MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE

LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU

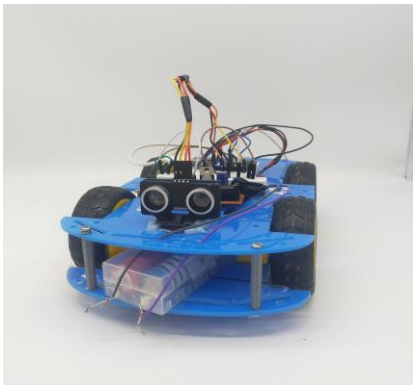
Faculté de Génie électriques & informatique

Département d'informatique

Thème :

Conception et réalisation d'un Rover

Cas : détection d'obstacles



Encadré par :

- PR. LALAM Mustapha.

Réalisé par :

- SEHAKI Sofiane
- MONIR Marzouk

Promotion : 2022/2023

Table des matières :

1 Introduction Générale

La technologie évolue d'un instant à l'autre et les appareils deviennent de plus en plus intelligents grâce au développement des composants électroniques comme les microcontrôleurs qui sont aujourd'hui présents dans les objets connectés (les véhicule autonome, avion, satellites…).

Dans de nombreux secteurs d'activités comme l'astronomie, l'agriculture, la sécurité, ils sont recours au système mobile autonome pour effectuer des taches dans une zone ou la présence humaine est impossible et difficile.

Un système mobile autonome peut engendrer des problèmes Comme la présence d'un obstacle dans son environnement comme qui provoque un disfonctionnement au niveau du système.

Pour pallier à ce problème on na cours à utiliser les systèmes temps réelles.

Les système temps réelle se différencient des autres systèmes informatiques, par la prise en compte des contraintes temporelles, dont le respect est aussi important que l'exactitude du résultat.

Dans ce TP nous allons réaliser et concevoir un rover autonome

Qui détecte l'obstacle et prendre un contrôle total de celle-ci.

La première partie de ce TP concerne la partie conception : elle contiendra les composants, leurs interfaces et les logiciels à utilisées les taches.

Dans la deuxième partie concerne la partie réalisation : on n'y trouve les différentes étapes de réalisation du système, le câblage et le code source.

2. La partie conception

2.1 **Introduction :**

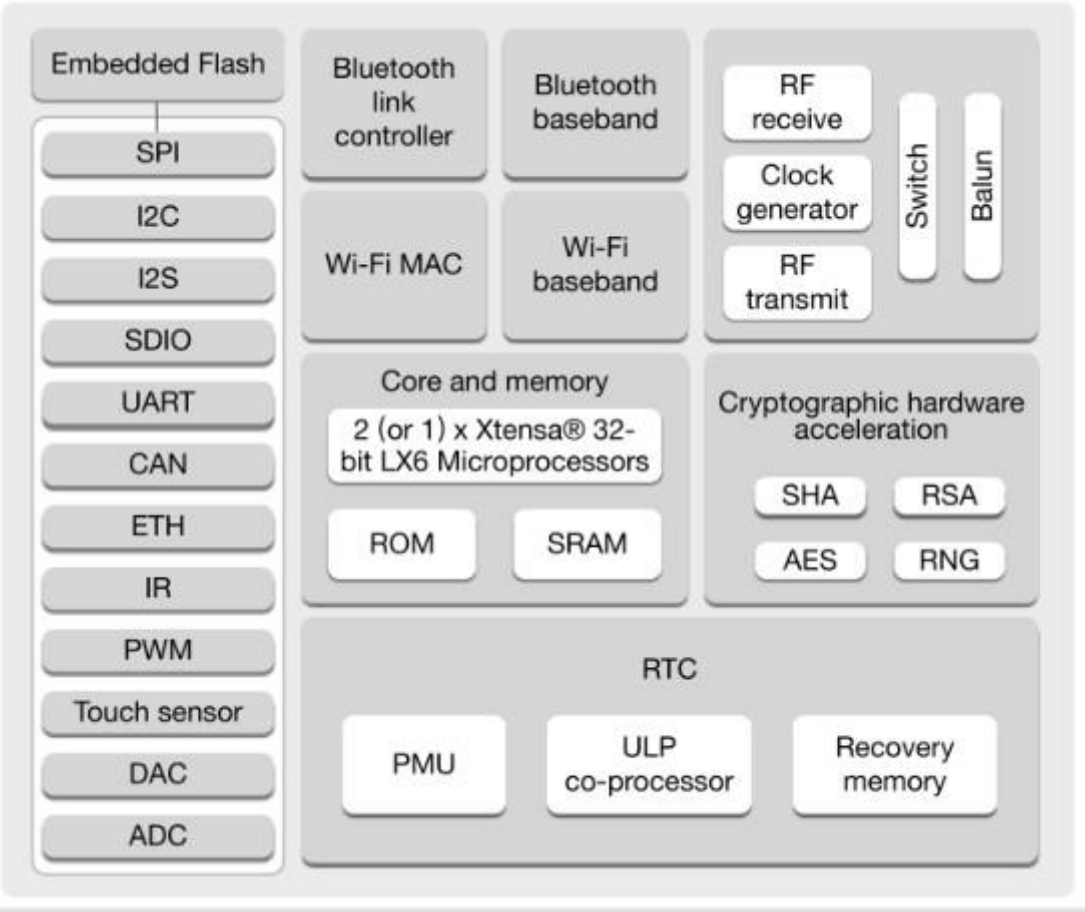
Dans cette partie nous allons discuter sur les composants électroniques nécessaire à une bonne fonctionnalité et à la réalisation de taches pour le rover

2.2 **Composants et périphériques à utilisés :**

❖ ESP32 DEVKIT :

Est une carte de développement compatible à l'environnement ARDUINO, utilisée pour réaliser des prototypes d'applications temps réels comme le rover ou le dispositif de surveillance, doté d'un microcontrôleur duel core Et 38 pins d'entrés sortis, elle intègre la gestion du WIFI et du Bluetooth standard et le Bluetooth LOWEnergy BLE.

❖ Caractéristique de l’ESP32 DEVKIT :



Processeur : dual-core 32-bit LX6 microprocessor, allant jusqu’à 600 MIPS.

