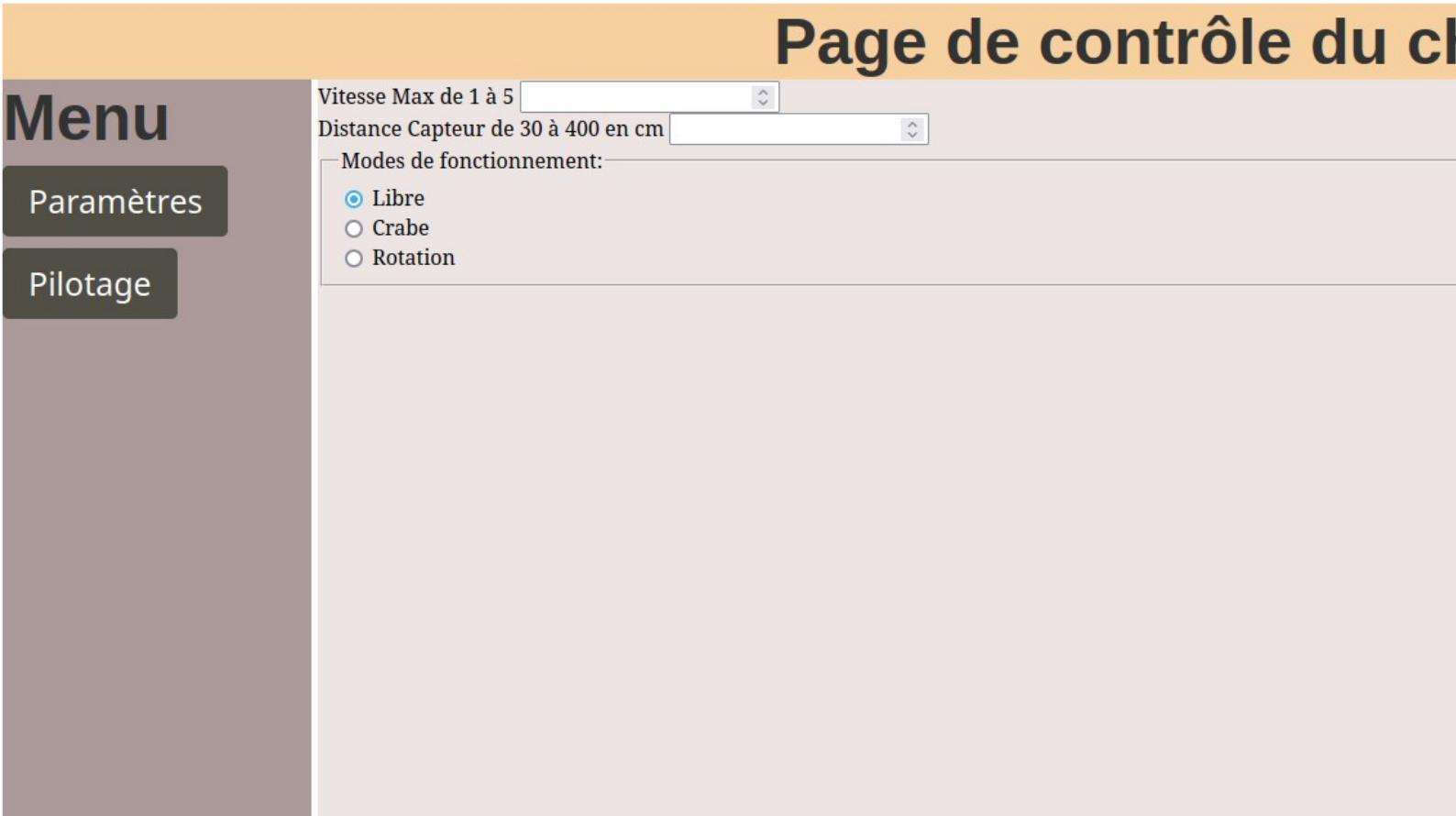
<div>
  <input type="radio" id="Rotation" name="modes" value="mode3" />
  <label for="Rotation">Rotation</label>
</div>
```

Ce code permet, avec la mise en place de 3 radios boutons, de sélectionner le mode de fonctionnement désiré par l'utilisateur, sachant qu'il ne peut en sélectionner qu'un seul .

Cela correspond au code JavaScript du menu, permettant de changer entre pilotage et paramètres. Pour cela, on écoute si l'utilisateur clique sur l'un des deux boutons du menu, puis on cache ce qui n'est pas nécessaire.

```
let togg1 = document.getElementById("b1");
let togg2 = document.getElementById("b2");
let d2 = document.getElementById("controller");
let d3 = document.getElementById("parametres");
let d4 = document.getElementById("arret");
d2.hidden = true;
d3.hidden = true;
d4.hidden = true;
document.getElementById("b1").addEventListener("click", () =>
  { d2.hidden = true;
    d3.hidden = false;
    d4.hidden = true;
  },
  false,);
document.getElementById("b2").addEventListener("click", () =>
  { d3.hidden = true;
    d4.hidden = false;
    d2.hidden = false;
  },
  false,);
```

Voici le visuel obtenu :



Pour la partie pilotage, elle est constituée de quatre flèches directionnelles, quatre boutons et également un bouton d'arrêt d'urgence.

Voici le code de ces différents boutons :

```
<div id="controller" class="controller">

    <!-- Boutons -->
    <div id="boutons">
        <button class="button top" type="button"></button>
        <button class="button left" type="button"></button>
        <button class="button bottom" type="button"></button>
        <button class="button right" type="button"></button>
    </div>
    <!-- Flèches directionnelles -->
    <div id="fleches">
        <button class="directional-pad down" type="button"></button>
        <button class="directional-pad up" type="button"></button>
        <button class="directional-pad left" type="button"></button>
        <button class="directional-pad right" type="button"></button>
    </div>

</div>
<div id="arret">
    <button class="urgence-btn">Arrêt d'Urgence</button>
</div>
```

Voici le code HTML permettant la déclaration des 4 boutons et des 4 flèches directionnelles, déclarées également comme bouton.

