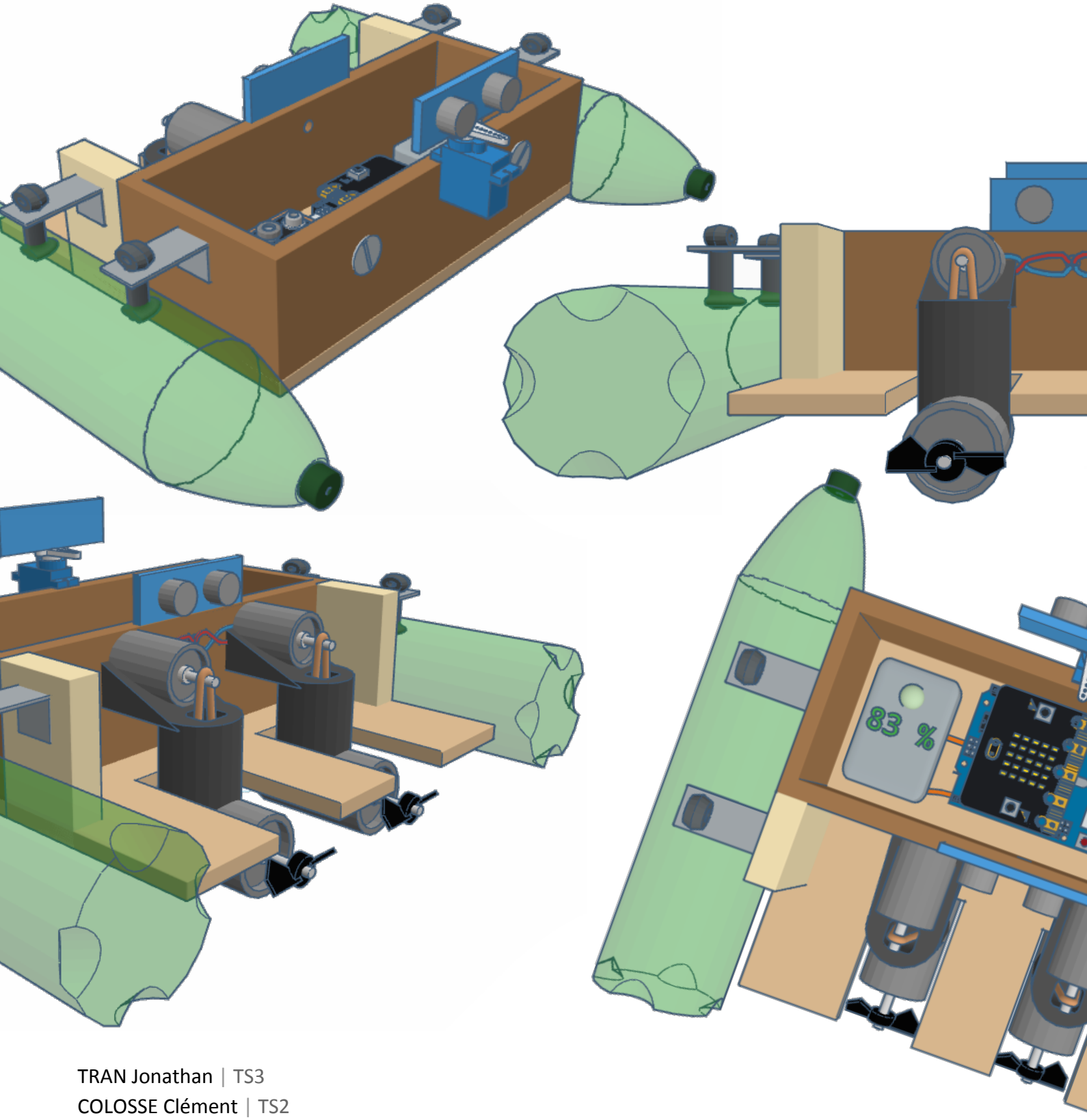
Projet ISN 2019

Construction d'un bateau téléguidé



TRAN Jonathan | TS3
COLOSSE Clément | TS2
DELPLACE Alexis | TS2

Sommaire

I. Présentation

L'équipe

Le groupe que nous avons formé afin de réaliser notre projet final d'ISN est composé de 3 élèves : COLOSSE Clément, TRAN Jonathan, et moi-même DELPLACE Alexis, élèves de Terminale S2 et S3 de l'établissement Lycée Louise Michel de Gisors.

Le projet

Enjeux et problématiques

Notre objectif initial était de fabriquer de toutes pièces un objet concret afin d'allier robotique et programmation cela nous permettant la découverte de la programmation de microcontrôleur. De part la nature de notre projet, nous devions répondre aux problématiques actuelles que sont le respect de l'environnement et la réduction des coûts.
Afin que notre projet s'inscrive dans le cadre du développement durable, nous avons choisi de fabriquer notre production principalement à base d'objets de récupération.

Positionnement du projet

Notre première idée fut de fabriquer une voiture à laquelle nous aurions rajouté un système électronique lui permettant d'être téléguidée. Cependant, nous nous sommes aperçus que ce projet avait été réalisé de multiples fois et qu'il était trop simple de trouver des tutoriels sur internet ce qui nous aurait amené à faire du simple copier-coller.
Puis nous avons eu l'idée d'un bateau qui, en plus d'être original, reprenait les objectifs du projet initial. En effet le nombre de tutoriels sur les bateaux était bien moindre que sur les autres types de véhicules comme des quadricoptères ou des voitures.
De plus, contrairement à notre bateau, les projets existants ne prenaient pas en compte l'utilisation de la de nombreux objet recyclés ainsi que la réduction des coûts puisque de nombreux projets utilisent des kits préfabriqués.

II. Analyse du besoin et recherches d'idées

Cahier des charges

Fabriquer un bateau téléguidé qui doit :

- Se déplacer dans toutes les directions
- Éviter les obstacles automatiquement
- Être téléguidé via une télécommande
- Suivre un itinéraire GPS
- Sauvegarder son itinéraire via GPS

Langage et matériel utilisés

La base

Afin de réaliser notre projet, nous avons décidé d'utiliser un Arduino Uno, cette carte électronique comportant un microcontrôleur qui peut être programmé afin de produire et analyser des signaux électriques ce qui permet de réaliser diverses tâches. Nous avons choisi cette carte puisque c'est la plus répandue dans le monde du DIY électronique, par conséquent, il existe de nombreuses ressources sur celle-ci. De plus, elle est peu coûteuse et a l'avantage d'être sous licence libre, ce qui permet à d'autres entreprises de la commercialiser, on peut ainsi trouver des copies de cette carte à moins de 10€, ce qui a permis à chaque membre de l'équipe de s'en procurer une et de pouvoir travailler en parallèle les uns des autres sur les tâches qui nous étaient affectées en dehors des séances d'ISN. Pour ce qui est du langage, l'arduino en possède un qui lui est propre, il était donc logique d'utiliser celui-ci.

Les composants

Une fois le cahier des charges défini, nous avons dû trouver le moyen réaliser les différentes tâches définies dans celui-ci.
Le premier problème a été de trouver un moyen afin que le bateau détecte les obstacles. Il existe trois types de capteurs permettant de mesurer une distance :

- Capteur infrarouge
  Portée: 0,1 à 0,8 m | Précision: N/A | Prix: 16,80 €

- Lidar
  Portée: 0,3 à 12 m | Précision: 1 cm | Prix: 42,00€

- Capteur à ultrasons
  Portée: 0,02 à 3,5 m | Précision: 1 cm | Prix: 5,95 €

Au vu des caractéristiques techniques et de son prix, le capteur à ultrasons est le plus approprié à notre projet.
Le deuxième dilemme auquel nous avons dû faire face est la choix de la technologie de télécommunication. Il existe là aussi trois types de communication :

- RF (Radio Fréquence)
  Portée: 100 m | Consommation: N/A | Prix: 13,90 €

- WIFI
  Portée: 150 m | Consommation: 130 mA | Prix: 15,15 €

- Bluetooth Low Energy (BLE)
  Portée: 50 m | Consommation: 8,5 mA | Prix: 20,80 €

Nous avons décidé de ne pas retenir la RF puisque le bateau devait pouvoir être téléguidé depuis un smartphone, ce dernier ne possédant pas de module RF. Nous avons finalement retenu le bluetooth (BLE) puisque la consommation électrique d'un module bluetooth est bien plus faible que celle d'un module WIFI. De plus, le bluetooth est beaucoup plus simple à mettre en place par rapport au WIFI.
En plus de cela, nous avons dû acheter un servomoteur ainsi qu'un contrôleur de moteur. Les moteurs ont quant à eux été récupérés sur une ancienne imprimante.