Mémoire :

- ✓ SRAM : 520 Ko
- ✓ ROM : 448 Ko

Périphériques et interfaces :

- ✓ 34 ports GPIO avec fonctions alternatives (multiplexés)
- ✓ 18 canaux ADC à 12 bits
- ✓ 2 DACs à 8 bits
- ✓ Capteur tactile
- ✓ De multiples Interfaces série (2xSPI, 2xI2C, et 3xUART)
- ✓ Une interface pour bus CAN

- ✓ Une interface pour carte SD
- ✓ Une interface Ethernet
- ✓ Une interface pour moteur (PWM)
- ✓ Un capteur magnétique

Connectivité Wifi :

- ✓ Le module Wifi peut gérer 4 interfaces virtuelles, supportant la norme
- ✓ 802.11 b, g, n
- ✓ Prise en charge des mode Station, AP, et le mode Promiscuous
- ✓ Antenne intégrée

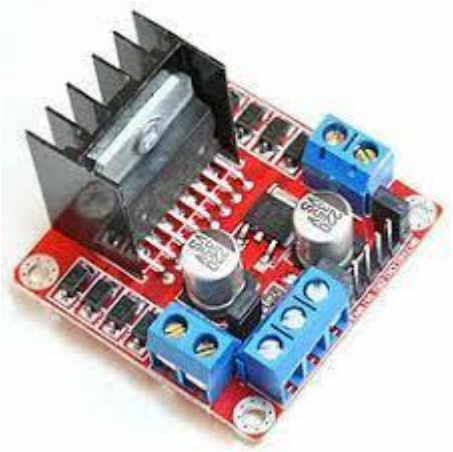
Connectivité Bluetooth :

- ✓ Compatible avec la norme Bluetooth 4.2
- ✓ Gestion des topologies Piconet et Scaternet
- ✓ Supporte les connexions simultanées
- ✓ Utilisations simultanée des modes découvrable et scan

Fonctions de sécurité :

- ✓ Cryptage de code dans la mémoire flash
- ✓ Accélérateur matériel pour les fonctions cryptographiques
- ✓ Supporte le cryptage AES, RSA et ECC
- ✓ Supporte le Hashage SHA-2
- ✓ Générateur de nombres aléatoires

❖ L298 :



C’est un composant électronique qui permet de contrôler avec précision la vitesse et de la direction des moteurs à courant continu grâce à ses pins d’entrées sortis.

Les Broches de connexion :

IN1, IN2 : deux entrées pour contrôler la direction du premier moteur.

IN3, IN4 : deux entrées pour contrôler la direction du deuxième moteur.

EN1, EN2 : deux entrés pour contrôler la vitesse.

12V,5V : deux entrées d’alimentation pour fournir la puissance nécessaire à l’exécution des différentes commandes aux moteurs.

GND : une masse de l'alimentation.

❖ Fiche capteur HC-SR04 :

Le capteur HC-SR04 utilise les ultrasons pour déterminer la distance d'un objet.
L'écart est d'environ 3 cm avec un objet placé à 2 m, ce qui représente une erreur inférieure à 2 %.

Caractéristiques :

Dimensions :

- ✓ 45 mm x 20 mm x 15 mm

Plage de mesure :

- ✓ 2 cm à 400 cm

Résolution de la mesure :

- ✓ 0.3 cm

Angle de mesure efficace :

- ✓ 15 °

Largeur d'impulsion sur l'entrée de déclenchement :

- ✓ 10 µs

Broches de connexion :

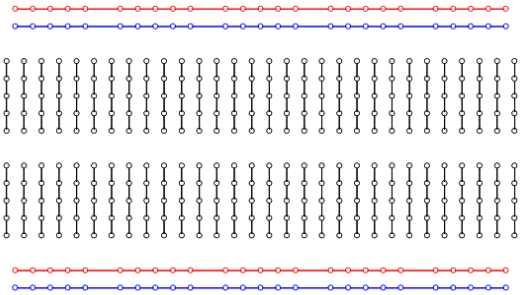
Vcc : Alimentation +5 V DC

Trig : Entrée de déclenchement de la mesure (Trigger input)

Echo : Sortie de mesure donnée en écho (Echo output)

GND : Masse de l'alimentation

❖ BREAD BOARD :



Un bread board est un outil pratique pour le prototypage et la création de circuits. Il vous offre la flexibilité de connecter librement et rapidement des composants électroniques afin de tester des idées et de créer des prototypes. De plus, il est facile et peu coûteux à configurer et à utiliser.

❖ Files électrique :

Les files de liaison sont utilisés pour connecter deux points dans un circuit électronique, ce qui facilite la circulation du courant d'une certaine manière.

❖ Les DC moteurs :

Est un dispositif électrique électromagnétique qui convertit le courant électrique direct en mouvement, il peut être intégré à des objets plus complexe tels que les rovers.

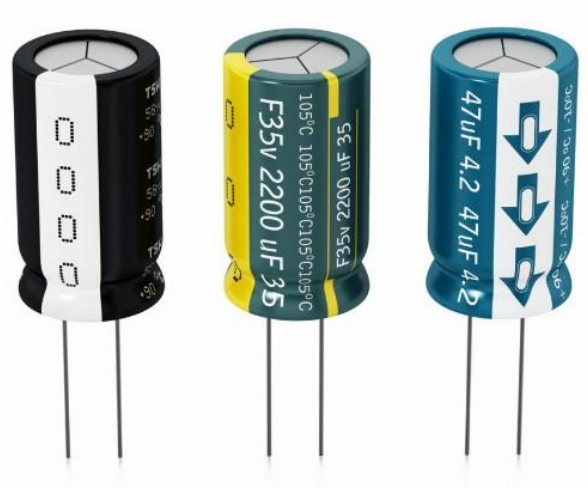
- ❖ Les piles 3.7 v Li-ion :

C’est une ressource d’énergie limité rechargeable nécessaire pour alimenter et offrir une haute capacité et une longue durée de vie pour les composants électroniques du rover.

❖ Régulateur 5 volts :

Le régulateurs 5 volts est un dispositif qui convertit un courant électrique de tension plus élevée en courant électrique de tension plus basse.

❖ Les condensateurs :



C’est un composant électronique qui offre deux fonctionnalités, il fonctionne comme une résistance et comme alimentation à l’absence d’une source d’énergie, elle est recommander d’utiliser deux

condensateurs à l'entrée et à la sortie du régulateur de ce dernier, elle est recommandée par les concepteurs des régulateurs.

2.3 La politique d'ordonnancement :

Les taches dans notre programme s'exécutent d'une façon parallèle et pour les faire fonctionner en coopération, on va utiliser le mécanisme de synchronisation de freeRtos

2.4 Les taches à utilisées :

- La tache surveillance : Elle est consacrée pour surveiller tous s'il y'a un obstacle à 50 cm et prendre une décision sur la direction à prendre (droite ou gauche) dans le cas où il y'a un obstacle, dans le cas où il n y'en a pas d'obstacle le rover continue d'avancer.

- La tache capter : qui permet de sauvegarder la distance entre le rover et son environnement dans une variable.

- La tache avancer : elle permet de faire avancer le rover.

- La tache droite : elle permet de diriger le rover à droite.

- La tache gauche : elle permet de diriger le rover à gauche.

2.5 Les logiciels à utilisées :

- Visual studio : c'est un éditeur de texte qui permet de programmer des applications de différent langage de programmation grâce à un catalogue très riche des extensions parmi elle la plateforme IO qu'on va utiliser dans la réalisation de ce projet.
- Plateforme IO : c'est une extension de visuel studio qui permet de programmer différent types de carte de développement comme ESP32 et ARDUINO grâce à ses interfaces très faciles à utilisées.

2.6 Conclusion :

Dans cette partie nous avons parlés sur la conception du hardware et du software du système et dans la partie qui suit nous allons entamer de la partie réalisation.