```

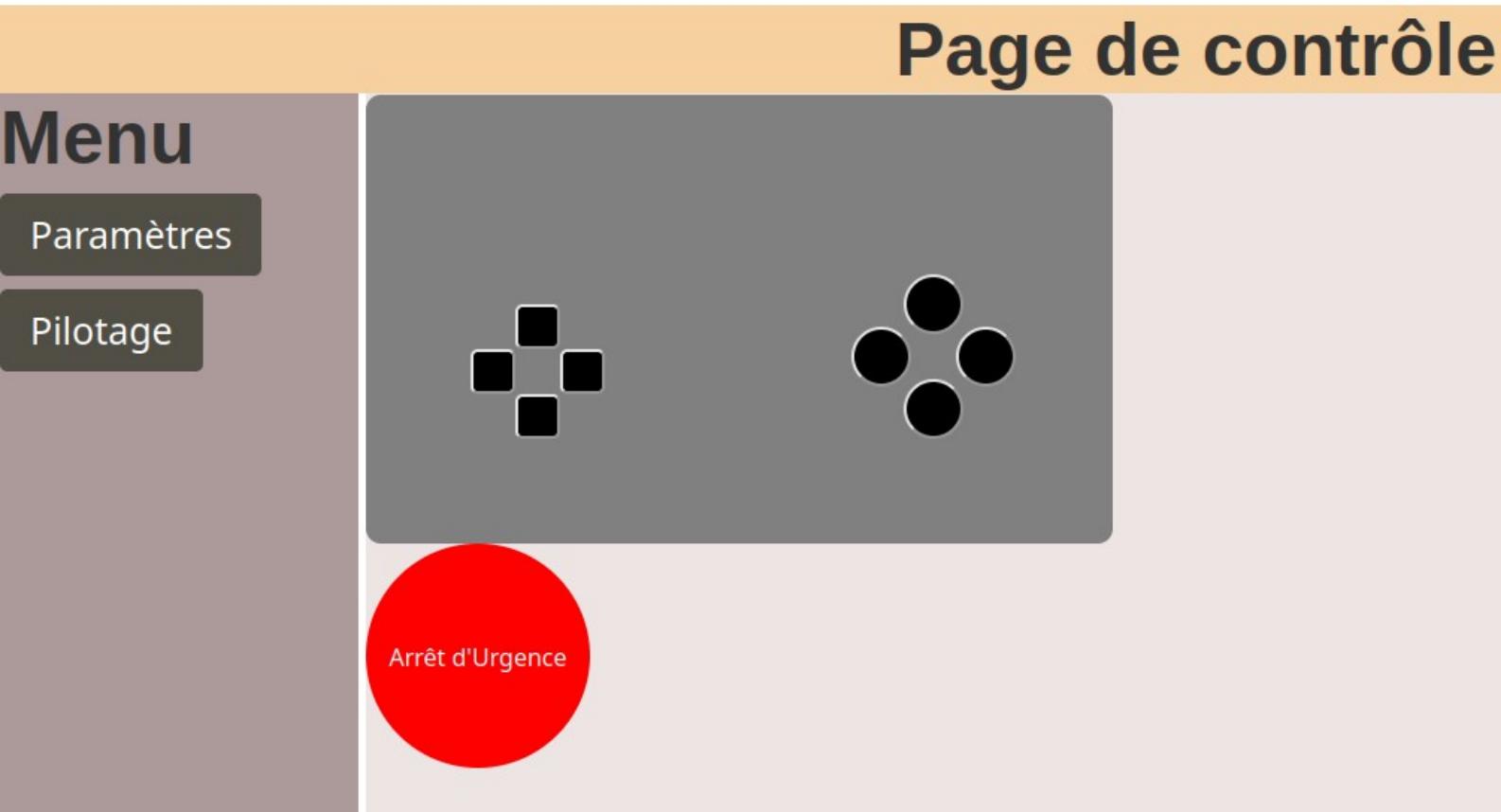
.controller {
  width: 500px;
  height: 300px;
  background-color: gray;
  border-radius: 10px;
  position: relative;
}
.button {
  width: 40px;
  height: 40px;
  background-color: black;
  border-radius: 50%;
  position: absolute;
}
.button:hover {
  background-color: #959389;
}
.button.top {
  bottom: 140px;
  right: 100px;
}
.button.left {
  bottom: 105px;
  right: 65px;
}
.button.bottom {
  bottom: 70px;
  right: 100px;
}
.button.right {
  bottom: 105px;
  right: 135px;
}

.directional-pad {
  width: 30px;
  height: 30px;
  border-radius: 5px;
  background-color: black;
  position: absolute;
}
.directional-pad:hover {
  background-color: #959389;
}
.directional-pad.down {
  bottom: 70px;
  left: 100px;
}
.directional-pad.up {
  bottom: 130px;
  left: 100px;
}
.directional-pad.left {
  bottom: 100px;
  left: 70px;
}
.directional-pad.right {
  bottom: 100px;
  left: 130px;
}
.urgence-btn {
  padding: 15px;
  background-color: red;
  color: white;
  border: none;
  border-radius: 50%;
  font-size: 16px;
  cursor: pointer;
  width: 150px;
  height: 150px;
  display: flex;
  justify-content: center;
  align-items: center;
  text-align: center;
}

```

Ici, le code CSS permettant de mettre en forme la manette, en rendant notamment les boutons ronds et en créant une véritable croix directionnelle comme une manette physique.

Voici le visuel obtenu :



Abordons désormais la partie la plus importante de mon travail, à savoir le code permettant de récupérer les informations provenant de la manette connectée en Bluetooth à l'ordinateur embarqué, et leur lien avec l'application de pilotage moteur.

Dans un premier temps, j'ai recherché une librairie me permettant non seulement de me connecter à la manette, mais également d'en écouter les événements, ce qui permet de récupérer les informations lors de la pression sur un bouton par exemple. Pour cela, j'ai donc utilisé la librairie Python nommée Pygame.

Pygame est une bibliothèque Python open-source conçue pour le développement de jeux vidéo. Elle permet aux développeurs de créer des jeux et des programmes multimédias en 2D de manière simple et efficace. Voici une présentation de ses principales caractéristiques et fonctionnalités :

Multiplateforme : Pygame est compatible avec divers systèmes d'exploitation, notamment Windows, macOS et Linux, ce qui permet de développer des jeux multiplateformes.

Simplicité d'utilisation : La bibliothèque est conçue pour être facile à utiliser, même pour les débutants en programmation. Elle offre une API intuitive et bien documentée.

Gestion des événements : La bibliothèque permet de gérer facilement les événements du clavier, de la souris et des contrôleurs de jeu, ce qui est essentiel pour l'interactivité dans les jeux.

Je vais maintenant vous présenter différents extraits du codeCe p permettant la gestion de la manette et de ses événements, ainsi que la réutilisation des données reçues :

```
import pygame
from pygame.locals import *
```

Permet d'importer, en début de code, la librairie Pygame.

```
pygame.init()
nb_joysticks = pygame.joystick.get_count()
print(nb_joysticks)
```

Permet d'initialiser les objets de type, en scannant les périphériques connectés. On récupère le nombre de joysticks reconnus, puis on l'affiche pour vérifier.

```
vmax = 1
```

Permet d'initialiser notre variable de vitesse maximale à 1.

```
if nb_joysticks > 0:  
  
    mon_joystick = pygame.joystick.Joystick(0)  
    print(mon_joystick)  
  
    mon_joystick.init()
```

Ici, on teste tout d'abord si un joystick est bien détecté, puis on le sélectionne afin de l'afficher puis de l'initialiser.

```
nb_boutons = mon_joystick.get_numbuttons()  
print(nb_boutons)  
if nb_boutons != 15:  
    print("Vous n'avez pas branché de Joystick...")  
    sys.exit()
```

On compte ensuite les boutons de notre manette, puis on l'affiche. Si ce nombre de boutons est différent de 15, qui est le nombre de boutons de la manette que l'on utilise pour ce projet, on affiche un petit message et l'on sort du programme à l'aide de la librairie sys.

La bibliothèque `sys` de Python fait partie de la bibliothèque standard et offre un accès direct à certaines variables et fonctions qui interagissent étroitement avec l'interpréteur Python. Elle permet notamment de manipuler des éléments essentiels de l'environnement d'exécution de Python.

```
membstate=map(mon_joystick.get_button,range(nb_boutons))  
print(membstate)
```

On initialise ici la mémoire de l'état des boutons, pour le nombre de boutons que l'on a compté. On affiche également cette valeur pour vérification.

```
memhat=[False,False]  
hat=[False,False]
```

On initialise ici l'état de la croix directionnel. La variable hat[1] contenant l'information des flèches haut et bas et hat[0] celle des flèches gauche et droite. La valeur est initialisée à False pour éviter toute action au lancement du programme.

```
continuer = 1

while continuer:
    time.sleep(0.01)
    #Màj de l'état de la manette
    ev=pygame.event.get()
    #recuperation de l'état des boutons
    bstate=map(mon_joystick.get_button,range(nb_boutons))
    # Permet de vérifier si un changement d'état a eu lieu entre 2 boucles
    bstate_ev=map(lambda c,m:c-m,bstate,membstate)
    # Conversion de bstate en decimal pour détecter les combinaisons de touches
    bstate_int=sum(map(lambda b:int(mon_joystick.get_button(b))*(2**b),range(nb_boutons)))
```

Ici, tant que le programme est lancé, on met à jour l'état de la manette, et on écoute surtout l'état des boutons, afin de savoir si un changement à lieu entre deux boucles. On effectue également une conversion pour traiter les combinaisons de touche, si l'on veut avancer et tourner en même temps par exemple.

Dans cette partie, on écoute les événements de la croix directionnelle. Si un bouton est pressé, cela modifie la valeur de nos variables, qui sont par la suite envoyés à la carte pilotage moteur qui engendrera des actions par la suite. Par exemple, si la flèche de gauche est pressée, la valeur de x passera à 1 et la roue tournera à gauche.