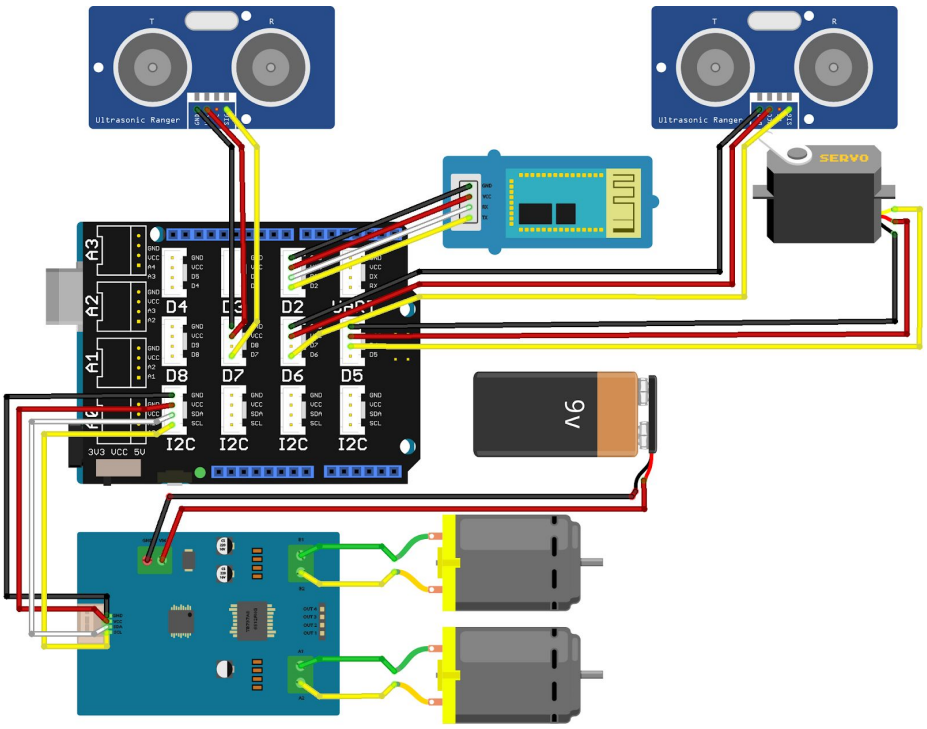


Schéma du circuit électrique - (RP)

La structure

Afin de réaliser la structure, nous avons utilisé des matériaux de récupération, tels que du bois, des bouteilles en plastique et d'autres pièces métalliques (équerres, visserie, etc.). Seules les hélices ont été achetées.

III. Répartition des tâches et démarche collaborative

Planning

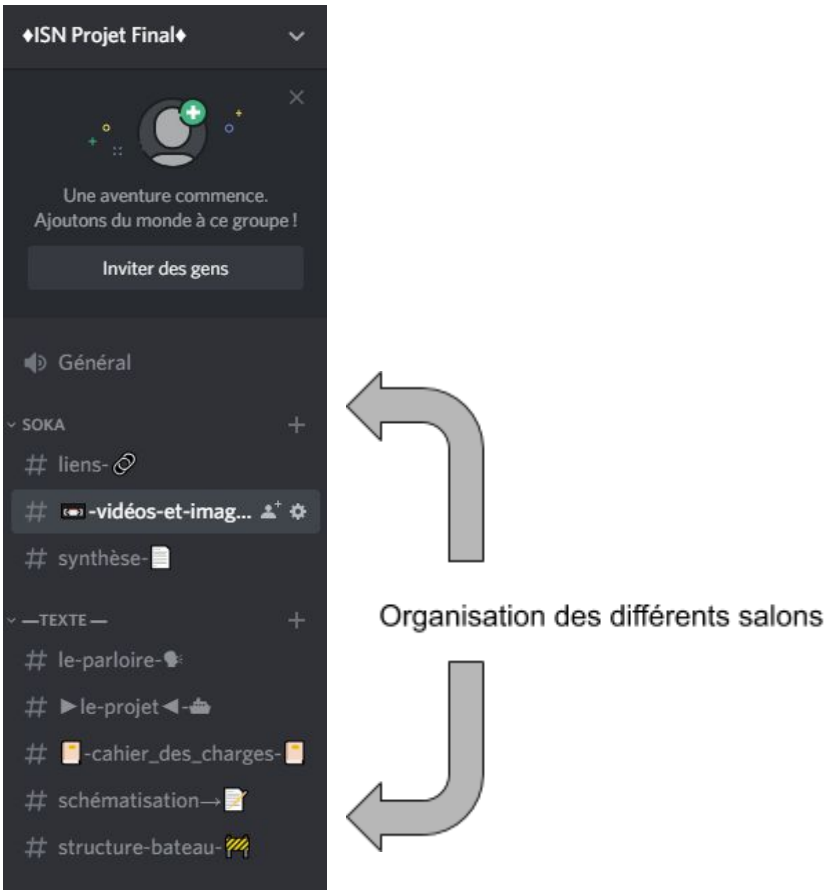
Du dimanche 20 au mardi 30 janvier	Aperçu général du projet (idées, perspectives, cahier des charges)
Du samedi 9 au lundi 25 février	Conception des plans du bateau et du circuit électrique.
Du samedi 6 au mardi 23 avril	Phases de développement du code.
Du samedi 27 avril au mardi 14 mai	Fabrication de la structure du bateau
Du mercredi 29 mai au lundi 3 juin	Finalisation du projet.

Tâches

Tâches / Nom	**TRAN Jonathan**	**DELPLACE Alexis**	**COLOSSE Clément**
Schéma du bateau	✓	✓	✓
Schéma électrique du bateau	✗	✓	✗
Construction du bateau	✓	✓	✓
Schéma de la télécommande	✗	✗	✗
Schéma électrique de la télécommande	✓	✗	✗
Fabrication de la télécommande	✗	✗	✗
Programmation de la télécommande	✗	✗	✗
Capteur à ultrasons	✗	✓	✓
Bluetooth	✗	✓	✗
Moteur	✓	✓	✗
Modélisation de pièces 2D/3D	✗	✗	✓
GPS (retour à la maison)	✗	✗	✗
GPS (suivre un itinéraire (.gpx))	✗	✗	✗
Led rouge pour le détecter la nuit	✗	✗	✗

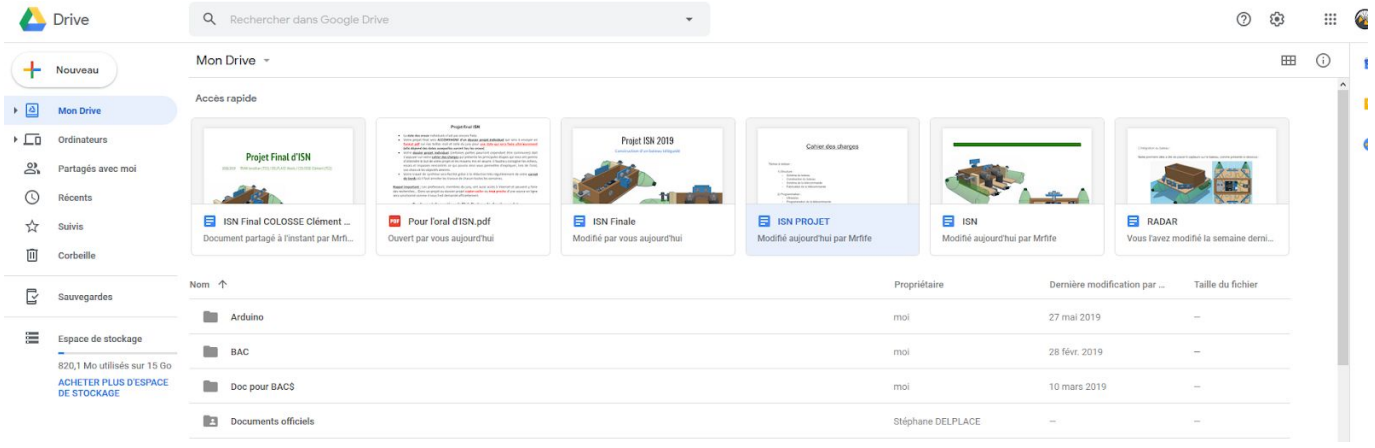
Moyens de mise en commun

Le logiciel Discord nous a permis de créer des “salons” structurés par thème afin de mieux s'organiser et d’échanger en temps réel.



Capture d’écran de Discord

Le logiciel de traitement de texte Google Docs nous a permis de rédiger les différentes parties du dossier en simultané, de plus chacun avait la possibilité de faire des commentaires ou proposer des modifications sur le travail des autres afin de l’améliorer et le compléter.



Capture d’écran de Google Drive

Enfin Google Drive nous à permis de partager les différents fichiers nécessaires à la réalisation du dossier final.

IV. Réalisation

Les capteurs à ultrasons

Le fonctionnement

La première tâche à laquelle je me suis attelé est la programmation des capteurs à ultrasons. J'ai commencé par me renseigner sur la méthode de fonctionnement de celui-ci, que je vais détailler dans cette partie.