3 La partie réalisation

3.1 **introduction** :

Dans cette partie, nous allons entamer la mise à n'ouvre de notre projet, pour cela, nous allons définir le schéma de câblage et le code source.

3.2 **Software** :

- Les bibliothèques :

```
#include <Arduino.h>
#include <freertos/event_groups.h>
#include <freertos/Task.h>
#include <BLEDevice.h>
```

Arduino.h : est un fichier d'en-tête dans l'IDE Arduino qui fournit aux développeurs une bibliothèque de fonctions et de constantes prédéfinies et des timers, Il est également connu sous le nom de « couche d'abstraction matérielle Arduino » et aide les développeurs à s'éloigner des détails de bas niveau du matériel.

Quelques fonctions utilisées de Arduino.h sont :
PinMode : pour définir le mode de la pine (entré,sortie,les deux en meme temps).
digitalWrite : pour écrire une valeur logique sur une pine spécifique de l'ESP32.
PulseIn : pour mesurer une valeur d'un signale logique.

BLEDevice.h : est un fichier d'en-tête fourni par Arduino pour une utilisation avec l'API Arduino BLE. Il contient des

fonctions et des classes utilisées pour configurer, contrôler et interagir avec les appareils Bluetooth Low Energy (BLE).

Quelques fonctions utilisées de BLEDevice.h sont :

init() : pour initialier le bluetooth.
createserver() : pour créer le serveur.
createservice() : pour créer le service associe au serveur.

freertos.h : est un fichier d'en-tête fourni par FreeRTOS. Il contient des prototypes de fonctions, des structures de données et des macros liées aux tâches utilisées pour créer et utiliser des tâches, des files d'attente, des sémaphores et d'autres objets RTOS.

Quelques fonctions utilisées de freertos.h sont :

xTaskCreate, xTaskCreatePinnedToCore : pour créer une tache.
xTaskDelete () : pour supprimer les taches.
xEventGroupWaitBits, xEventGroupSetBits : pour synchroniser une tache à l'attente de l'évènement.

- Définition des pins :

```
//deffiniton des pins
#define  motor3Pin1 21
#define  motor3Pin2 22
#define  enable3Pin 4
#define  motor4Pin1 13
#define  motor4Pin2 25
#define  enable4Pin 12
#define  PIN_TRIG 5
#define  PIN_ECHO 23
```

Dans cette partie on n'a défini les pins grâce à la primitive « difine », Pour nous faciliter la tâche de mémorisation lors de la programmation.

❖ Création des flags des bits d'évènement du rover :

```
//les flags des bites d'evenement du rover
#define BIT_stop (1<<0)// 1 decaler de 0 pour controler le la tache STOP
#define BIT_droite (1<<1)// 1 decaler de 1 pour controler le la tache DROITE
#define BIT_gauche (1<<2)// 1 decaler de 2 pour controler le la tache GAUCHE
#define BIT_avancer (1<<3)// 1 decaler de 3 pour controler le la tache AVANCER
#define BIT_contourner (1<<4)// 1 decaler de 4 pour controler le la tache CONTOURNER
#define BIT_communication (1<<5)// 1 decaler de 5 pour controler le la tache COMMUNICATION
#define BIT_capter (1<<6)// 1 decaler de 6 pour controler le la tache CAPTER
#define BIT_surveillance (1<<7)// 1 decaler de 7 pour controler le la tache SURVEILLANCE
```

Dans cette partie du code on n'a défini les flags D'évènement du rover pour donner un sens aux évènements pour ne pas faire d'erreurs lors de l'implémentation de l'évent groupe.

❖ Les paramètres PWM :

```
//parametre pwm
const int16_t freq = 30000;
const int16_t pwmChannel = 5;
const int16_t resolution = 8;
```

La résolution : elle est fixée à 8 bits pour permettre 255 de niveau de puissance.
La fréquence : on n'a fixé la fréquence du signal PWM à 30 kHz.
PwmChannel : fon n'a fixé le channel à 5.

❖ Déclaration des TaskHandel_t :

```
TaskHandle_t stopH;//pointe vers la tache STOP
TaskHandle_t capterH;//pointe vers la tache CAPTER
TaskHandle_t avancerH;//pointe vers la tache AVANCER
TaskHandle_t droiteH;//pointe vers la tache DROITE
TaskHandle_t gaucheH;//pointe vers la tache GAUCHE
TaskHandle_t communicationH;//pointe vers la tache COMMUNICATION
TaskHandle_t surveillanceH;//pointe vers la tache SURVEILLANCE
```

Dans cette partie on n'a déclarer des pointeurs vers les taches, pour nous permettre de les supprimer.

❖ Déclaration de la classe Callbacks :

```
#define customService BLEUUID((uint16_t)0x1700)//definir L'ID du bluetooth
BLECharacteristic customCharacteristic(BLEUUID((uint16_t)0x1A00), BLECharacteristic::PROPERTY_R
class MyCharacteristicCallbacks: public BLECharacteristicCallbacks {//prendre en charge les eve
  void onWrite(BLECharacteristic *customCharacteristic){// pour prendre en charge les donnees r
    std::string rcvString = customCharacteristic->getValue();// recuperer la donnee recus

    if(rcvString.length()==1){//si la taille du message egale 1
      value=rcvString[0];// affecter la valeur recu du bluetooth vers la variable value
      if (value == 'd')//demarrer le rover
      {
        vTaskResume(surveillanceH);//pour debloquer la tache surveillance
      }
      if (value == 's')// arreter le rover
      {
        xEventGroupSetBits(rover , BIT_stop);//pour debloquer la tache stop
      }
    }
  }
};
```

On n'a déclaré cette classe pour gérer l'évènement de réception du message à l'instant du déclanchement de ce dernier grâce à la méthode onWrite(), ou on n'a tester la donnée reçu si elle est égale à 'd' on va reprendre la tache surveillance pour que le rover puisse redémarrer, dans le cas où la donnée est égale à 's' on débloque la tache STOP.

❖ La tache avancer () :

```
void avancer (void *parametre){
  for(;;){ //pour executer d'une maniere infinie
    xEventGroupWaitBits(rover ,BIT_avancer,pdTRUE,pdFALSE,portMAX_DELAY);//pour bloquer la tach
    //les moteurs à droite en avant
    digitalWrite(motor3Pin1, LOW);
    digitalWrite(motor3Pin2, HIGH);
    // les moteurs à gauche en avant
    digitalWrite(motor4Pin1, HIGH);
    digitalWrite(motor4Pin2, LOW);
    ledcWrite(pwmChannel, vitesse); // generer un signal caree
  }
}
```

C’est une tâche qui permet de faire avancer le rover, au début elle est en état bloqué, en attendant que la tache surveillance la débloquer, pour ensuite envoyer un signal logique et analogique au pont H grâce à la fonction digitalWrite() et ledcWrite().