```

if nb_hats>0:
    hat=mon_joystick.get_hat(0)
    hat_ev=map(lambda hc,hm:hc!=hm,hat,memhat)
    |
y = 0
x = 0

while True in hat_ev:
    if hat[1] ==1 :
        #print("haut")
        y=2

    elif hat[1] ==-1 :
        y=1
        #print("bas")

    if hat[0] ==1 :
        #print("droite")
        x=2

    elif hat[0] ==-1 :
        #print("gauche")
        x=1

```

Ici, on traite le bouton R2, qui fait office d'accélérateur. S'il est pressé, la variable passe à 1, et les roues avancent. La vitesse des roues est également modifiable avec les boutons Y et X de la manette. Y permet de faire augmenter cette vitesse quand X permet de la diminuer, en agissant tous les deux sur la valeur de la variable vmax.

V°) Cahier de recette :

Fiche de test

- Recette globale : d'un cas d'utilisation et/ou d'une fonctionnalité globale**
- Test de faisabilité**
- Test unitaire**
- Test d'intégration**

Nom de l'étudiant : Treussart Mewen

Date : 21/05/2024

Objectif du test : Vérifier la récupération des événements de la manette

Conditions de réalisation

- Besoins matériels
Raspberry Pi 4
Manette Bluetooth
- Besoins logiciels (nom et version)
Thonny Python
- Noms des fichiers utilisés
testmanette.py

Scénario

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status OK / NOK
1	Sur la Raspberry, ouvrir Thonny.		Thonny est ouvert.	OK
2	Sur la Raspberry, lancer le Bluetooth		Bluetooth activé.	OK
3	Allumer la manette à l'aide du bouton central.		Manette allumée.	OK
4	Vérifier que la manette est bien connectée à la Raspberry		La manette est détectée et connectée à la Raspberry.	OK

5	Dans Thonny, lancer le programme testmanette		Le programme se lance sans erreur	OK
6	Vérifier lors de la pression des boutons si les informations sont bien obtenues		Les informations sont bien obtenues et conforme aux affichages prévus.	OK

Annexes (captures d'écran et schémas...)

haut
 2
 bas
 1
 gauche
 1
 droite
 2
 R2
 1
 up
 down

Fiche de test

- Recette globale : d'un cas d'utilisation et/ou d'une fonctionnalité globale
- Test de faisabilité
- Test unitaire
- Test d'intégration

Nom de l'étudiant : Treussart Mewen

Date : 21/05/2024

Objectif du test : Vérifier la récupération des événements de la manette et la transmission à la carte moteur

Conditions de réalisation

- Besoins matériels
Raspberry Pi 4

Manette Bluetooth
Carte moteur ESP32

- Besoins logiciels (nom et version)
Thonny Python
- Noms des fichiers utilisés
testmanette.py

Scénario

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status OK / NOK
1	Sur la Raspberry, ouvrir Thonny.		Thonny est ouvert.	OK
2	Sur la Raspberry, lancer le Bluetooth		Bluetooth activé.	OK
3	Allumer la manette à l'aide du bouton central.		Manette allumée.	OK
4	Vérifier que la manette est bien connectée à la Raspberry		La manette est détectée et connectée à la Raspberry.	OK
5	Dans Thonny, lancer le programme testmanette		Le programme se lance sans erreur	OK
6	Vérifier lors de la		Les informations sont	OK

	pression des boutons si les informations sont bien obtenues		bien obtenues et conforme aux affichages prévus.	
7	Vérifier que les informations sont bien envoyées à la carte moteur.	115200 bauds, liaison USB	Les informations sont bien envoyées à l'ESP 32, qui les reçoit et influe sur les moteurs en fonction.	OK

Annexes (captures d'écran et schémas...)