Capteur à ultrasons - (Source)

Le capteur comporte 4 pins, mais seulement 3 sont utiles dans notre cas. Il y a 2 pins d'alimentation, la pin GND (noire) correspond à la masse et la pin VCC (rouge) correspond à l'entrée 5V. La dernière, pin SIG (jaune) pour signal permet la communication entre le capteur et l'arduino.

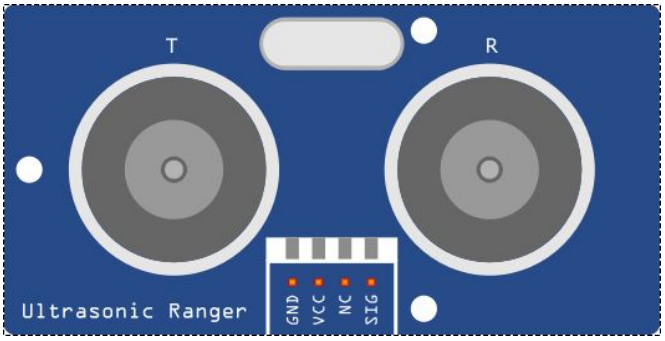


Schéma du capteur à ultrasons - (Source)

Cette communication s'opère via un signal PWM (Pulse With Modulation), c'est un signal généré grâce à un courant électrique. Par exemple, on applique une tension de 5V pendant 3 secondes puis à 0V pendant 3 secondes et ainsi de suite, ce qui donne un signal carré comme celui-ci :

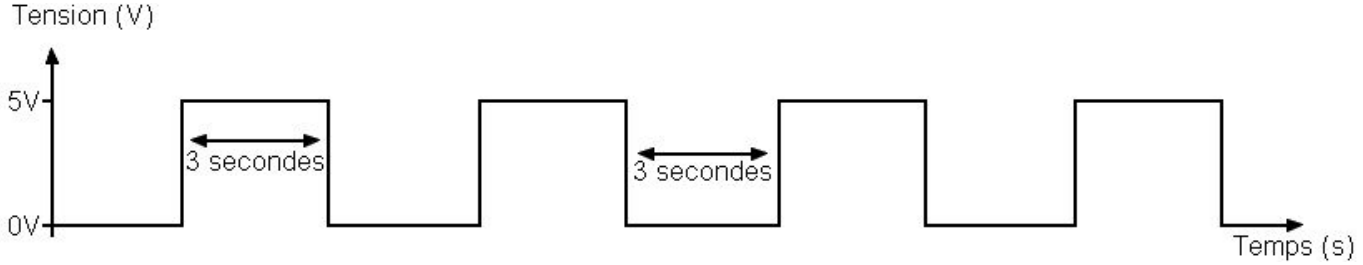


Schéma signal PWM - (RP)

Le principe de fonctionnement du capteur est plutôt simple :

- L'arduino envoie via la pin SIG une impulsion de 10 µS (que l'on appelle trigger)

Schéma signal trigger - (RP)

- Suite à celui-ci, le capteur envoie une salve de 8 impulsions avec une fréquence de 40 kHz via l'émetteur (noté T) et le récepteur (noté R) s'active.

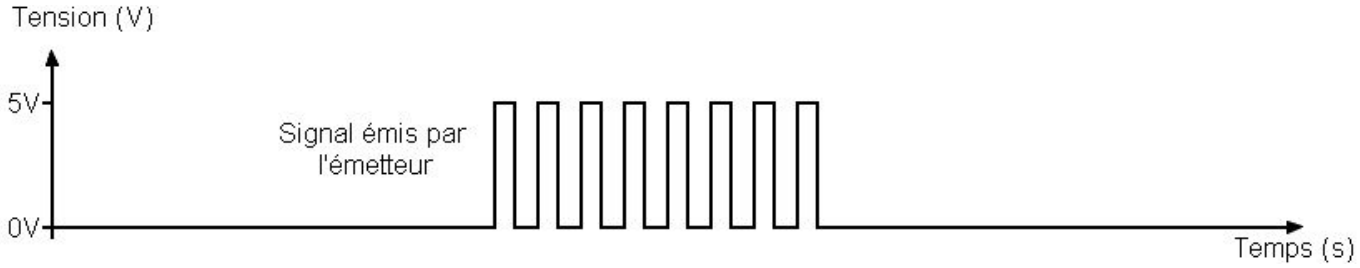


Schéma signal émetteur - (RP)

- S'il y a un obstacle, l'onde envoyée rebondit sur celui-ci et est captée par le récepteur. Le capteur à ultrasons renvoie par la pin SIG une impulsion (que l'on appelle echo) d'une durée égale à celle que l'onde a prise pour être émise puis captée par le capteur.

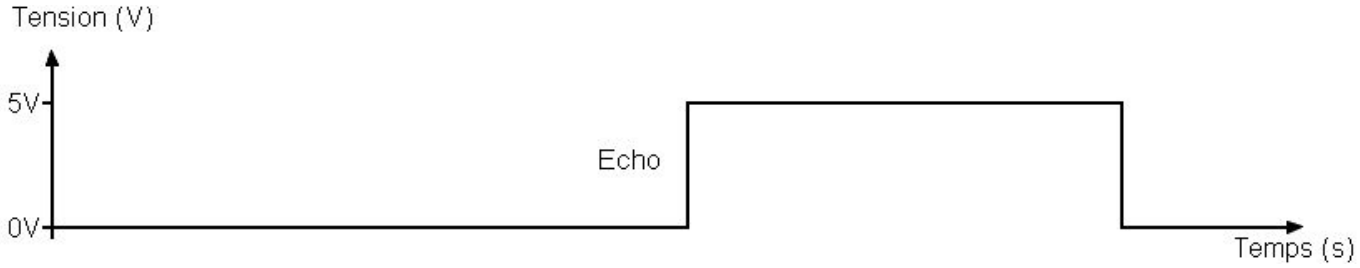


Schéma signal écho - (RP)

La distance entre le capteur et l'obstacle s'obtient donc par la formule suivante :

Distance = (Durée du signal reçue) x (Vitesse du son dans l'air) / 2

NB : On divise par 2 car l'onde fait un aller-retour entre l'émetteur et le récepteur.

Le code

Une fois toutes ces connaissances assimilées, j'ai commencé à coder. Je me suis vite rendu compte après avoir lu quelques tutoriels que la programmation en arduino était très orientée objet. Cet aspect m'a particulièrement intéressé et m'a permis d'utiliser les connaissances acquises lors de la réalisation du projet pour le BAC blanc d'ISN.

L'objet "Capteur Ultrason"

```
1  class Ultrasonic //Capteur Ultrason
2  {
3    private:
4       //Varibales :
5       int pin;
6    public:
7       //Fonctions :
8       Ultrasonic(int select_pin);
9       int Measure();
10 };
```

On remarque qu'il existe 2 types d'attributs en C, les "`private`" et les "`public`". Les attributs de type "`private`" peuvent seulement être appelés par des fonctions propres à l'objet en question. Ici, la variable "`pin`" (l.2) peut être modifiée uniquement par les fonctions "`Ultrasonic`" (l.8) et "`Measure`" (l.9). En revanche ces deux fonctions qui elles sont des attributs de type "`public`" peuvent être appelées par tout le programme.

La fonction "Ultrasonic"

```
1  Ultrasonic::Ultrasonic(int select_pin) //Constructeur
2  {
3    pin = select_pin;
4  }
```

Cette fonction est le constructeur, elle se distingue des autres, car elle porte le même nom que l'objet. Le constructeur s'exécute à chaque création d'un objet. Ici, elle permet d'attribuer une pin à un capteur à ultrasons. Par exemple, lorsque l'on crée un objet "capteur ultrason n°1" avec pour paramètre 5, le programme associe l'objet "capteur ultrason n°1" avec le 5 et par conséquent, toutes communications avec celui-ci se fera via la pin 5.

La fonction “Measure”

```
1  int Ultrasonic::Measure() // Mesure de la distance
2  {
3    //Trigger
4    pinMode(pin, OUTPUT);
5    digitalWrite(pin, LOW);
6    delayMicroseconds(2);
7    digitalWrite(pin, HIGH);
8    delayMicroseconds(10);
9    digitalWrite(pin,LOW);
10
11   //Measure
12   pinMode(pin,INPUT);
13   return ((pulseIn(pin,HIGH)*SOUND_SPEED)/2);
14 }
```

Comme expliqué précédemment afin de faire une mesure à l’aide du capteur ultrason, il faut lui envoyer un signal “trigger”, l’émission de ce signal se découpe en 3 étapes :

- Premièrement, on configure la pin attribué au capteur à ultrasons avec lequel on veut faire une mesure en mode “`OUTPUT`” afin de pouvoir appliquer une tension (l.4).
- Ensuite, on initialise la pin en le positionnant sur son état “`LOW`”, c’est-à-dire qu’on définit la tension sortant du pin à 0V et ceux pendant 2 microsecondes (l.5/6).
- Après cela, on positionne la pin sur son état “`HIGH`” pendant 10 microsecondes (l.7/8), autrement dit, on définit la tension sortant du pin à 5V pendant cette durée. Cette commande permet de déclencher la mesure.
- Pour finir on bascule la pin vers son état initial, c’est à dire “`LOW`” (l.9).