❖ La tache stope () :

```
void stop(void *parametre){
  for (;;){// pour executer d'une maniere infinie
    xEventGroupWaitBits(rover ,BIT_stop,pdTRUE,pdFALSE,portMAX_DELAY);// pour bloquer la tache
    // arreter les moteurs gauches et droites
    digitalWrite(motor3Pin1, HIGH);
    digitalWrite(motor3Pin2, HIGH);
    digitalWrite(motor4Pin1, HIGH);
    digitalWrite(motor4Pin2, HIGH);
    vTaskSuspend(surveillanceH);//bloquer la tache surveillance
  }
}
```

C’est une tâche qui permet d’arrêter le rover, au début elle est bloquée, en attendant qu’une commande soit envoyer par l’utilisateur, par la suite elle va exécuter digitalWrite(), et suspendre la tache surveillance avec la commande vTaskSuspend(), pour arrêter le rover.

❖ La tache droite () :

```
void droite(void *parametre){
  for (;;){// pour executer d'une maniere infinie
    xEventGroupWaitBits(rover ,BIT_droite,pdTRUE,pdFALSE,portMAX_DELAY);//bloquer la tache DROIT
  // les moteurs droite en arrière
  digitalWrite(motor3Pin1, LOW);
  digitalWrite(motor3Pin2, HIGH);
  // les moteurs gauche en avant
  digitalWrite(motor4Pin1, LOW);
  digitalWrite(motor4Pin2, HIGH);
  ledcWrite(pwmChannel, vitesse);
  }
}
```

C’est une tâche qui permet de tourner le rover à droite, au début elle est en état bloqué, en attendant que la tache surveillance la débloquer, pour ensuite envoyer un signal logique et analogique au pont H grâce à la fonction digitalWrite() et ledcWrite().

❖ La tache gauche () :

```
void gauche(void *parametre){
  for (;;){ // pour executer d'une maniere infinie
    xEventGroupWaitBits(rover ,BIT_gauche,pdTRUE,pdFALSE,portMAX_DELAY);// bloquer la tache GAU
    //mettre les moteurs à droite en avant
    digitalWrite(motor3Pin1, HIGH);
    digitalWrite(motor3Pin2, LOW);
    //les moteurs gauche en arrière
    digitalWrite(motor4Pin1, HIGH);
    digitalWrite(motor4Pin2, LOW);
    ledcWrite(pwmChannel, vitesse);
  }
}
```

C’est une tâche qui permet de tourner le rover à gauche, au début elle est en état bloqué, en attendant que la tache surveillance la débloquer, pour ensuite envoyer un signal logique et analogique au pont H grâce à la fonction digitalWrite() et ledcWrite().

❖ La tache surveillance () :

```
void surveillance (void *parametre){
  vTaskDelay(100);//le temps de réinitialisation du capteur
  etat = 0; //initialiser l'etat par default avancer  ( 0 )
  etat_preced= -1; //initialiser l'etat precedent ( -1 )
  for (;;){//pour executer d'une maniere infinie
    xEventGroupSetBits(rover , BIT_capter);//pour debloquer la tache capter
    xEventGroupWaitBits(rover ,BIT_surveillance,pdTRUE,pdFALSE,portMAX_DELAY);//pour bloquer la
    if(distance<=50){//tester s'il ya un obstacle à 50 cm
      if(etat != etat_preced){// comparaison entre la tache par defaut et la tache precedente
        etat = random(0,2);//generer une valeure aleatoire entre 0 et 1
      }
      if (etat == 1){// le choix de la direction se fait a droite
        xEventGroupSetBits(rover , BIT_droite);// pour debloquer la tache droite
        etat_preced = etat;// affecter l'etat actuelle à l'etat precedent
      }else{//le choix de la diection se fait à gauche
        xEventGroupSetBits(rover , BIT_gauche);//debloquer la tache gauche
        etat_preced =etat;// affecter l'etat actuelle à l'etat precedent
      }
    }else{//s'il ya pas d'obstacle
      xEventGroupSetBits(rover , BIT_avancer);//debloquer la tache avancer
      etat_preced=-1;//pour faire un autre chix de la direction
    }
  }
}
```

C’est la tâche principale du rover, au début débloquer la tache capter, puis sera bloquer jusqu’à ce que la tache capter la débloque, ensuite un fois la tache surveillance débloquer, teste s’il y’a un obstacle, et décidera aléatoirement de la direction du rover (gauche, droite), si l’obstacle n’est pas présent elle débloque la tache avancer, puis remettre l’état précèdent à -1 pour générer une direction aléatoirement.

❖ La tache capter () :

```
void capter (void *parametre){
  vTaskDelay(100);//le temps de réinitialisation du capteur
  for (;;){//pour executer d'une maniere infinie
    xEventGroupWaitBits(rover ,BIT_capter,pdTRUE,pdFALSE,portMAX_DELAY);// bloquer la tache cap

    digitalWrite(PIN_TRIG, HIGH);//emmition du signal
    delayMicroseconds(10);// un delai d'emmition
    digitalWrite(PIN_TRIG, LOW);//arreter l'emmition

    duration = pulseIn(PIN_ECHO, HIGH);//le temps d'aller retour
    distance = duration/58 ; //calculer la distance, 58 est 1/la vitesse de l'ultratson
    xEventGroupSetBits(rover , BIT_surveillance);//pour debloquer surveillance

  }
}
```

Au début elle est bloquée, ensuite la tache capter envoie une onde ultrason pendant 10 microsecondes, ensuite elle calcule la distance :

$344 * 10^2$ cm -- $1 * 10^6$us

? -- 1 us

$(1*344*10^2) / 10^6 = 344*10\text{-}^4$cm/us

V=d/t

D = (t*v) /2

D =((t/(1/v)) /2

D = t/(1/2v)

1/2v = 1/ ((344*10^-4) * 2) = 58 ,14 (cm^-1*us)

D = t / 58

T : valeur obtenu par la fonction PulseIn.

V : la vitesse de son en cm/us.

344 : la vitesse du son m/s.

2 : le temps d'aller-retour d'une onde ultrason.

❖ Setup :

```
void setup() {
  rover =  xEventGroupCreate();// creation de l'event groupe

  BLEDevice::init("MyESP32");// creer le BLE Device

  BLEServer *pServer = BLEDevice::createServer();// creer le BLE serveur

  BLEService *pService = pServer->createService(customService);// creer le BLE service

  pService->addCharacteristic(&customCharacteristic);// creer le BLE carachteristic

  customCharacteristic.setCallbacks(new MyCharacteristicCallbacks());

  pServer->getAdvertising()->addServiceUUID(customService);

  pService->start();// demmarer le service
  // demmarer la publicité du Bluetooth BLE Device pour que les autres machines
  // puissent reperer le bluetooth
  pServer->getAdvertising()->start();
```

Au debut nous avn créer l'event groupe, ensuite on na configurer le BLE serveur et on l'a démarré

```
// definir les modes
pinMode(PIN_TRIG, OUTPUT);
pinMode(PIN_ECHO, INPUT);
pinMode(motor3Pin1, OUTPUT);
pinMode(motor3Pin2, OUTPUT);
pinMode(enable3Pin, OUTPUT);
pinMode(motor4Pin1, OUTPUT);
pinMode(motor4Pin2, OUTPUT);
pinMode(enable4Pin, OUTPUT);
```