```

haut      Carte moteur: 12
2          Carte moteur: 02
bas        Carte moteur: 13
1          Carte moteur: 04
gauche
1
droite
2
R2
1
up
down

```

VI°) Problèmes rencontrés

Le premier problème rencontré est l'ambition du projet, et la difficulté à estimer le temps que différentes tâches a réalisé allaient prendre. En effet, j'ai pour ma part dû revoir à la baisse mes objectifs par manque de temps. Ensuite, la seconde difficulté rencontré est le manque d'information sur les transmissions Bluetooth sur une Raspberry sous Linux. C'est pourquoi le temps de recherche a été assez important afin de trouver les librairies me permettant de développer le code que je souhaitais. Un autre problème, majeur celui-ci, est en lien avec la section CRCI. En effet, ces derniers ont mis un temps important à livrer la première roue du chariot, pour finalement ne pas livrer le chargeur en entier.

VII°) Conclusion

En conclusion, ce projet m'a permis de développer mes compétences dans de multiples langages, à savoir l'HTML, le JavaScript, le CSS et le Python. De plus, j'ai appris à travailler avec du matériel Bluetooth, technologie peu utilisée au cours de la formation. De plus, apprendre à gérer des contraintes données par un client, s'adapter à celle-ci, et aussi gérer les différents retards et imperfections entraîner par un autre parti, font vraiment à part entière d'un projet.

Ce projet restera une expérience enrichissante en tout point, et me permettra à l'avenir d'être encore plus performant dans mes projets futurs.

Partie Individuelle :Rapport sur la Programmation et la Gestion des Capteurs du Chariot

Auteur: CARLIN Iannis

Introduction

Ce rapport présente la programmation et la gestion des capteurs pour un chariot motorisé téléguidé, destiné à transporter des plaques d'acier. Le chariot est contrôlé à distance via une manette connectée en Bluetooth à une carte Raspberry Pi, qui communique avec une carte Arduino Nano V3 pour la gestion des capteurs ainsi que 2 cartes ESP32 et 2 cartes Arduino pour la gestion des différents composants motorisé du chariot. Au cours de ce projet je me suis concentré sur la programmation de la carte Arduino Nano V3 qui s'occupe de la gestion des capteurs à ultrasons.

Objectifs du Projet

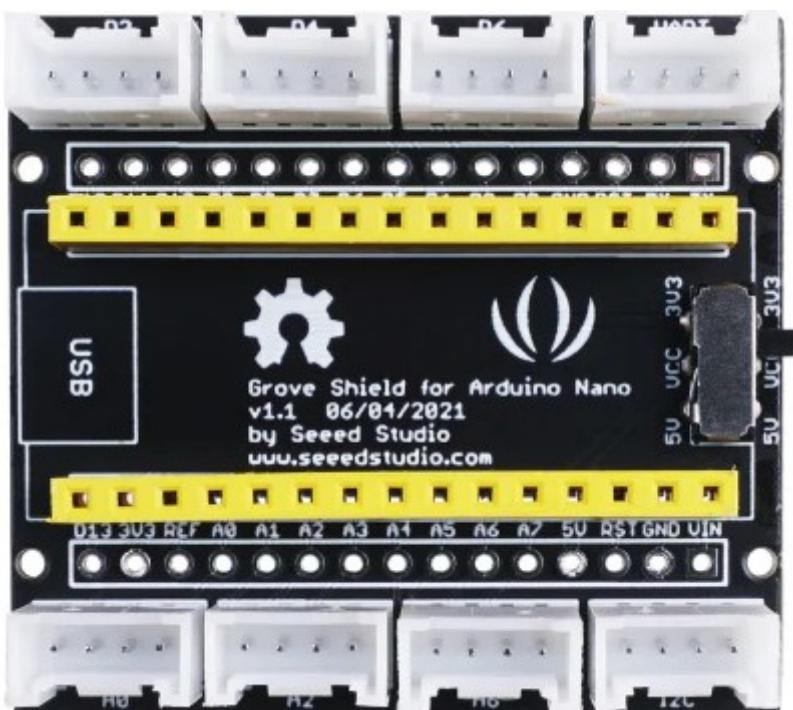
- Contrôle Téléguidé : Permettre le contrôle du chariot via une manette Bluetooth.
- Détection d'Obstacles : Utiliser des capteurs à ultrasons pour détecter les obstacles et assurer la sécurité du chariot.
- Signalisation : Intégrer un gyrophare pour signaler le mouvement du chariot.
- Gestion des Moteurs : Assurer la communication avec 2 cartes ESP32 et 2 cartes Arduino pour la gestion des moteurs.

Matériel Utilisé

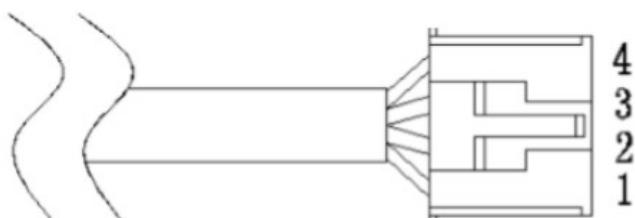
1. Carte Raspberry Pi : Serveur central pour la réception des commandes Bluetooth et le contrôle global.
2. Carte Arduino Nano V3 : Gestion des capteurs à ultrasons et du gyrophare.
3. Carte 2 cartes ESP32 et 2 cartes Arduino : Gestion des moteurs des roues.
4. Capteurs à Ultrasons ME007YS : Détection des obstacles autour du chariot.
5. Gyrophare : Indicateur lumineux pour signaler le mouvement.
6. Manette Bluetooth : Interface de commande pour l'utilisateur.

Connexions et Configuration des Capteurs

Les capteurs à ultrasons ME007YS ont été choisis pour leur imperméabilité à la poussière et leur précision. Ils sont connectés à la carte Arduino Nano V3 via des connectiques UART sur un shield Grove. Des ajustements ont été nécessaires (inversion entre les PINS 1 et 2) pour adapter les pins des capteurs au shield, garantissant ainsi une communication UART efficace.



Pinout



Label	Name	Description
1	VCC	Power Input
2	GND	Ground
3	RX	Function Pin
4	TX	UART Output

Tests Unitaires

Des tests unitaires ont été effectués pour vérifier le bon fonctionnement de chaque fonction individuelle du programme. Cela inclut la vérification de la lecture des données des capteurs, le calcul de la distance aux obstacles et la gestion de la communication série.

Fiche de test

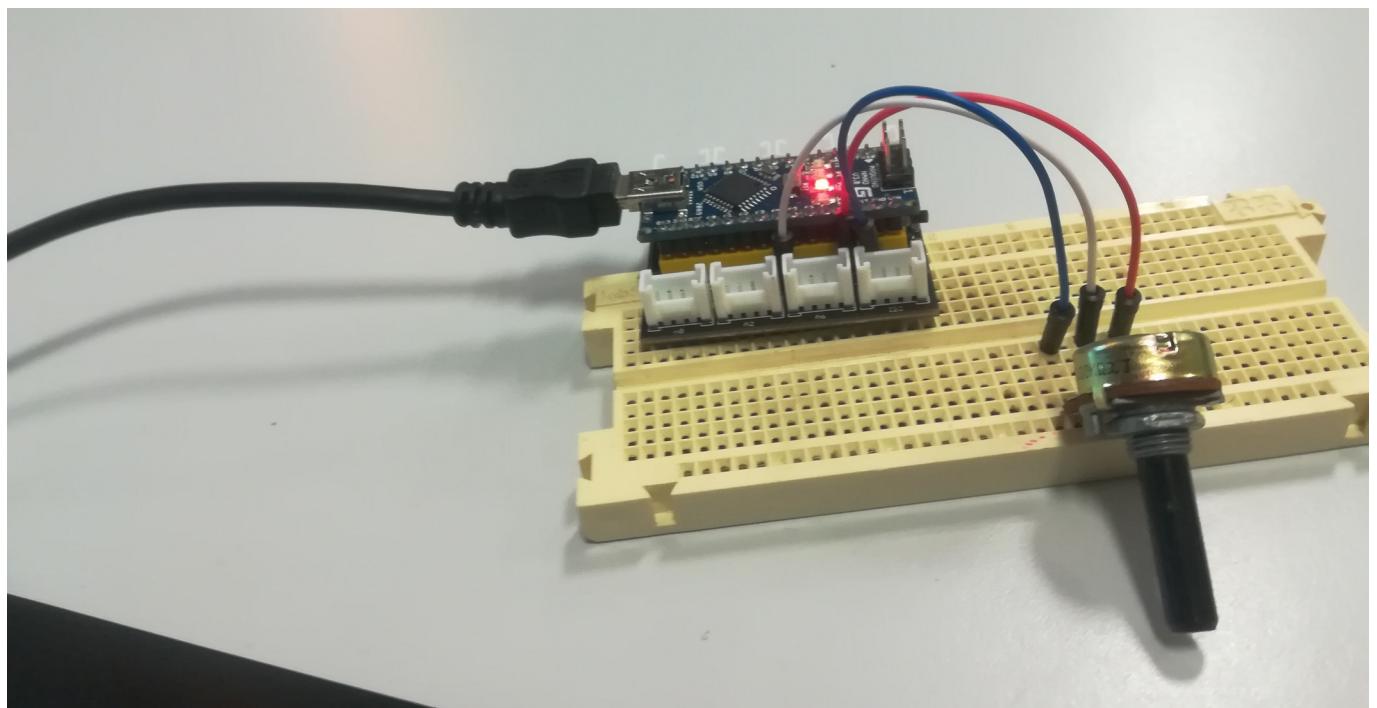
<input type="checkbox"/> Recette globale ; d'un cas d'utilisation et/ou d'une fonctionnalité globale <input checked="" type="checkbox"/> Test de faisabilité <input type="checkbox"/> Test unitaire <input type="checkbox"/> Test d'intégration			
Nom de l'étudiant: Carlin Jannis Date: 12/02/2024 Objectif du test: Déetecter un obstacle / Signaler un problème			
Conditions de réalisation <ul style="list-style-type: none">Besoins matériels<ul style="list-style-type: none">-carte arduino nano-capteurs à ultra-son (X6)-gyrophare-potentiomètreBesoins logiciels (nom et version)<ul style="list-style-type: none">-IDE arduinoNoms des fichiers utilisés<ul style="list-style-type: none">.ino			
Scénario			
ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu
1	Téléverser programme de test sur la carte arduino	IDE arduino	Carte fonctionnelle Code exécuté sans erreur
2	Mesure de distance à l'aide des capteurs	Recâblage des liaisons UART des capteurs	Affichage des distances attendu dans le programme de test (annexe 1)
3	Mise en marche du gyrophare		Le gyrophare éclaire correctement
4	Mesure d'angle du potentiomètre		Affichage de l'angle en ° du potentiomètre (annexe 2)

(annexe 1)



```
11:44:01.760 -> distance 1 =29.90cm  
11:44:02.255 -> distance 2 =47.30cm
```

(annexe 2)



```
117 void setup()
118 {
119     Serial.begin(115200);
120     Capteur_Avant.begin(9600);
121     Capteur_Avant_Gauche.begin(9600);
122     Capteur_Avant_Droite.begin(9600);
123     Capteur_Arriere.begin(9600);
124     Capteur_Arriere_Gauche.begin(9600);
125     Capteur_Arriere_Droite.begin(9600);
126     pinMode(18,INPUT);
127
128 }
129
130 void loop()
131 {
132     Serial.println(analogRead(18)*(1023/300)); //valeur potentiomètre [0;1023] angle [0°;300°]
133     delay(500);
134     /*
135     Désin Capteurs
136 }
```

Output Serial Monitor ×

New Line ▾ 115200 baud ▾

```
16:27:30.103 -> 103
16:27:30.583 -> 103
16:27:31.125 -> 103
```

Programmation de la Carte Arduino Nano V3

Choix du Langage de Programmation

Le langage C++ a été choisi pour programmer la carte Arduino Nano V3, en raison de sa compatibilité avec la carte et de la familiarité que j'avais avec ce langage. Étant donné que les capteurs à ultrasons utilisent des connexions par port série, j'ai utilisé la bibliothèque `SoftwareSerial` dans mon code pour créer des ports série virtuels, permettant la communication avec les capteurs.

Problème de Délai des Ports Série Virtuels

Le principal problème rencontré lors de la programmation était le délai nécessaire pour les ports série virtuels créés (environ 500 ms entre l'acquisition des données de deux capteurs). Cela m'a obligé à créer la fonction `Prio_Capteur` pour prioriser les capteurs en fonction de leur importance. De plus, un thread côté Raspberry Pi a été mis en place pour ne pas ralentir la communication avec les cartes de gestion des moteurs et la manette.

Programmation de la carte Arduino

1. Initialisation des Ports Série

La première étape de la programmation consiste à initialiser les ports série virtuels pour chaque capteur à ultrasons en utilisant la bibliothèque `SoftwareSerial`.