Il ne reste plus qu'à mesurer la durée du signal renvoyé par le capteur et appliquer la formule vue auparavant, pour cela :

- On configure la pin en mode “`INPUT`” afin de recevoir un signal (l.12).
- Puis, on chronomètre la durée du signal reçue grâce à la fonction “pulseIn”(l.13), celle-ci permet de mesurer une durée d’impulsion. Par exemple, si le paramètre est “`HIGH`”, la fonction “`pulseIn`” lance un chronomètre lorsqu’une tension de 5V est appliquée puis arrête le chronomètre lorsque la tension passe à 0V. Pour finir, la fonction renvoie la durée chronométré.
- Enfin, on rentre cette durée dans la formule (l.13).

Après toutes ces commandes exécutées, la fonction “`Measure`” renvoie sous la forme d’un entier la distance qui sépare le capteur à ultrasons d’un obstacle en centimètres.

Le test

Afin de tester mon programme en charge des capteurs à ultrasons, j'ai codé un petit programme permettant d'afficher la distance séparant le capteur d'un obstacle (programme).

Ensuite, j'ai branché un capteur à ultrasons sur l'arduino.

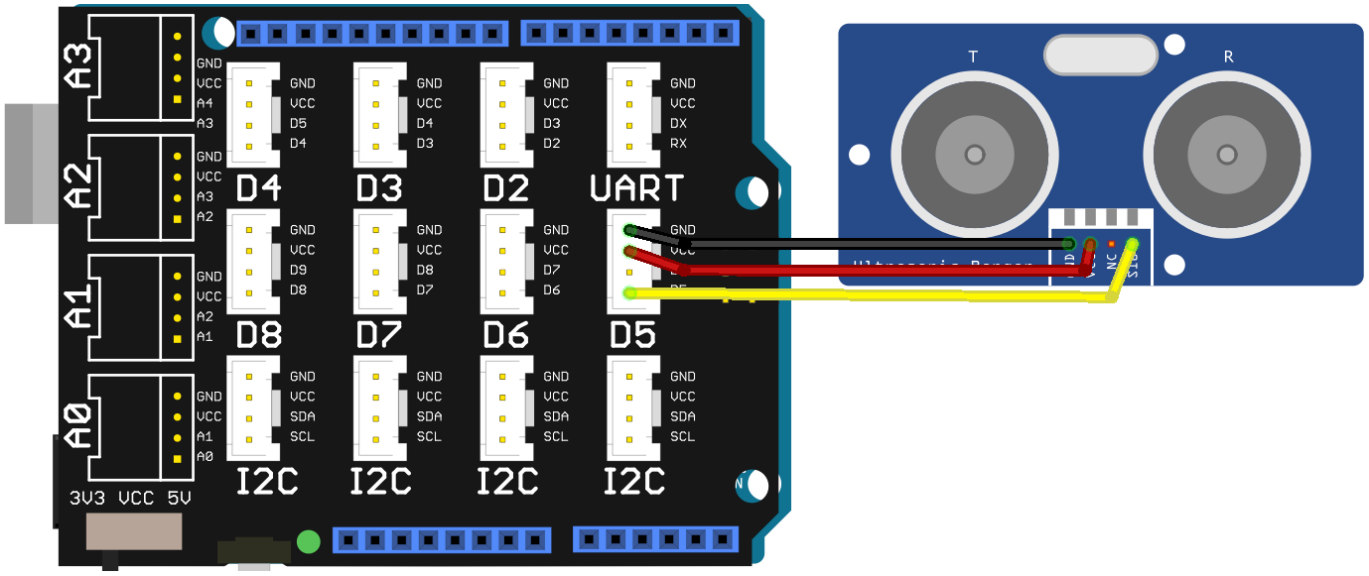


Schéma du montage du capteur à ultrasons - (RP)

Puis, j'ai placé un obstacle à environ 30 cm du capteur et j'ai vérifié si la valeur affichée à l'écran était bonne.

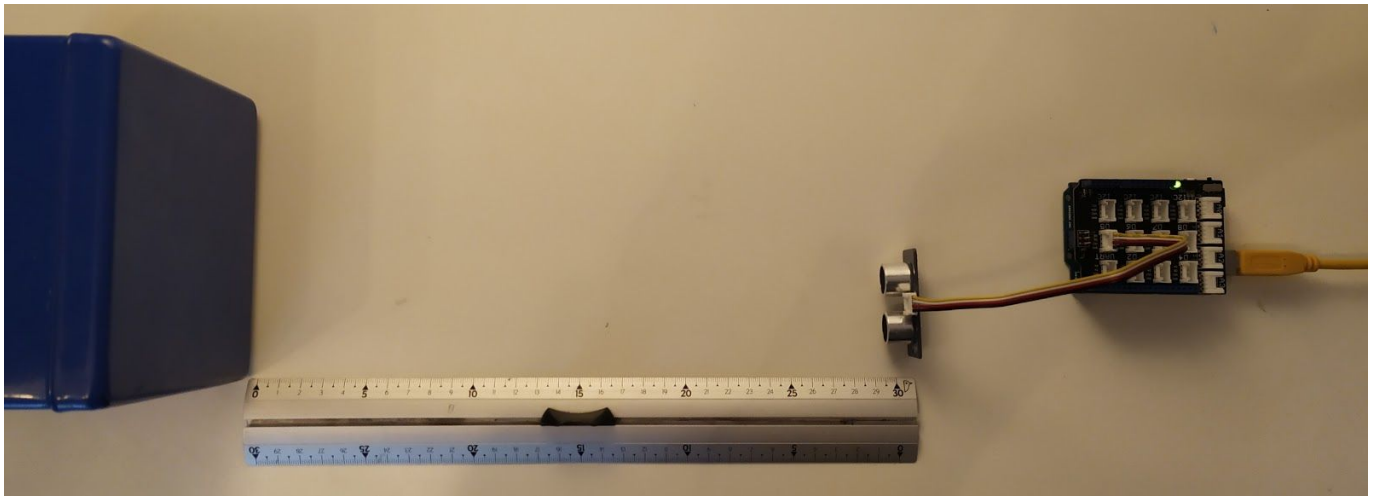


Photo du test

Capture d'écran du moniteur série

Nous obtenons une mesure de 29 cm, nous pouvons donc en conclure que le programme en charge des mesures de distance est fonctionnel.

L'intégration au bateau

Notre première idée a été de placer 6 capteurs sur le bateau, comme présenté ci-dessous :

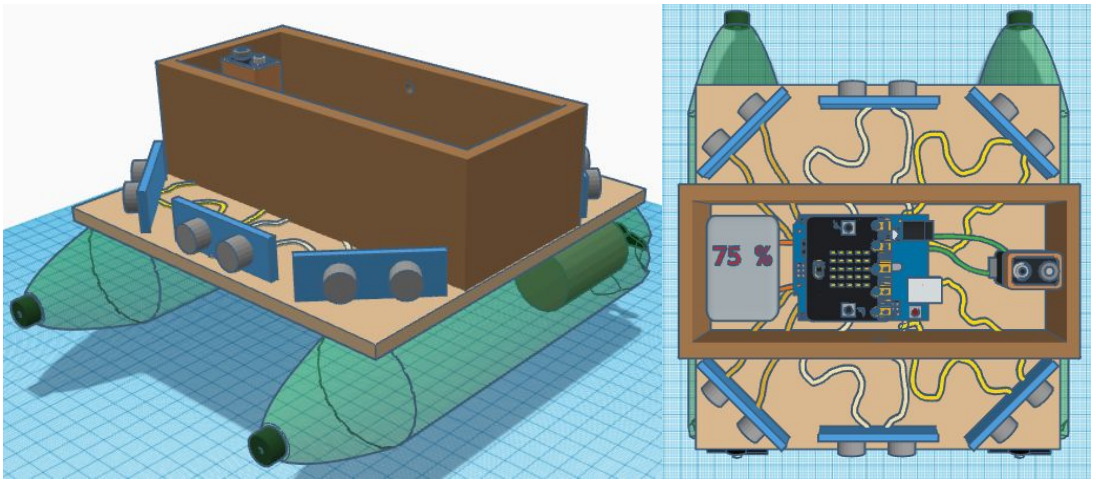


Schéma bateau - (RP)

Le principe de fonctionnement était le suivant : le bateau réagit en fonction du capteur qui a détecté un obstacle. Par exemple, si c'est le capteur avant gauche qui détecte un obstacle, alors le bateau recule pendant 5 secondes et pivote de 45° vers la droite. En revanche, si c'est le capteur avant droite qui détecte un obstacle, alors le bateau recule pendant 5 secondes, puis pivote de 45° vers la gauche.
Le problème que nous avons rencontré a été le fait que l'Arduino ne possède pas assez de connecteurs pour les 6 capteurs et le reste de l'équipement. En effet, le "Base Shield V2" que nous utilisons ne possède que 7 ports digitaux (D suivi d'un nombre sur le schéma). Or deux ports sont déjà utilisés, un est destiné au module bluetooth et l'autre au module GPS.