Définition des modes en mode entrée / sortis.

```
ledcSetup(pwmChannel, freq, resolution);// definition du signal carrer pour controler la vite
// pour attacher le signal carrer à la pin
ledcAttachPin(enable3Pin, pwmChannel);
ledcAttachPin(enable4Pin, pwmChannel);
```

On n’a configuré le signal carrer pwm pour Controller la vitesse du rover.

```
// creation des taches
xTaskCreatePinnedToCore(surveillance,"servo_surveillance",1024,NULL,7,&surveillanceH,1);
xTaskCreatePinnedToCore(capter,"capter",1024,NULL,7,&capterH,1);
xTaskCreate(stop,"stop",1024,NULL,7,&stopH);
xTaskCreate(avancer,"avancer",1024,NULL,7,&avancerH);
xTaskCreate(droite,"droite",1024,NULL,7,&droiteH);
xTaskCreate(gauche,"gauche",1024,NULL,7,&gaucheH);
```

Création des taches et association de quelques taches à des cores spécifiques.

❖ Source Code du rover :

```
#include <Arduino.h>
#include <freertos/event_groups.h>
#include <freertos/task.h>
#include <BLEDevice.h>


//deffiniton des pins

#define  motor3Pin1 21
#define  motor3Pin2 22
#define  enable3Pin 4
#define  motor4Pin1 13
#define  motor4Pin2 25
#define  enable4Pin 12
#define  PIN_TRIG 5
#define  PIN_ECHO 23
//les flages des bites d'evenement du rover
#define BIT_stop (1<<0)// 1 decaler de 0 pour controler le la tache STOP
#define BIT_droite (1<<1)// 1 decaler de 1 pour controler le la tache DROITE
#define BIT_gauche (1<<2)// 1 decaler de 2 pour controler le la tache GAUCHE
#define BIT_avancer (1<<3)// 1 decaler de 3 pour controler le la tache AVANCER
#define BIT_contourner (1<<4)// 1 decaler de 4 pour controler le la tache
CONTOURNER
#define BIT_communication (1<<5)// 1 decaler de 5 pour controler le la tache
COMMUNICATION
#define BIT_capter (1<<6)// 1 decaler de 6 pour controler le la tache CAPTER
#define BIT_surveillance (1<<7)// 1 decaler de 7 pour controler le la tache
SURVEILLANCE



//parametre pwm
const int16_t freq = 30000;
const int16_t pwmChannel = 5;
const int16_t resolution = 8;

TaskHandle_t stopH;//pointe vers la tache STOP
TaskHandle_t capterH;//pointe vers la tache CAPTER
TaskHandle_t avancerH;//pointe vers la tache AVANCER
TaskHandle_t droiteH;//pointe vers la tache DROITE
TaskHandle_t gaucheH;//pointe vers la tache GAUCHE
TaskHandle_t communicationH;//pointe vers la tache COMMUNICATION
TaskHandle_t surveillanceH;//pointe vers la tache SURVEILLANCE
```

```
SemaphoreHandle_t s;//declaration d'un semaphore pour la syncronisation de des taches

int duration;//le temps d'aller retour des ondes ultrasons
int distance;//la distance entre l'obsacle et le rover

int8_t etat;//variable pour la gestion du rover
int8_t etat_preced;//l'etat precedent du rover

EventGroupHandle_t rover;//declaration de l'event groupe

#define vitesse 220 //vitesse du rover

BLECharacteristic *pCharacteristic;//represente une information qui est transmise ou recus

bool deviceConnected = false;// l'etat du conectivite de l'esp

char value= 'd';//la variable recu par l'utilisateur initialiser a demarrer

#define customService BLEUUID((uint16_t)0x1700)//definir L'ID du bluetooth
BLECharacteristic customCharacteristic(BLEUUID((uint16_t)0x1A00), BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE);//pour envoyer et recevoir les donnees
class MyCharacteristicCallbacks: public BLECharacteristicCallbacks {//prendre en charge les evenements du Bluetooth
  void onWrite(BLECharacteristic *customCharacteristic){// pour prendre en charge les donnees recus
    std::string rcvString = customCharacteristic->getValue();// recuperer la donnee recus

    if(rcvString.length()==1){//si la taille du message egale 1
      value=rcvString[0];// affecter la valeur recu du bluetooth vers la variable value
      if (value == 'd')//demarrer le rover
      {
        vTaskResume(surveillanceH);//pour debloquer la tache surveillance
      }
      if (value == 's')// arreter le rover
      {
        xEventGroupSetBits(rover , BIT_stop);//pour debloquer la tache stop
      }
    }
  }
};

void avancer (void *parametre){
  for(;;){ //pour executer d'une maniere infinie
```

```
    xEventGroupWaitBits(rover
,BIT_avancer,pdTRUE,pdFALSE,portMAX_DELAY);//pour bloquer la tache avancer


    //les moteurs à droite en avant
    digitalWrite(motor3Pin1, LOW);
    digitalWrite(motor3Pin2, HIGH);
    // les moteurs à gauche en avant
    digitalWrite(motor4Pin1, HIGH);
    digitalWrite(motor4Pin2, LOW);
    ledcWrite(pwmChannel, vitesse); // generer un signal caree
  }
}
void stop(void *parametre){
  for (;;){// pour executer d'une maniere infinie
    xEventGroupWaitBits(rover ,BIT_stop,pdTRUE,pdFALSE,portMAX_DELAY);// pour
bloquer la tache STOP
    // arreter les moteurs gauches et droites
    digitalWrite(motor3Pin1, HIGH);
    digitalWrite(motor3Pin2, HIGH);
    digitalWrite(motor4Pin1, HIGH);
    digitalWrite(motor4Pin2, HIGH);
    vTaskSuspend(surveillanceH);//bloquer la tache surveillance
  }
}
void droite(void *parametre){
  for (;;){// pour executer d'une maniere infinie
   xEventGroupWaitBits(rover
,BIT_droite,pdTRUE,pdFALSE,portMAX_DELAY);//bloquer la tache DROITE

  // les moteurs droite en arrière
  digitalWrite(motor3Pin1, LOW);
  digitalWrite(motor3Pin2, HIGH);
  // les moteurs gauche en avant
  digitalWrite(motor4Pin1, LOW);
  digitalWrite(motor4Pin2, HIGH);
  ledcWrite(pwmChannel, vitesse);
  }
}
void gauche(void *parametre){
  for (;;){ // pour executer d'une maniere infinie
    xEventGroupWaitBits(rover ,BIT_gauche,pdTRUE,pdFALSE,portMAX_DELAY);//
bloquer la tache GAUCHE

    //mettre les moteurs à droite en avant
    digitalWrite(motor3Pin1, HIGH);
    digitalWrite(motor3Pin2, LOW);
    //les moteurs gauche en arrière
    digitalWrite(motor4Pin1, HIGH);
```