```
1 #include <SoftwareSerial.h>
2
3 SoftwareSerial Capteur_Avant(2,3); // (D2) RX, TX
4 SoftwareSerial Capteur_Avant_Gauche(4,5); // (D4) RX, TX
5 SoftwareSerial Capteur_Avant_Droite(6,7); // (D6) RX, TX
6 SoftwareSerial Capteur_Arriere(14,15); // (A0) RX, TX
7 SoftwareSerial Capteur_Arriere_Gauche(16,17); // (A2) RX, TX
8 SoftwareSerial Capteur_Arriere_Droite(0,1); // (uart) RX, TX
```

Chaque capteur est associé à un port RX et un port TX spécifiques pour permettre la communication série. Cela permet à la carte Arduino de recevoir des données de chaque capteur.

2. Déclaration des Variables

Les variables globales suivantes sont déclarées pour stocker les données reçues :

- l'état des moteurs (avant/arriere)
- l'orientation des roues (droit/gauche)
- la trame envoyée par la raspberry
- les trames envoyées par les capteurs
- une variable stockant la distance des objets

```

10 int avant =0;
11 int arriere =0;
12 int droite =0;
13 int gauche =0;
14
15 String a = "10"; //buffer pour la trame recus de la raspberry
16
17 unsigned char data[4]={}; //buffer pour les trames recu des capteurs
18 float distance;

```

3. Fonction info_capteur

Cette fonction lit les données des capteurs à ultrasons et calcule la distance d'un obstacle. Elle envoie ensuite un signal à la carte Raspberry Pi en fonction de la distance mesurée afin d'arrêter le chariot en cas de danger.

- envoie «!0 » si aucun danger n'est à signaler
- envoie «!1 » si il faut arrêter le chariot
- envoie «!2 » si il y a eu un problème dans l'acquisition des données du capteur

```

20 void info_capteur(SoftwareSerial mySerial) //calcule la distance d'un obstacle et communique le danger à la catre RaspberryPi
21 {
22     mySerial.listen(); //écoute le port du capteur concerné
23
24     delay(500);
25
26     do //mise à jour de la trame (bit 0 : header=0xff; bit 1 : distance hight 8-bit; bit 2 : distance low 8-bit; bit 3 : checksum
27     {
28         for(int i=0;i<4;i++)
29         {
30             data[i]=mySerial.read();
31         }
32     }while(mySerial.read()==0xff);
33
34     mySerial.flush();
35     if(data[0]==0xff)
36     {
37
38         int sum;
39         sum=(data[0]+data[1]+data[2])&0x00FF;
40         if(sum==data[3]) //vérification du checksum
41         {
42             distance=(data[1]<<8)+data[2]; //calcule de la distance
43             if(distance>300) //objet assez loin
44             {
45                 Serial.println("10");
46             }
47             else //objet trop proche
48             {
49                 Serial.println("11");
50             }
51         }
52         else //erreur de données
53         {
54             Serial.println("12");
55         }
56     }
57     mySerial.stopListening(); //arret de l'écoute du port
58 }

```

Cette fonction est essentielle pour la détection des obstacles et la sécurité du chariot. Elle vérifie l'intégrité des données en utilisant un checksum et calcule la distance à l'obstacle le plus proche.

4. Fonction Prio_Capteur

Cette fonction détermine les capteurs à lire en priorité en fonction de l'état des variables avant, gauche et droite (et arrière lorsque la marche arrière sera implémenté au projet).

```
60 void Prio_Capteur() //récupère les informations des capteurs prioritaire
61 {
62     if (avant)
63     {
64         info_capteur(Capteur_Avant);
65         //Serial.println("Capteur_Avant");
66     }
67     if (gauche)
68     {
69         info_capteur(Capteur_Avant_Gauche);
70         //Serial.println("Capteur_Avant_Gauche");
71     }
72     else if (droite)
73     {
74         info_capteur(Capteur_Avant_Droite);
75         //Serial.println("Capteur_Avant_Droite");
76     }
77     else
78     {
79         info_capteur(Capteur_Avant);
80         //Serial.println("Capteur_Avant");
81     }
82 }
83 */
84 /*
85     /* else if (arriere)
86     {
87         info_capteur(Capteur_Arriere);
88         Serial.println("Capteur_Arriere");
89     }
90     if (gauche)
91     {
92         info_capteur(Capteur_Arriere_Gauche);
93         Serial.println("Capteur_Arriere_Gauche");
94     }
95     else if (droite)
96     {
97         info_capteur(Capteur_Arriere_Droite);
98         Serial.println("Capteur_Arriere_Droite");
99     }
100    else
101    {
102        info_capteur(Capteur_Arriere);
103        Serial.println("Capteur_Arriere");
104    }
105 }
106 */
107 else //si le chargeur et à l'arrêt on autorise l'avancement un cours instant quel qu soit les obstacles
108 {
109     Serial.println("!0");
110 }
111 }
112 }
```

Cette fonction permet de gérer la priorité des capteurs en fonction de la direction du mouvement du chariot, optimisant ainsi la détection des obstacles et réduisant l'impact du délai entre 2 mesure possible.

5. Fonction Angle_Roue

Cette fonction lit la valeur analogique d'un potentiomètre pour déterminer l'orientation des roues et met à jour les variables droite et gauche.

```
114 float Angle_Roue(int port) //calcule la position des roues (valeur potentiomètre [0;1023] angle [-150°;150°])
115 {
116     if (analogRead(port) < 500) //roues orientées à gauche
117     {
118         droite = 0;
119         gauche = 1;
120     }
121     else if (analogRead(port) > 523) //roues orientées à droite
122     {
123         droite = 1;
124         gauche = 0;
125     }
126     else //roues alignées
127     {
128         droite = 0;
129         gauche = 0;
130     }
131     delay(500);
132 }
```

Cette fonction est utilisée pour connaître l'orientation des roues du chariot, ce qui est crucial pour la fonction Prio_Capteur.

6. Fonction CharToBool

Cette fonction transforme un caractère en une valeur booléenne.

```
134 bool CharToBool(char carac) //permet de transformer un char en booleen
135 {
136     if (carac == '0')
137     | return false;
138     return true;
139 }
```

Cette fonction simplifie la conversion des commandes reçues de la Raspberry Pi, permettant de les traiter efficacement dans le code.

7. Fonction setup

La fonction setup initialise les ports série et les pins de l'Arduino Nano V3.

```

141 void setup() //initialise toute les connection
142 {
143     Serial.begin(115200);
144     Capteur_Avant.begin(9600);
145     Capteur_Avant_Gauche.begin(9600);
146     Capteur_Avant_Droite.begin(9600);
147     Capteur_Arriere.begin(9600);
148     Capteur_Arriere_Gauche.begin(9600);
149     Capteur_Arriere_Droite.begin(9600);
150     pinMode(18,INPUT);
151     pinMode(19,OUTPUT);
152 }

```

Cette fonction est exécutée une seule fois au démarrage de la carte Arduino et initialise tous les ports série nécessaires à la communication avec les capteurs à ultrasons. Elle configure également les pins utilisés pour lire les valeurs analogiques du potentiomètre et contrôler le gyrophare.

8. Fonction loop

La fonction loop est le cœur du programme, exécutant en boucle les commandes de la Raspberry Pi et appelant les fonctions nécessaires à la détection des obstacles et à la gestion du gyrophare.