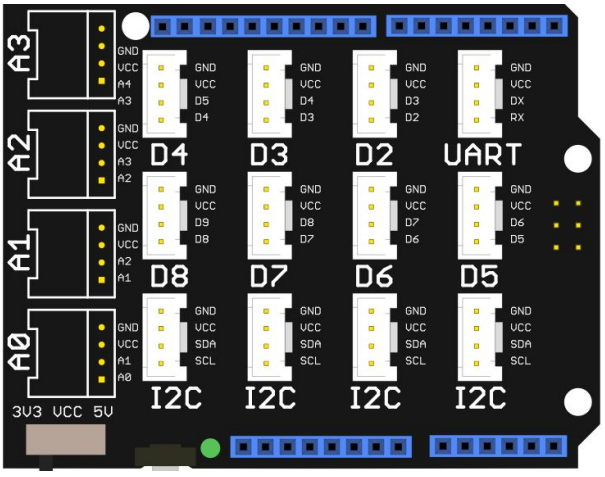


Schéma du shield - (Source)

Pour pallier ce problème, nous avons dû revoir le design de notre bateau. Après quelques recherches sur internet, Clément a eu l'idée d'un radar.

Le radar

Le fonctionnement

Celui-ci consiste en un capteur à ultrasons monté sur un servomoteur :

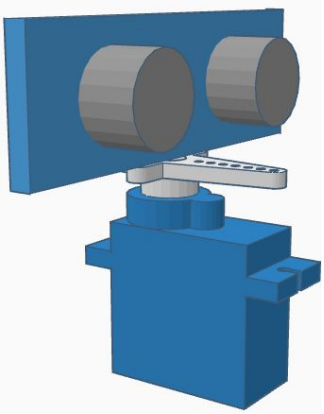


Schéma du radar - (RP)

Le principe de fonctionnement restait à peu près le même que celui que nous avions imaginé mais en utilisant un seul capteur. Le bateau ne réagit plus en fonction du capteur qui détecte l'obstacle mais, de la position du servomoteur sur lequel se trouve un capteur.
Nous avons donc décidé de conserver cette idée. Clément a codé la fonction que j'ai ensuite adaptée pour faciliter son intégration dans le programme général du bateau.

Le code

Nous avons donc utilisé la bibliothèque "`Servo.h`" qui permet la création d'objets "servomoteur" et d'interagir avec eux.

```
1  Servo servo_front;
```

On crée donc un objet "`servo_front`".

```
1  servo_front.attach(SERVO_FRONT);
2  servo_front.write(0);
```

Ensuite, on définit la pin correspondant au servomoteur grâce à la fonction "`attach`" (l.1) ; ici, le paramètre pris par cette fonction est "`SERVO_FRONT`" qui est défini plus haut dans le programme comme étant 5. Il est important de faire ceci afin de ne pas "hardcoder" les valeurs dans le programme et les regrouper dans une zone dédiée via des "`#define`" et permettre ainsi une meilleure maintenabilité du code. La fonction "`write`" quant à elle permet de faire bouger le servomoteur sur 180° ; ici, la valeur prise est 0°. Notre servomoteur est maintenant prêt à être utilisé.

Afin de coder notre système de radar, j'ai créé plusieurs tâches chacune correspondant à une mesure avec un angle différent, le capteur fait une mesure à chaque fois que le servomoteur pivote de 45°.

Schéma de fonctionnement du radar - (RP)

Voici pour exemple, le code pour la tâche correspondant à la mesure à 135° :

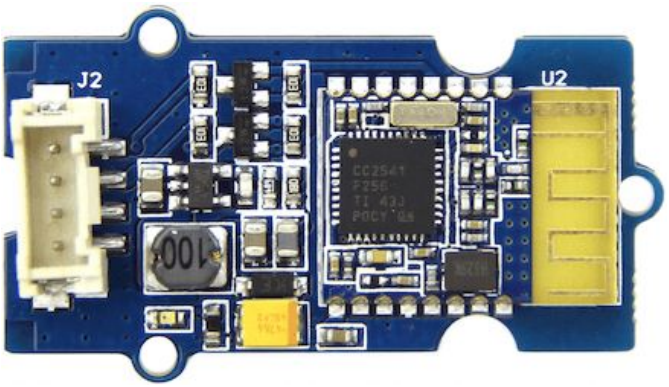
```
1  void task_SM135()
2  {
3    range_front_135 = ultrasonic_front.Measure();
4    delay(25);
5    if (range_front_135 < 50){
6        avoid_forward_135();
7    }
8    servo_front.write(180);
9    delay(150);
10 }
```

Premièrement, nous créons une fonction de type "`void`" (l.1) . Ensuite, nous faisons une mesure de la distance que l'on sauvegarde dans la variable "`range_front_135`" (l.3). Nous appliquons, après cela, un délai de 25 millisecondes afin de laisser au capteur le temps de faire la mesure (l.4). Puis, si la distance mesurée est inférieure à 50 cm, alors le programme appelle la fonction "`avoid_forward_135`" qui fait reculer le bateau, puis le fait tourner de 45°. Enfin le programme fait pivoter le servomoteur jusqu'à 180° afin que le capteur à ultrasons soit dans la bonne position pour la mesure suivante. Un délai de 150 millisecondes est appliqué afin de laisser le temps au servomoteur de pivoter.

Le module Bluetooth

Le fonctionnement

À ce stade, notre robot possède ses fonctions motrices et il est capable d'éviter les obstacles, la prochaine étape consiste donc à pouvoir le piloter à distance. Je me suis donc attelé à la mise en place du bluetooth lui permettant de communiquer via un smartphone.



Module bluetooth - (Source)

Le module bluetooth comporte 4 pins, 2 pins d'alimentation, la pin GND (noire) qui correspond à la masse et la pin VCC (rouge) à l'entrée 5V. Les 2 autres pins servent à la communication entre l'arduino et le module, la pin Rx permet la réception de données et la pin Tx la transmission.

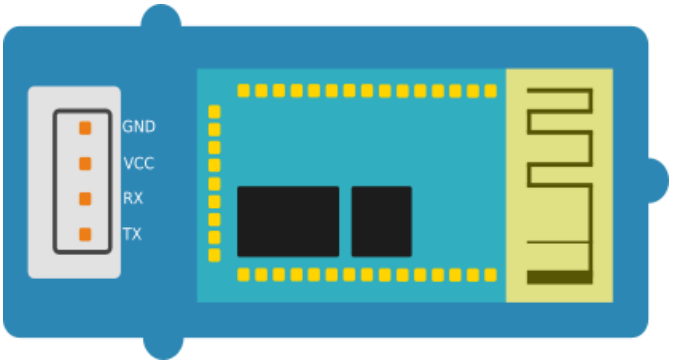


Schéma du module bluetooth - (RP)

Cette communication se fait grâce à une UART (Universal Asynchronous Receiver Transmitter). Cette liaison permet la communication entre un ordinateur et un port série, c'est d'ailleurs ce composant qui permet la liaison entre l'ordinateur et l'arduino lorsqu'on le branche par USB grâce à une émulation UART sur USB.

Les messages transmis sur une liaison UART sont appelés des trames, celles-ci sont composées de plusieurs bits :

- Un bit start, annonçant la fin du message
- Un bit stop, annonçant la fin du message
- Le message constitué d'un octet (8 bits)

Par exemple, si nous voulons transmettre la lettre “a” qui a pour code binaire 10000110, il faudra générer le signal suivant :

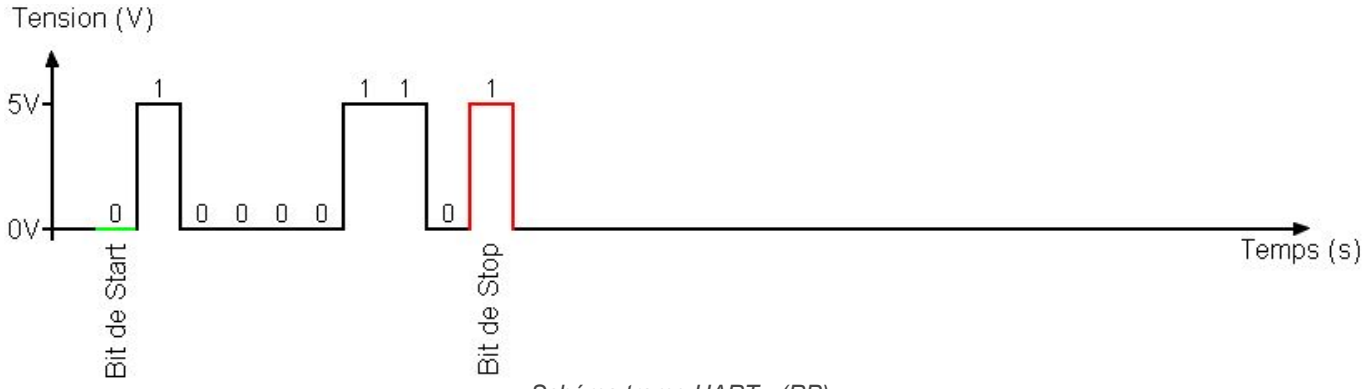


Schéma trame UART - (RP)

Le code

Contrairement aux capteurs ultrasons pour lesquels j’ai défini une classe, ici, nous allons utiliser la bibliothèque “SoftwareSerial.h” qui permet de créer des objets ayant la capacité de communiquer via un port série virtuel. En effet le port série physique présent sur l’arduino est réservé à la communication entre le l’arduino et le port USB . On crée donc un objet “BLE” grâce à la classe “SoftwareSerial” afin de pouvoir communiquer avec le module bluetooth :

```
1  SoftwareSerial BLE(BLE_RX,BLE_TX);
```

Lors de la création de l’objet “BLE”, nous devons préciser les pins correspondant à l’envoi et à la réception de données, ici, ces pins portent les noms “BLE_RX” et “BLE_TX” définis plus haut dans le programme comme étant respectivement les pins 2 et 3.