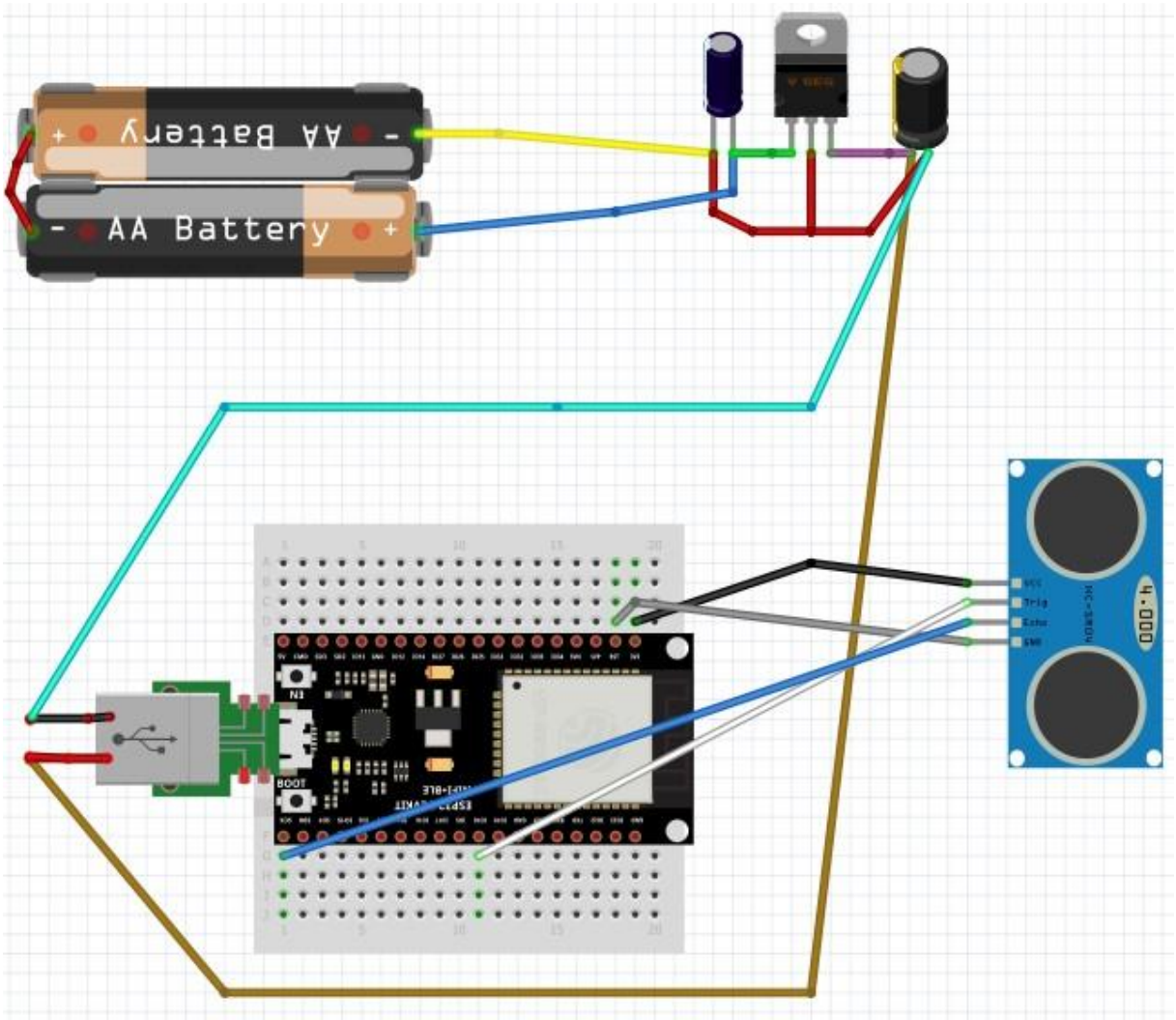
```
    digitalWrite(motor4Pin2, LOW);
    ledcWrite(pwmChannel, vitesse);
  }
}
void surveillance (void *parametre){
  vTaskDelay(100);//le temps de réinitialisation du capteur
  etat = 0; //initialiser l'etat par default avancer  ( 0 )
  etat_preced= -1; //initialiser l'etat precedent ( -1 )
  for (;;){//pour executer d'une maniere infinie
    xEventGroupSetBits(rover , BIT_capter);//pour debloquer la tache capter
    xEventGroupWaitBits(rover
,BIT_surveillance,pdTRUE,pdFALSE,portMAX_DELAY);//pour bloquer la tache
surveillance

    if(distance<=50){//tester s'il ya un obstacle à 50 cm
      if(etat != etat_preced){// comparaison entre la tache par defaut et la
tache precedente
        etat = random(0,2);//generer une valeure aleatoire entre 0 et 1
      }

      if (etat == 1){// le choix de la direction se fait a droite
        xEventGroupSetBits(rover , BIT_droite);// pour debloquer la tache
droite
        etat_preced = etat;// affecter l'etat actuelle à l'etat precedent
      }else{//le choix de la diection se fait à gauche
        xEventGroupSetBits(rover , BIT_gauche);//debloquer la tache gauche
        etat_preced =etat;// affecter l'etat actuelle à l'etat precedent
      }

    }else{//s'il ya pas d'obstacle
      xEventGroupSetBits(rover , BIT_avancer);//debloquer la tache avancer
      etat_preced=-1;//pour faire un autre chix de la direction
    }
  }
}
void capter (void *parametre){
  vTaskDelay(100);//le temps de réinitialisation du capteur
  for (;;){//pour executer d'une maniere infinie
    xEventGroupWaitBits(rover ,BIT_capter,pdTRUE,pdFALSE,portMAX_DELAY);//
bloquer la tache capter

    digitalWrite(PIN_TRIG, HIGH);//emmition du signal
    delayMicroseconds(10);// un delai d'emmition
    digitalWrite(PIN_TRIG, LOW);//arreter l'emmition

    duration = pulseIn(PIN_ECHO, HIGH);//le temps d'aller retour
    distance = duration/58 ; //calculer la distance, 58 est 1/la vitesse de
l'ultratsons
    xEventGroupSetBits(rover , BIT_surveillance);//pour debloquer surveillance
```

```
  }
}
void setup() {
  rover =  xEventGroupCreate();// creation de l'event groupe

  BLEDevice::init("MyESP32");// creer le BLE Device

  BLEServer *pServer = BLEDevice::createServer();// creer le BLE serveur

  BLEService *pService = pServer->createService(customService);// creer le BLE
service

  pService->addCharacteristic(&customCharacteristic);// creer le BLE
carachteristic

  customCharacteristic.setCallbacks(new MyCharacteristicCallbacks());//
ajouter l'objet des Callbacks à la caractéristique pour prendre en charge les
evenements des receptions des messsages

  pServer->getAdvertising()->addServiceUUID(customService);// ajouter l'id du
service

  pService->start();// demmarer le service
  // demmarer la publicité du Bluetooth BLE Device pour que les autres
machines
  // puissent reperer le bluetooth
  pServer->getAdvertising()->start();

  // definir les modes
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
  pinMode(motor3Pin1, OUTPUT);
  pinMode(motor3Pin2, OUTPUT);
  pinMode(enable3Pin, OUTPUT);
  pinMode(motor4Pin1, OUTPUT);
  pinMode(motor4Pin2, OUTPUT);
  pinMode(enable4Pin, OUTPUT);

  ledcSetup(pwmChannel, freq, resolution);// definition du signal carrer pour
controler la vitesse du rover
  // pour attacher le signal carrer à la pin
  ledcAttachPin(enable3Pin, pwmChannel);
  ledcAttachPin(enable4Pin, pwmChannel);

  // creation des taches
  xTaskCreatePinnedToCore(surveillance,"servo_surveillance",1024,NULL,7,&surve
illanceH,1);
  xTaskCreatePinnedToCore(capter,"capter",1024,NULL,7,&capterH,1);
```