```

154 void loop() //programme principal
155 {
156     if (Serial.readString() != "")
157     {
158         a = Serial.readString(); // lit les données reçus par la raspberryPi
159         if(CharToBool(a[1])) //si les roues sont en marche
160         {
161             avant = 1; // mise à jour de la variable pour la fonction Prio_Capteur
162             analogWrite(19,255); //mise en tension du gyrophare
163         }
164         else //sinon
165         {
166             avant =0; // mise à jour de la variable pour la fonction Prio_Capteur
167             analogWrite(19,0); //mise hors tension du gyrophare
168         }
169     }
170     //Angle_Roue(18);
171     Prio_Capteur(); //mise ajour des obstacles potentiels
172 }

```

Cette fonction lit les commandes reçues via le port série, met à jour les états des variables « avant gauche et droite » avant d'appeler les fonctions de détection et de priorité des capteurs. Elle contrôle également l'état du gyrophare en fonction de l'état des moteurs.

Programmation de la communication (RaspberryPi)

```
import threading
from queue import LifoQueue, Empty

line3 = ""

def tache_capteur(result_stack):
    while(1):
        time.sleep(0.05)
        tempo = ser_capt.readline().decode('utf-8').rstrip()
        result_stack.put(tempo)

result_stack = LifoQueue()

thread_capteur = threading.Thread(target=tache_capteur,
args=(result_stack,))

thread_capteur.daemon = True

thread_capteur.start()
```

Améliorations possibles

Pour améliorer le système actuel, plusieurs pistes peuvent être explorées. Tout d'abord, une optimisation des délais de communication entre les capteurs et la carte Arduino Nano V3 pourrait être envisagée. En outre, l'ajout de planification de trajectoire pourrait rendre le chariot plus autonome et capable de naviguer efficacement dans des espaces encombrés. Pour ce faire, il serait envisageable de mettre un capteur magnétique sous le chariot afin de « dessiner » des chemins pré définis à l'aide de bande magnétique au sol. Enfin, des améliorations de l'interface utilisateur, telles que des indicateurs visuels ou des alertes sonores en cas de danger détecté, pourraient accroître la convivialité du système et améliorer la sécurité des opérations de transport. En combinant ces améliorations, le chariot motorisé téléguidé pourrait devenir un outil encore plus polyvalent et efficace pour le transport de plaque d'acier.

Conclusion

La programmation et la gestion des capteurs pour le chariot motorisé téléguidé ont été couronnées de succès, malgré les défis liés aux délais de communication des ports série virtuels. Les solutions mises en œuvre, telles que la fonction `Prio_Capteur` et l'utilisation d'un thread côté Raspberry Pi, ont permis d'optimiser le fonctionnement global du système malgré les problèmes de délai rencontré. Le chariot est maintenant capable de détecter les obstacles, de signaler son mouvement via le gyrophare et de répondre efficacement aux commandes de la manette Bluetooth.

Ce projet a démontré l'importance de la collaboration entre les équipes de programmation, et a mis en évidence la nécessité d'une communication efficace entre les différents composants d'un système complexe.

Annexe

Manuel d'utilisation de la manette



Bouton R2 (grande gâchette droite): Avancer

Flèche gauche: Tourner à gauche

Flèche droite: Tourner à droite

Y: Augmenter la vitesse

X: Diminuer la vitesse

Bouton central: Allumer ou éteindre la manette en restant appuyé

Code des tests :

TP_test_motorshield :

```
#include <Arduino.h>
```

```
int directionPin = 12;  
int pwmPin = 3;  
int brakePin = 9;
```

```
//uncomment if using channel B, and remove above definitions  
//int directionPin = 13;  
//int pwmPin = 11;  
//int brakePin = 8;  
//boolean to switch direction
```

```
bool directionState;  
void setup()  
{  
//define pins  
pinMode(directionPin, OUTPUT);  
pinMode(pwmPin, OUTPUT);  
pinMode(brakePin, OUTPUT);  
}
```

```
void loop()  
{  
//change direction every loop()  
directionState = !directionState;  
//write a low state to the direction pin (13)  
if(directionState == false)  
{  
  digitalWrite(directionPin, LOW);  
}  
//write a high state to the direction pin (13)  
else  
{  
  digitalWrite(directionPin, HIGH);  
}  
//release breaks  
digitalWrite(brakePin, LOW);
```

```
//set work duty for the motor
analogWrite(pwmPin, 255);
delay(2000);
//activate breaks
digitalWrite(brakePin, HIGH);
//set work duty for the motor to 0 (off)
analogWrite(pwmPin, 0);
delay(2000);
}
```

TP_test_output_esp32 :

```
#include <inttypes.h>
#include <stdio.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <ds18x20.h>
#include <esp_err.h>
#include "esp_system.h"
#include "esp_spi_flash.h"
#include "driver/gpio.h"
#include "driver/dac.h"
#include<unistd.h>

int app_main()
{
    dac_output_enable(DAC_CHAN_0);
    while(1)
    {
        dac_output_voltage(DAC_CHAN_0, 255);
    }
    return 0;
}
```

Tp_test_arduino_liaison_serie :

```
#include <Arduino.h>

void setup() {
    Serial.begin(9600); // Démarre la communication série à 9600
    bauds
}

void loop()
{

    if (Serial.available() > 0)
    {
        String variable = Serial.readStringUntil('\n');
        Serial.print("message reçu: ");
        Serial.println(variable);
    }
}
```

Tp_test_liaison_serie_esp32:

```
#include <inttypes.h>
#include <stdio.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <ds18x20.h>
#include <esp_err.h>
#include "esp_system.h"
#include "esp_spi_flash.h"
#include "driver/gpio.h"
#include "driver/dac.h"
#include <unistd.h>
#include "driver/uart.h"
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/uart.h"
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "esp_log.h"
```

```

/***
 * This is an example which echos any data it receives on
configured UART back to the sender,
 * with hardware flow control turned off. It does not use UART driver
event queue.
 *
 * - Port: configured UART
 * - Receive (Rx) buffer: on
 * - Transmit (Tx) buffer: off
 * - Flow control: off
 * - Event queue: off
 * - Pin assignment: see defines below (See Kconfig)
*/

```

```

#define ECHO_TEST_TXD (GPIO_NUM_1)
#define ECHO_TEST_RXD (GPIO_NUM_3)
#define ECHO_TEST_RTS (UART_PIN_NO_CHANGE)
#define ECHO_TEST_CTS (UART_PIN_NO_CHANGE)

#define ECHO_UART_PORT_NUM      (0)
#define ECHO_UART_BAUD_RATE    (115200)
#define ECHO_TASK_STACK_SIZE   (2048)

static const char *TAG = "UART TEST";

#define BUF_SIZE (1024)

static void echo_task(void *arg)
{
    /* Configure parameters of an UART driver,
     * communication pins and install the driver */
    uart_config_t uart_config = {
        .baud_rate = ECHO_UART_BAUD_RATE,
        .data_bits = UART_DATA_8_BITS,
        .parity   = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .source_clk = UART_SCLK_DEFAULT,
    };
    int intr_alloc_flags = 0;
}

```

```

#if CONFIG_UART_ISR_IN_IRAM
    intr_alloc_flags = ESP_INTR_FLAG_IRAM;
#endif

ESP_ERROR_CHECK(uart_driver_install(ECHO_UART_PORT_NUM,
BUF_SIZE * 2, 0, 0, NULL, intr_alloc_flags));

ESP_ERROR_CHECK(uart_param_config(ECHO_UART_PORT_NUM,
&uart_config));
    ESP_ERROR_CHECK(uart_set_pin(ECHO_UART_PORT_NUM,
ECHO_TEST_TXD, ECHO_TEST_RXD, ECHO_TEST_RTS,
ECHO_TEST_CTS));