La fonction “setupBleConnection” :

```
1  void setupBleConnection()
2  {
3    BLE.begin(9600);
4    delay(100);
5    BLE.print("AT+CLEAR");
6    delay(100);
7    BLE.print("AT+ROLE0");
8    delay(100);
9    BLE.print("AT+SAVE0");
10   delay(100);
11   BLE.print("AT+NAMEBATEAU");
12 }
```

Maintenant que l’objet “BLE” correspondant à l'interface UART du module bluetooth est créé, il faut indiquer à l’arduino de commencer la communication avec celui-ci via un port série virtuel. La vitesse de cette communication est défini à 9600 baud, le baud étant une unité qui sert à mesurer la rapidité de modulation des signaux émis et qui correspond à une modulation par seconde (l.3).

Désormais, nous pouvons communiquer avec le module bluetooth, pour cela il existe des commandes prédéfinies qui permettent différentes actions. Ce sont les commandes AT, développées à l'origine pour les modems Hayes, une grande société d'électronique des années 80, celles-ci sont aujourd'hui utilisées dans de nombreux modems. Lorsque le module bluetooth reçoit ces commandes, le microcontrôleur intégré à celui-ci interprète les commandes et agit en conséquence. Voici la liste des commandes utilisées lors de l'initialisation du bluetooth ainsi que leurs fonctions :

- "AT+CLEAR" (l.5) : efface tous les réglages précédents.
- "AT+ROLE0" (l.7) : définit le module bluetooth comme esclave. Dans une connexion bluetooth, il y a un maître et un esclave, le maître donne des ordres à l'esclave qui les exécute.
- "AT+SAVE0" (l.9) : sauvegarde les informations de connexions.
- "AT+NAMEBATEAU" (l.11) : configure le nom diffusé sur par bluetooth

La fonction delay(100) (l.418/6/10) permet de mettre un délai de 100 millisecondes entre chaque commande envoyée afin de laisser le temps au microcontrôleur du module bluetooth de les analyser et d'agir en fonction de celles-ci.

Maintenant que le module bluetooth est configuré, nous pouvons l'utiliser afin de communiquer avec un smartphone ou tout autre appareil ayant un module bluetooth.

La fonction "task_BLE"

```
1  void task_BLE()
2  {
3    if(BLE.available()){
4
5        recvChar = BLE.read();
6        if (recvChar != last_recvChar){
7
8        if(recvChar == 'z'){
9        last_recvChar = 'z';
10       go_forward();
11       }
12
13       if(recvChar == 's'){
14       last_recvChar = 's';
15       idle();
16       }
17     }
18   }
19 }
```

Elle permet la réception de caractères envoyés depuis un smartphone, à chaque caractères nous associerons une fonction motrice.

Nous commençons par créer une fonction de type “void” (l.1). Ensuite nous vérifions si des données ont été reçues par le module bluetooth (l.3). Si c’est le cas, le programme stocke la donnée reçue dans une variable de type char (l.5). Ensuite, si le caractère reçu est différent de celui reçu lors du précédent appel à la fonction, ce qui implique un changement d'état, alors les conditions suivantes sont appliquées (l.6). Par exemple, si le caractère reçu est un “z”, le programme enregistre le dernier caractère reçu comme “z” et appelle la fonction “go_forward” qui permet d’avancer (l.8 à 11).

Caractère	Tâche	Fonction (programme)
“z”	avancer	go_forward
“s”	s'arrêter	idle
“q”	pivoter sur la gauche	go_left
“d”	pivoter sur la droite	go_right
“r”	reculer	go_backward

Le test

De manière à tester mon code, j’ai développé un programme qui affiche le caractère reçu par le bluetooth (programme).

J’ai branché le module bluetooth à l’arduino.

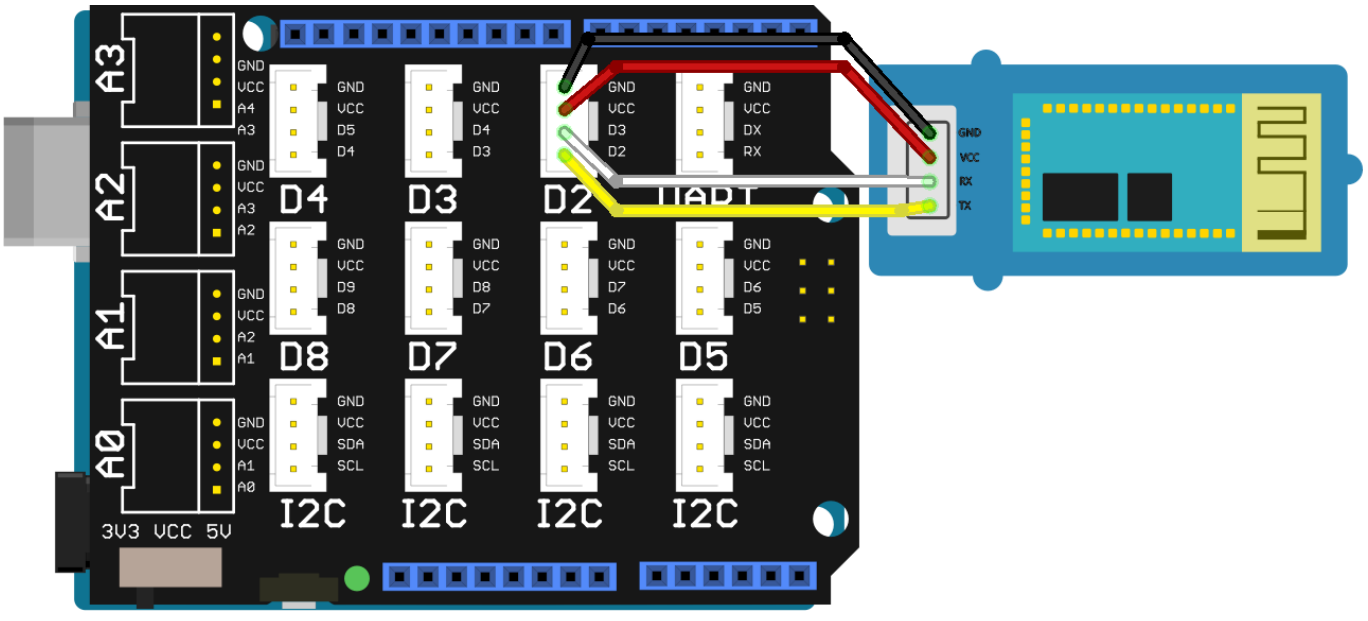
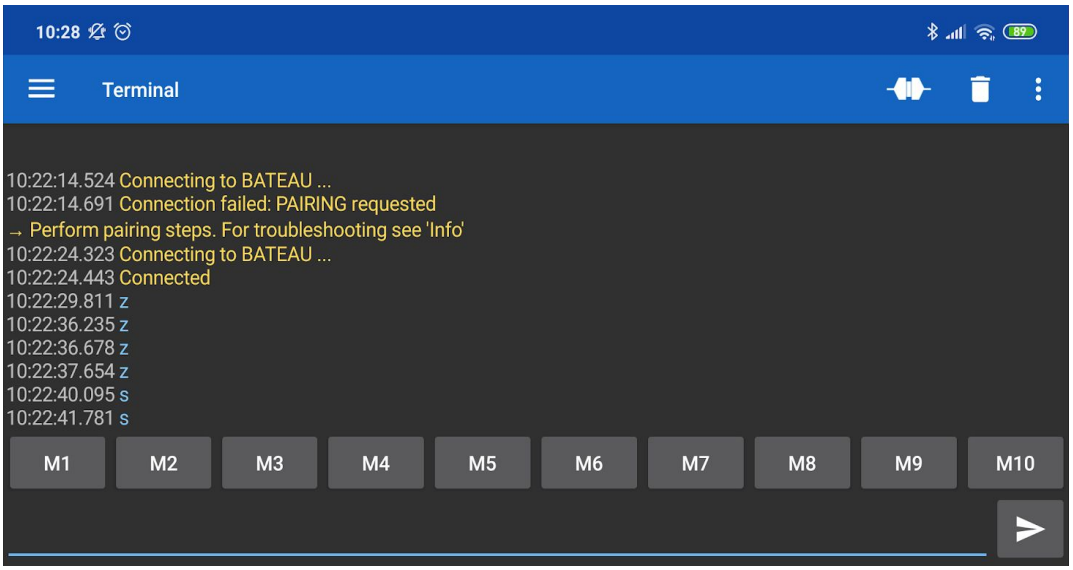


Schéma du montage du module bluetooth - (RP)

J’ai connecté mon smartphone via bluetooth à l’arduino puis j’ai envoyé de caractère à travers le terminal de mon téléphone.