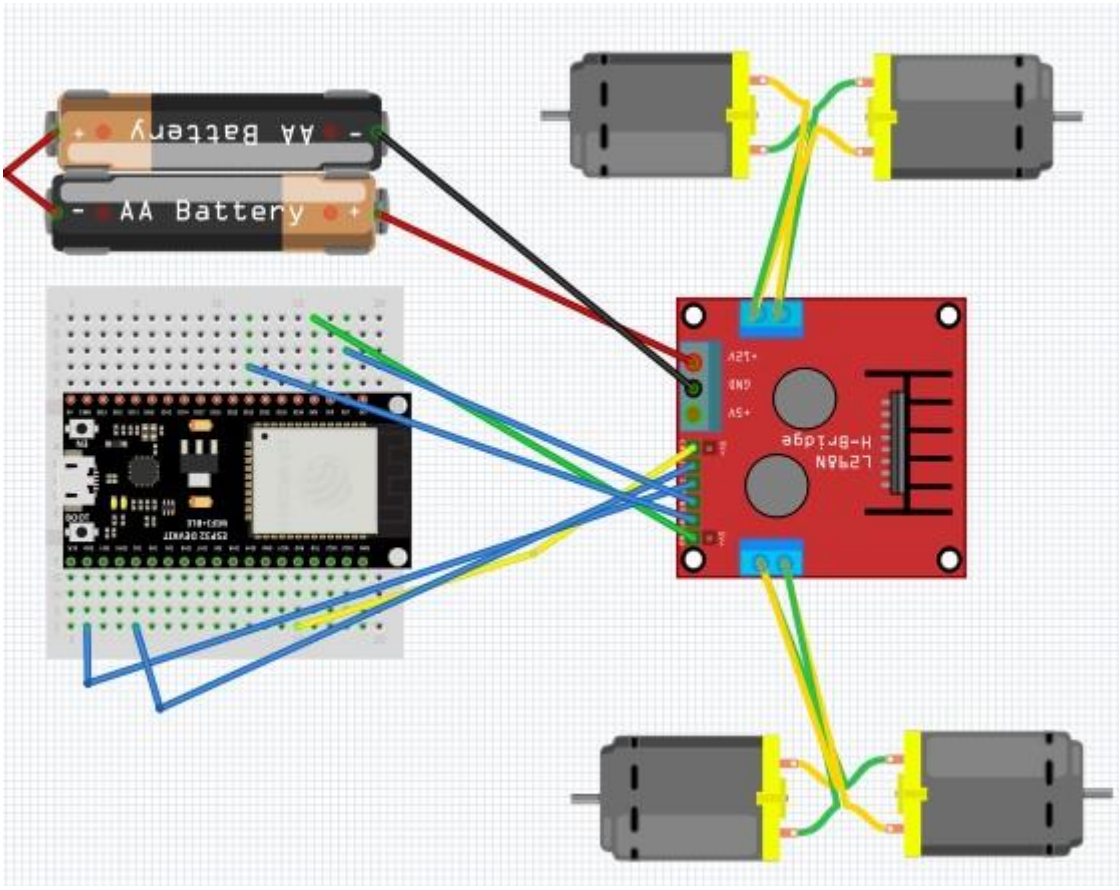
```
  xTaskCreate(stop,"stop",1024,NULL,7,&stopH);
  xTaskCreate(avancer,"avancer",1024,NULL,7,&avancerH);
  xTaskCreate(droite,"droite",1024,NULL,7,&droiteH);
  xTaskCreate(gauche,"gauche",1024,NULL,7,&gaucheH);

}

void loop (){}
```

3.2 **Hadrware** :

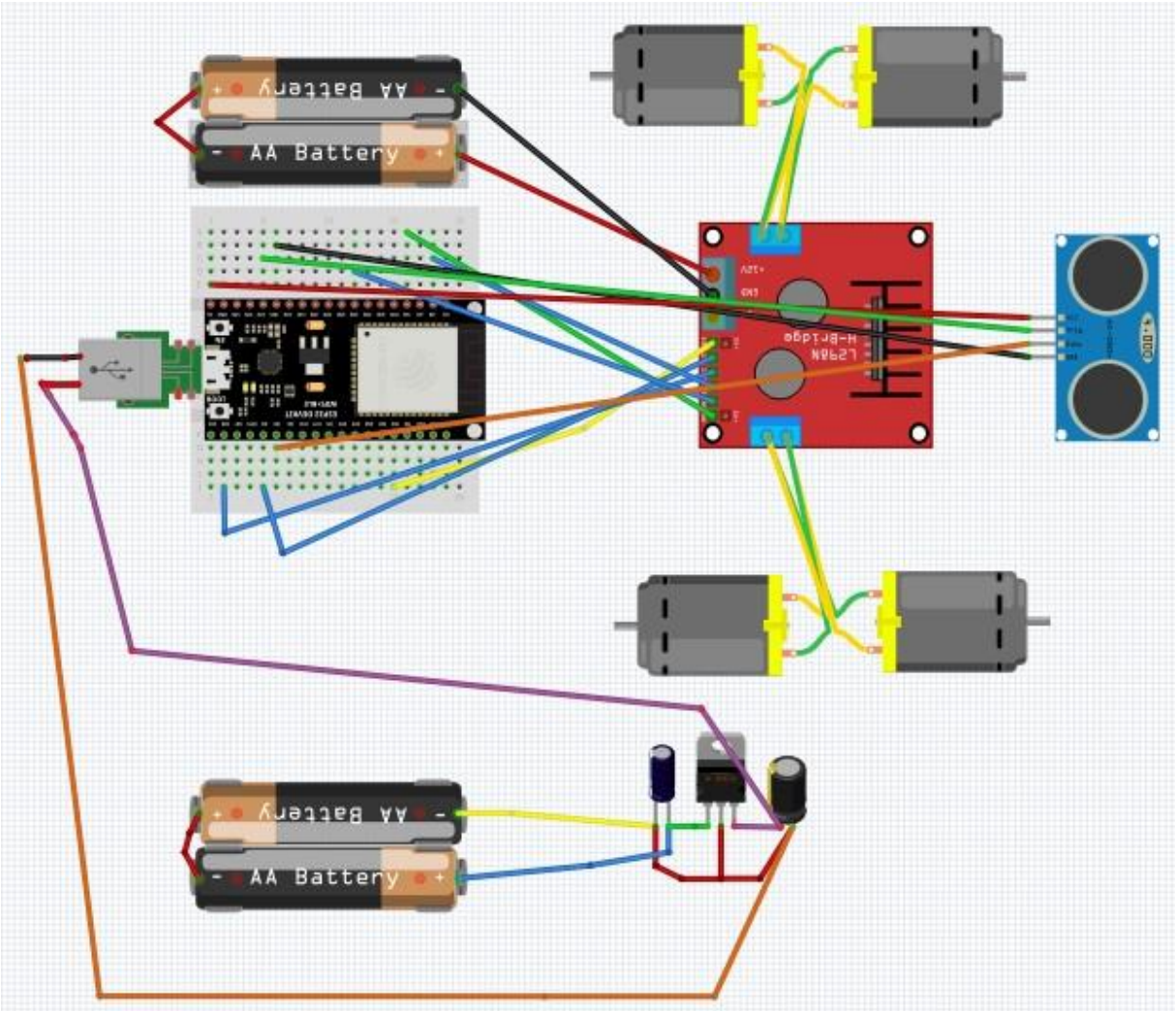
❖ Schéma de câblage entre ESP32, radar, piles, régulateur 5V :