// Configure a temporary buffer for the incoming data
uint8_t *data = (uint8_t *) malloc(BUF_SIZE);

while (1) {
    // Read data from the UART
    int len = uart_read_bytes(ECHO_UART_PORT_NUM, data,
(BUF_SIZE - 1), 20 / portTICK_PERIOD_MS);
    // Write data back to the UART
    uart_write_bytes(ECHO_UART_PORT_NUM, (const char *) data,
len);
    if (len) {
        data[len] = '\0';
        ESP_LOGI(TAG, "Recv str: %s", (char *) data);
    }
}
}

void app_main(void)
{
    xTaskCreate(echo_task, "uart_echo_task",
ECHO_TASK_STACK_SIZE, NULL, 10, NULL);
}

```

Code de la solution :

Arduino :

```
#include <Arduino.h>
#include <inttypes.h>
#include <stdio.h>
#include <unistd.h>
int directionPin = 12;
int pwmPin = 3;
int brakePin = 9;
int direction_roue = 0;
void setup()
{
    Serial.begin(9600); // Démarré la communication série à 9600
    bauds
    pinMode(directionPin, OUTPUT);
    pinMode(pwmPin, OUTPUT);
    pinMode(brakePin, OUTPUT);
}
void tourner_droite()
{
    digitalWrite(directionPin, LOW);
    digitalWrite(brakePin, LOW);
    analogWrite(pwmPin, 255);
}
void tourner_gauche()
{
    digitalWrite(directionPin, HIGH);
    digitalWrite(brakePin, LOW);
    analogWrite(pwmPin, 255);
}
void frein()
{
    digitalWrite(brakePin, HIGH);
    analogWrite(pwmPin, 0);
}
void recevoir()
{
```

```

if (Serial.available() > 0)
{
    direction_roue = Serial.read();

    Serial.println(direction_roue);

}

delay(10);
}

void output()
{
    if (direction_roue == 49)
    {
        tourner_droite();
    }
    else if (direction_roue == 50)
    {
        tourner_gauche();
    }
    else
    {
        frein();
    }
    delay(40);
    direction_roue = 0;
}

}

void loop()
{
    recevoir();
    output();
}

```

Code esp32 :

```

#include <inttypes.h>
#include <stdio.h>
#include <freertos/FreeRTOS.h>

```

```
#include <freertos/task.h>
#include <ds18x20.h>
#include <esp_err.h>
#include "esp_system.h"
#include "esp_spi_flash.h"
#include "driver/dac.h"
#include <unistd.h>
#include "driver/uart.h"
#include <stdio.h>
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "esp_log.h"

#define TEST_TXD (GPIO_NUM_1)
#define TEST_RXD (GPIO_NUM_3)
#define TEST_RTS (UART_PIN_NO_CHANGE)
#define TEST_CTS (UART_PIN_NO_CHANGE)

#define UART_PORT_NUM (0)
#define UART_BAUD_RATE (115200)
#define TASK_STACK_SIZE (2048)
#define BUF_SIZE (1024)

bool etat_roue = false;
int vitesse = 120;

void vitesse_roue(uint8_t *trame)
{
    if (trame[1] == '1' && vitesse <= 253)
    {
        vitesse += 2;
    }
    else if (trame[1] == '2' && vitesse >= 122)
    {
        vitesse -= 2;
    }
}

void obstacle(uint8_t *trame)
```

```

{
    if(trame[3] == '1')
    {
        etat_roue = false;
    }
}

static void tache_output_roue1()
{
    while(1)
    {
        if (etat_roue)
        {
            dac_output_voltage(DAC_CHAN_0, vitesse);
        }
        else
        {
            dac_output_voltage(DAC_CHAN_0, 0);
        }
        vTaskDelay(10);

    }
}

static void tache_output_roue2()
{
    while(1)
    {
        if (etat_roue)
        {
            dac_output_voltage(DAC_CHAN_1, vitesse);
        }
        else
        {
            dac_output_voltage(DAC_CHAN_1, 0);
        }
        vTaskDelay(10);

    }
}

```

```

static void tache_recevoir()
{
    uart_config_t uart_config =
    {
        .baud_rate = UART_BAUD_RATE,
        .data_bits = UART_DATA_8_BITS,
        .parity   = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .source_clk = UART_SCLK_DEFAULT,
    };
    int intr_alloc_flags = 0;

#if CONFIG_UART_ISR_IN_IRAM
    intr_alloc_flags = ESP_INTR_FLAG_IRAM;
#endif

    ESP_ERROR_CHECK(uart_driver_install(UART_PORT_NUM,
BUF_SIZE * 2, 0, 0, NULL, intr_alloc_flags));
    ESP_ERROR_CHECK(uart_param_config(UART_PORT_NUM,
&uart_config));
    ESP_ERROR_CHECK(uart_set_pin(UART_PORT_NUM,
TEST_TXD, TEST_RXD, TEST_RTS, TEST_CTS));

    // Configure un buffer pour les données qui arrivent
    uint8_t *data = (uint8_t *) malloc(BUF_SIZE);

    while (1)
    {
        // Lis les données UART
        int len = uart_read_bytes(UART_PORT_NUM, data,
(BUF_SIZE - 1), 20 / portTICK_PERIOD_MS);
        // Change l'état des roues en fonction du message reçu
        vitesse_roue(data);
        if(data[0] == '1')
        {
            etat_roue = true;
        }
        else

```

```

    {
        etat_roue = false;
    }

    // Renvoie les données UART
    uart_write_bytes(UART_PORT_NUM, (const char *) data, len);
    vTaskDelay(10);
}
}

void app_main(void)
{
    dac_output_enable(DAC_CHAN_0);
    dac_output_enable(DAC_CHAN_1);
    xTaskCreate(tache_recevoir, "tache_recevoir",
TASK_STACK_SIZE, NULL, 10, NULL);
    xTaskCreate(tache_output_roue1, "tache_output_roue1", 2048,
NULL, 10, NULL);
    xTaskCreate(tache_output_roue2, "tache_output_roue2", 2048,
NULL, 10, NULL);
}

```

Code carte capteur :

```

#include <SoftwareSerial.h>

SoftwareSerial Capteur_Avant(4,5); // (D4) RX, TX
SoftwareSerial Capteur_Arriere(16,17); // (A2) RX, TX

bool avant;
bool arriere;
bool droite;
bool gauche;

unsigned char data[4]={};
float distance;

void info_capteur(SoftwareSerial mySerial)
{
    mySerial.listen();

    delay(500);

    do
    {
        for(int i=0;i<4;i++)
        {
            data[i]=mySerial.read();
        }
    }
}
```

```

        }
    }while(mySerial.read()==0xff);

    mySerial.flush();

    if(data[0]==0xff)
    {
        int sum;
        sum=(data[0]+data[1]+data[2])&0x00FF;
        if(sum==data[3])
        {
            distance=(data[1]<<8)+data[2];
            if(distance>280)
            {
                Serial.print("distance =");
                Serial.print(distance/10);
                Serial.println("cm");
            }
            else
            {
                Serial.println("Below the lower limit");
            }
        }
        else
        {
            Serial.print(data[0]);
            Serial.print(";");
            Serial.print(data[1]);
            Serial.print(";");
            Serial.print(data[2]);
            Serial.print(";");
            Serial.print(data[3]);

            Serial.println("ERROR1");
        }
    }

    mySerial.stopListening();
}

void setup()
{
    Serial.begin(115200);
    mySerial0.begin(9600);
    mySerial1.begin(9600);import pygame
import threading
from queue import LifoQueue, Empty
from pygame.locals import *
import sys
import serial
ser=serial.Serial(port='/dev/ttyACM0',baudrate=9600, timeout=0.05)
ser2=serial.Serial(port='/dev/ttyUSB0',baudrate=115200, timeout=0.1)
ser_capt=serial.Serial(port='/dev/ttyUSB1',baudrate=115200, timeout=0.1)

pygame.init()

nb_joysticks = pygame.joystick.get_count()
print(nb_joysticks)
disable=0
#Bouton pour limiter la vitesse du pointeur
LIMITER=7
vmax = 1
line3 = ""

```