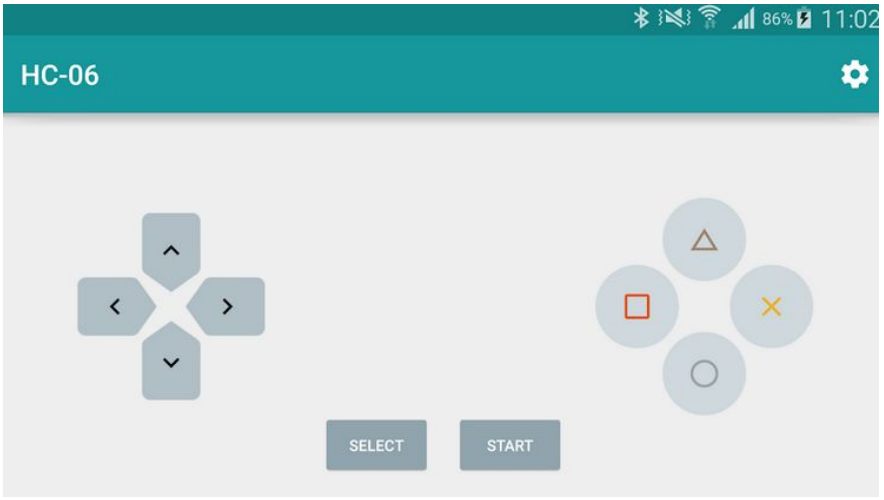


Capture d'écran du terminal du smartphone - (RP)

Capture d'écran du terminal du smartphone - (RP)

Le moniteur série affiche bien les caractères envoyés depuis mon smartphone, nous pouvons donc en conclure que le module bluetooth fonctionne correctement.

Le problème que nous avons rencontré plus tard est le fait que le bluetooth ne fonctionne qu'avec l'application terminal mais pas avec d'autre applications telles que des télécommandes virtuelles, comme celle-ci :



Capture d'écran de télécommande bluetooth virtuelle - (Source)

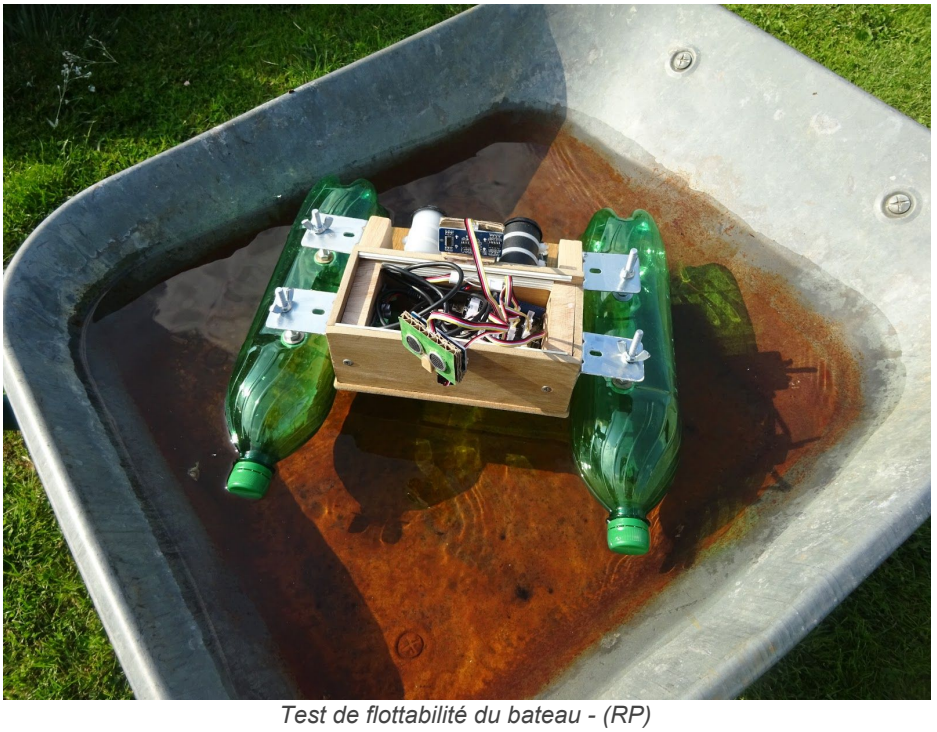
J'ai testé 6 applications différentes mais aucune n'arrive à se connecter à l'arduino. J'ai fais des recherches sur internet mais je n'ai pas trouvé la cause du problème.

V. Intégration et validation

Au cours de ce projet, toute l'équipe s'est organisée de manière coordonnée de façon à ce que chacun puisse apporter son aide au groupe. Ayant un peu plus d'expérience en programmation, j'ai pu aider mes camarades dans la réalisation du programme et au final chacun a su réaliser les tâches qui lui avaient été confiées. Je me suis donc occupé, comme prévu lors de la répartition des tâches, des capteurs à ultrasons et du module bluetooth.

La partie finale de notre projet a donc consisté à l'intégration au code des fonctions développées par chacun d'entre nous. Cette étape s'est passée sans difficulté majeure, car nous avions préalablement défini un certain nombre de consignes partagées. Ainsi nous avions, par exemple, défini les variables avec les mêmes noms ce qui a fortement facilité l'intégration finale.

Nous avions défini un mode de validation pour chacune des étapes de développement. Un test était réalisé de manière à s'assurer que chacune des fonctions était opérationnelle avant de passer à l'étape suivante (par exemple, le test du module bluetooth explicité auparavant). Cette méthode nous a permis d'éliminer petit à petit les sources d'erreur. Une fois le programme final assemblé, nous avons pu tester notre bateau afin de vérifier que toutes les fonctions mises en commun fonctionnaient correctement.



Test de flottabilité du bateau - (RP)

VI. Bilan et perspectives

Bilan et perspectives du projet

Au terme du développement de notre programme, le bateau est capable d'éviter les obstacles de manière autonome et d'être téléguidé depuis un smartphone. Même si aujourd'hui, le bateau est fonctionnel, ce n'est que la première version de celui-ci. En effet, afin de tenir le délai imposé, nous avons dû faire des concessions. Cependant voici une liste d'améliorations possibles pour les futures versions :

- Installer un module GPS afin de pouvoir suivre un itinéraire ou sauvegarder son itinéraire.
- Fabriquer une télécommande de manière à contrôler le bateau grâce à des joysticks.
- Développer une application android et ios afin de pouvoir téléguider le bateau depuis son smartphone.
- Améliorer l'aspect mécanique du bateau :
    - En améliorant la vitesse grâce à de meilleurs moteurs ou en utilisant une batterie 12V.
    - En utilisant des courroies à la place d'élastiques afin d'améliorer le rendement.

Bilan personnel

L'arduino permet à chacun de rendre possible la réalisation de nombreux projets qui semblent à priori complexes, ainsi j'ai pu expérimenter, avec mon équipe, la conception d'un bateau téléguidé. Au cours de la réalisation du projet, j'ai découvert la programmation de microcontrôleur tel l'arduino basé sur le langage C / C++, ainsi que des interfaces numériques simples mais néanmoins très répandues : UART, PWM, I2C.

Ce projet m'a permis d'avoir une première expérience de travail en groupe sur un gros projet. J'ai appris à travailler en même temps que mes camarades sur un même programme, en s'adaptant au rythme de chacun, dans un dialogue permanent : en travaillant sur une partie puis sur une autre, revenir sur la première, discuter pour résoudre un problème rencontré dans celle-ci puis reprendre la deuxième. Cela m'a donné un aperçu de ce que peut être le travail collaboratif en entreprise et m'encourage à trouver une formation afin de m'y préparer.