❖ Schéma de câblage entre ESP32, DC motors, L298 :

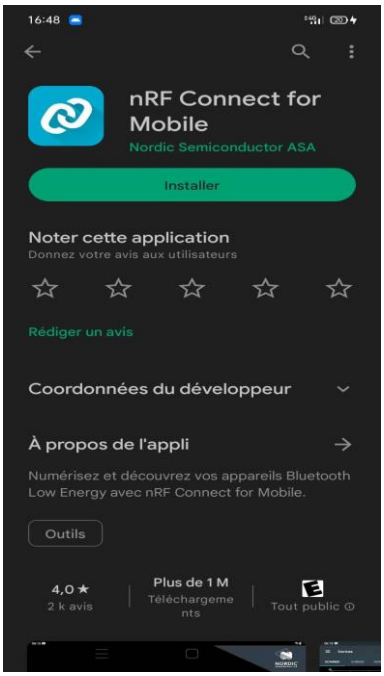


❖ Schéma globale du rover :

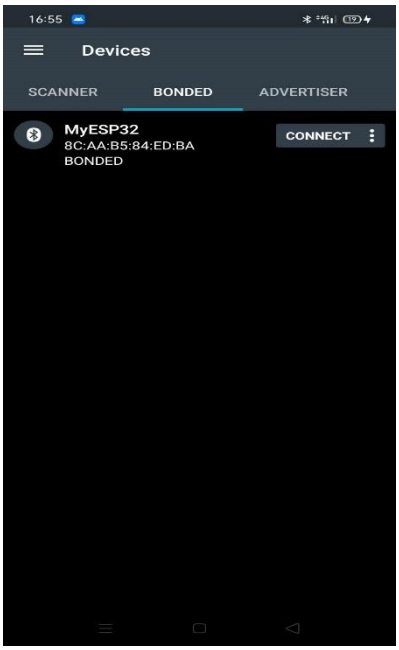


3.4 Interaction avec l'utilisateur :

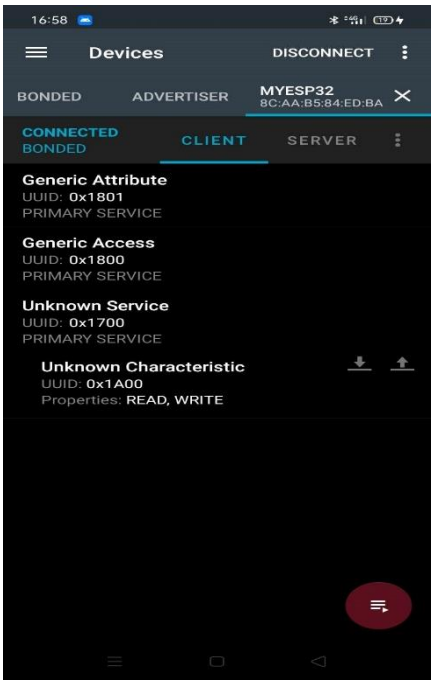
- Applications utilisée pour Controller le rover à distance : L'application à utiliser : nRF Connect, téléchargable depuis le Play store on n'accédons à ùce lien : https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp



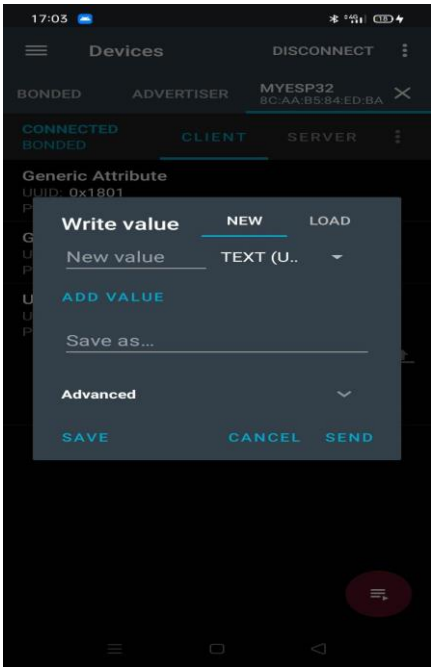
- Pour connecter l'esp, on n'accède à BONDED, puis on clique sur CONNECT :



❖ Une fois connecter à l’esp32 on sera automatiquement rediriger vers l’onglet MYESP32 :



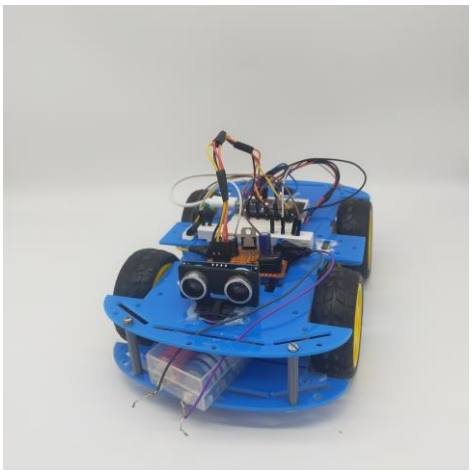
❖ Puis On clique sur Service et on envoie un caractère « s » pour arrêter et « d » pour démarrer :



3.5 Conclusion :

Dans cette partie de Hardware nous avons discutés la phase de réalisation de deux niveaux, niveau software ou on n'a expliqué en détails le code source, et Hardware ou on n'a spécifié le schéma de câblage de notre système embarqué.

5. Conclusion générale :



Dans ce projet on n'a appris beaucoup de principes sur la programmation temps réels tel que la manipulation des taches et leurs synchronisations, et sur les systèmes embarqués par exemple la conception d'un système à faible cout qui donne des résultats exactes et précis, après cette étape on va intégrer dans ce rover des capteurs et des dispositifs de réalisation pour surveiller des zones hostiles.