```

def tache_capteur(result_stack):
    while(1):
        time.sleep(0.05)
        tempo = ser_capt.readline().decode('utf-8').rstrip()
        result_stack.put(tempo)

result_stack = LifoQueue()

thread_capteur = threading.Thread(target=tache_capteur, args=(result_stack,))
thread_capteur.daemon = True
thread_capteur.start()

if nb_joysticks > 0:

    mon_joystick = pygame.joystick.Joystick(0)
    print(mon_joystick)

    mon_joystick.init()

    #On compte les boutons
    nb_boutons = mon_joystick.get_numbuttons()
    print(nb_boutons)
    if nb_boutons != 15:
        print("Vous n'avez pas branché de Joystick...")
        sys.exit()

    # Init de la memoire pour les etats des boutons
    membstate=map(mon_joystick.get_button, range(nb_boutons))
    print(membstate)
    # Init de la memoire pour l' etats de la croix
    memhat=[False, False]
    hat=[False, False]

    continuer = 1

    while continuer:
        time.sleep(0.01)
        #Maj de l'etats de la manette
        ev=pygame.event.get()
        #recuperation de l'état des boutons
        bstate=map(mon_joystick.get_button, range(nb_boutons))
        # Permet de verifier si un changement d'état a eu lieu entre 2 boucles
        bstate_ev=map(lambda c,m:c-m,bstate,membstate)
        # Conversion de bstate en decimal pour detecter les combinaison de
        touches
        bstate_int=sum(map(lambda
b:int(mon_joystick.get_button(b))*(2**b),range(nb_boutons)))

        # Gestion des touche HAUT,BAS,GAUCHE,DROITE du clavier avec
        # la croix de la manette
        nb_hats=mon_joystick.get_numhats()
        global command
        global command2
        if mon_joystick.get_button(4) == 1:
            #print("up")
            vmax = 1
            #if vmax < 4:
            #    vmax += 1
        if mon_joystick.get_button(3) == 1:
            #print("down")

```

```

        vmax = 2
        #if vmax > 0:
        #    vmax -= 1
    if mon_joystick.get_button(3) == 0 and mon_joystick.get_button(4) ==
0:
        vmax = 0

    if nb_hats>0:
        hat=mon_joystick.get_hat(0)
        hat_ev=map(lambda hc,hm:hc!=hm,hat,memhat)
        d=sroot.query_pointer().__data__
        y = 0
        x = 0

        while True in hat_ev:
            if hat[1] ==1 :
                #print("haut")
                y=2

            elif hat[1] ==-1 :
                y=1
                #print("bas")

            if hat[0] ==1 :
                #print("droite")
                x=2

            elif hat[0] ==-1 :
                #print("gauche")
                x=1
            if mon_joystick.get_button(9) == 1:
                #print("R2")
                y=1

        print("-----")

        command=str(x)
        ser.write(command.encode('utf-8'))
        line=ser.readline().decode('utf-8').rstrip()
        print("ligne direction: " +line)

        try:
            line3 = result_stack.get_nowait()
        except Empty:
            pass
        #command_capt='!'+str(y)
        #ser_capt.write(command_capt.encode('utf-8'))
        #line3=ser_capt.readline().decode('utf-8').rstrip()
        #print("ligne capteur: " +line3)

        command2=str(y)+str(vmax)+str(line3)
        ser2.write(command2.encode('utf-8'))
        line2=ser2.readline().decode('utf-8').rstrip()
        print("ligne moteur: "+line2)

else:
    print("Vous n'avez pas branché de Joystick...")

}

```

```

void loop()
{
    if (avant)
    {
        info_capteur(Capteur_Avant)
        if (gauche)
        {
            info_capteur(Capteur_Avant_Gauche)
        }
        else if (droite)
        {
            info_capteur(Capteur_Avant_Droite)
        }
        info_capteur(Capteur_Avant)
    }
    else if (arriere)
    {
        info_capteur(Capteur_Arriere)
        if (gauche)
        {
            info_capteur(Capteur_Arriere_Gauche)
        }
        else if (droite)
        {
            info_capteur(Capteur_Arriere_Droite)
        }
        info_capteur(Capteur_Arriere)
    }
}

```

}

Code Application gestion / IHM :

```

import pygame
import threading
from queue import LifoQueue, Empty
from pygame.locals import *
import sys
import serial
ser=serial.Serial(port='/dev/ttyACM0',baudrate=9600, timeout=0.05)
ser2=serial.Serial(port='/dev/ttyUSB0',baudrate=115200, timeout=0.1)
ser_capt=serial.Serial(port='/dev/ttyUSB1',baudrate=115200, timeout=0.1)

pygame.init()

nb_joysticks = pygame.joystick.get_count()
print(nb_joysticks)
disable=0
#Bouton pour limiter la vitesse du pointeur
LIMITER=7
vmax = 1
line3 = ""

def tache_capteur(result_stack):
    while(1):
        time.sleep(0.05)
        tempo =ser_capt.readline().decode('utf-8').rstrip()
        result_stack.put(tempo)

result_stack = LifoQueue()

```

```

thread_capteur = threading.Thread(target=tache_capteur, args=(result_stack,))

thread_capteur.daemon = True

thread_capteur.start()

if nb_joysticks > 0:

    mon_joystick = pygame.joystick.Joystick(0)
    print(mon_joystick)

    mon_joystick.init()

    #On compte les boutons
    nb_boutons = mon_joystick.get_numbuttons()
    print(nb_boutons)
    if nb_boutons != 15:
        print("Vous n'avez pas branché de Joystick...")
        sys.exit()

    # Init de la memoire pour les etats des boutons
    membstate=map(mon_joystick.get_button,range(nb_boutons))
    print(membstate)
    # Init de la memoire pour l' etats de la croix
    memhat=[False,False]
    hat=[False,False]

    continuer = 1

    while continuer:
        time.sleep(0.01)
        #Maj de l'etats de la manette
        ev=pygame.event.get()
        #recuperation de l'état des boutons
        bstate=map(mon_joystick.get_button,range(nb_boutons))
        # Permet de verifier si un changement d'état a eu lieu entre 2 boucles
        bstate_ev=map(lambda c,m:c-m,bstate,membstate)
        # Conversion de bstate en decimal pour detecter les combinaison de
        touches
        bstate_int=sum(map(lambda
b:int(mon_joystick.get_button(b))*(2**b),range(nb_boutons)))

        # Gestion des touche HAUT,BAS,GAUCHE,DROITE du clavier avec
        # la croix de la manette
        nb_hats=mon_joystick.get_numhats()
        global command
        global command2
        if mon_joystick.get_button(4) == 1:
            #print("up")
            vmax = 1
            #if vmax < 4:
            #    vmax += 1
        if mon_joystick.get_button(3) == 1:
            #print("down")
            vmax = 2
            #if vmax > 0:
            #    vmax -= 1
        if mon_joystick.get_button(3) == 0 and mon_joystick.get_button(4) ==
0:
            vmax = 0

        if nb_hats>0:
            hat=mon_joystick.get_hat(0)

```

```

hat_ev=map(lambda hc,hm:hc!=hm,hat,memhat)
d=sroot.query_pointer().__data__
y = 0
x = 0

while True in hat_ev:
    if hat[1] ==1 :
        #print("haut")
        y=2

    elif hat[1] ==-1 :
        y=1
        #print("bas")

    if hat[0] ==1 :
        #print("droite")
        x=2

    elif hat[0] ==-1 :
        #print("gauche")
        x=1
    if mon_joystick.get_button(9) == 1:
        #print("R2")
        y=1

    print("-----")

    command=str(x)
    ser.write(command.encode('utf-8'))
    line=ser.readline().decode('utf-8').rstrip()
    print("ligne direction: " +line)

    try:
        line3 = result_stack.get_nowait()
    except Empty:
        pass
    #command_capt='!'+str(y)
    #ser_capt.write(command_capt.encode('utf-8'))
    #line3=ser_capt.readline().decode('utf-8').rstrip()
    #print("ligne capteur: " +line3)

    command2=str(y)+str(vmax)+str(line3)
    ser2.write(command2.encode('utf-8'))
    line2=ser2.readline().decode('utf-8').rstrip()
    print("ligne moteur: "+line2)

else:
    print("Vous n'avez pas branché de Joystick...")

```