Annexes

Sources

- Référence du langage
- *Livre de projets arduino* tiré du “starter kit arduino”
- Fiche technique du capteur à ultrasons
- Fiche technique du module bluetooth
- Vidéo explicative sur le PWM
- Vidéo explicative UART

NB : La mention “RP” signifie *réalisation personnelle*

Code

Programme du bateau

```
//Bibliothèques:

#include <Wire.h> //I2C
#include <SoftwareSerial.h> //BLE
#include <Servo.h> //ServoMoteur


//Constantes:

#define US_FRONT 7
#define US_REAR 6

#define MOTOR_LEFT  0
#define MOTOR_RIGHT 1

#define SERVO_FRONT 5

#define BLE_RX 2
#define BLE_TX 3

#define I2C_MOTORDRIVER_ADDR          0x14
#define I2C_MOTORDRIVER_CMD_STOP      0x01
#define I2C_MOTORDRIVER_CMD_FORWARD   0x02
#define I2C_MOTORDRIVER_CMD_REVERSE   0x03

#define SOUND_SPEED 0.034029
```

```
#define DISTANCE_AVOID 50


//Objets:

class Ultrasonic //Capteur Ultrason
{
  private:
     //Varibales :
     int pin;
  public:
     //Fonctions :
     Ultrasonic(int select_pin);
     int Measure();
};

Ultrasonic::Ultrasonic(int select_pin) //Constructeur
{
  pin = select_pin;
}

int Ultrasonic::Measure() // Mesure de la distance
{
  //Trigger
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
  delayMicroseconds(2);
  digitalWrite(pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pin,LOW);

  //Mesure
  pinMode(pin,INPUT);
  return ((pulseIn(pin,HIGH)*SOUND_SPEED)/2);
}


class Motor //Moteur
{
  private:
     //Varibales
     int channel;
  public:
     //Fonctions
     Motor(int set_channel);
```

```
      stop();
      run(int speed);
};

Motor::Motor(int set_channel) //Constructeur
{
  channel = set_channel;
}

Motor::stop() //Arrêt
{
  Wire.beginTransmission(I2C_MOTORDRIVER_ADDR);
  Wire.write(byte(I2C_MOTORDRIVER_CMD_STOP));
  Wire.write(byte(channel));
  Wire.endTransmission();
  delay(1);
}


Motor::run(int speed) //Activation
{
  if (speed > 255) speed = 255;
  else if (speed < -255) speed = -255;

  Wire.beginTransmission(I2C_MOTORDRIVER_ADDR);
  if (speed >=0)
      {
      Wire.write(byte(I2C_MOTORDRIVER_CMD_FORWARD));
      Wire.write(byte(channel));
      Wire.write(byte(speed));
      }
      else
      {
      Wire.write(byte(I2C_MOTORDRIVER_CMD_REVERSE));
      Wire.write(byte(channel));
      Wire.write(byte(-speed));
      }

  Wire.endTransmission();
  delay(1);
}


//Création des objets :
```

```
Ultrasonic ultrasonic_front(US_FRONT);
Ultrasonic ultrasonic_rear(US_REAR);
Motor motor_left(MOTOR_LEFT);
Motor motor_right(MOTOR_RIGHT);
Servo servo_front;
SoftwareSerial BLE(BLE_RX,BLE_TX);

//Variables:

int range_front_0;
int range_front_45;
int range_front_90;
int range_front_135;
int range_front_180;
int range_rear;
char recvChar;
char last_recvChar;


//Fonctions (Anti-collision):

void avoid_forward_0()
{
  delay(200);
  motor_left.run(-255);
  motor_right.run(-255);
  delay(2000);
  motor_left.stop();
  motor_right.stop();
  delay(100);
  motor_left.run(255);
  motor_right.run(-255);
  delay(1500);
  motor_right.stop();
}

void avoid_forward_45()
{
  delay(200);
  motor_left.run(-255);
  motor_right.run(-255);
  delay(5000);
  motor_left.stop();
  motor_right.stop();
  delay(250);
```

```
  motor_left.run(255);
  motor_right.run(-255);
  delay(500);
  motor_left.stop();
}

void avoid_forward_90()
{
  delay(200);
  motor_left.run(-255);
  motor_right.run(-255);
  delay(2000);
  motor_left.stop();
  motor_right.stop();
}

void avoid_forward_135()
{
  delay(200);
  motor_left.run(-255);
  motor_right.run(-255);
  delay(2000);
  motor_left.stop();
  motor_right.stop();
  delay(100);
  motor_right.run(255);
  motor_left.run(-255);
  delay(1500);
  motor_right.stop();
}

void avoid_forward_180()
{
  delay(200);
  motor_left.run(-255);
  motor_right.run(-255);
  delay(2000);
  motor_left.stop();
  motor_right.stop();
  delay(100);
  motor_right.run(255);
  motor_left.run(-255);
  delay(500);
  motor_right.stop();
}
```

```
void avoid_backward()
{
  delay(200);
  motor_left.run(-255);
  motor_right.run(-255);
  delay(2000);
  motor_left.stop();
  motor_right.stop();
}


//Fonctions Contrôle:

void go_forward()
{
  motor_left.run(255);
  motor_right.run(255);
}

void idle()
{
  motor_left.stop();
  motor_right.stop();
}

void go_backward()
{
  motor_left.run(-255);
  motor_right.run(-255);
}

void go_left()
{
  motor_left.run(-255);
  motor_right.run(255);
}

void go_right()
{
  motor_left.run(255);
  motor_right.run(-255);
}
```

```
//Fonction Bluetooth:

void setupBleConnection()
{
  BLE.begin(9600);
  delay(100);
  BLE.print("AT+CLEAR");
  delay(100);
  BLE.print("AT+ROLE0");
  delay(100);
  BLE.print("AT+SAVE0");
  delay(100);
  BLE.print("AT+NAMEBATEAU");
}


//Tâches :

void task_SM0()
{
  range_front_0 = ultrasonic_front.Measure();
  delay(25);
  if (range_front_0 < DISTANCE_AVOID){
      avoid_forward_0();
  }
  servo_front.write(45);
  delay(200);
}

void task_SM45()
{
  range_front_45 = ultrasonic_front.Measure();
  delay(25);
  if (range_front_45 < DISTANCE_AVOID){
      avoid_forward_45();
  }
  servo_front.write(90);
  delay(200);
}

void task_SM90()
{
  range_front_90 = ultrasonic_front.Measure();
  delay(25);
  if (range_front_90 < 50){
```

```
      avoid_forward_90();
  }
  servo_front.write(135);
  delay(200);
}

void task_SM135()
{
  range_front_135 = ultrasonic_front.Measure();
  delay(25);
  if (range_front_135 < DISTANCE_AVOID){
      avoid_forward_135();
  }
  servo_front.write(180);
  delay(250);
}

void task_SM180()
{
  range_front_180 = ultrasonic_front.Measure();
  delay(25);
  if (range_front_180 < DISTANCE_AVOID){
      avoid_forward_180();
  }
  servo_front.write(0);
  delay(500);
}

void task_SMrear()
{
  range_rear = ultrasonic_rear.Measure();
  delay(25);
   if (range_rear < DISTANCE_AVOID){
      avoid_backward();
  }
 }

void task_BLE()
{
  if(BLE.available()){

      recvChar = BLE.read();
      if (recvChar != last_recvChar){

      if(recvChar == 'z'){
```

```
      last_recvChar = 'z';
      go_forward();
      }

      if(recvChar == 's'){
      last_recvChar = 's';
      idle();
      }

      if(recvChar == 'w'){
      last_recvChar = 'w';
      go_backward();
      }

      if(recvChar == 'q'){
      last_recvChar = 'q';
      go_left();
      }

      if(recvChar == 'd'){
      last_recvChar = 'd';
      go_right();
      }
      }
  }
}


void setup()
{
  //Moniteur Série
  Serial.begin(9600);

  //I2C
  Wire.begin();

  //BLE
  pinMode(BLE_RX, INPUT);
  pinMode(BLE_TX, OUTPUT);
  setupBleConnection();

  //Servo
  servo_front.attach(SERVO_FRONT);
  servo_front.write(0);
```

```
  delay(5000);
}

void loop()
{
  task_SM0();
  task_BLE();
  task_SM45();
  task_BLE();
  task_SM90();
  task_BLE();
  task_SM135();
  task_BLE();
  task_SM180();
  task_BLE();
  task_SMrear();
  task_BLE();
}
```

Programme de test du capteur à ultrasons

Retour à la lecture du dossier

```
//Constantes
#define PIN_US 5
#define SOUND_SPEED 0.034029

class Ultrasonic //Capteur Ultrason
{
  private:
      //Varibales :
      int pin;
  public:
      //Fonctions :
      Ultrasonic(int select_pin);
      int Measure();
};

Ultrasonic::Ultrasonic(int select_pin) //Constructeur
{
  pin = select_pin;
}

int Ultrasonic::Measure() // Mesure de la distance
{
  //Trigger
```

```
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
  delayMicroseconds(2);
  digitalWrite(pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pin,LOW);

  //Measure
  pinMode(pin,INPUT);
  return ((pulseIn(pin,HIGH)*SOUND_SPEED)/2);
}

//Création objet
Ultrasonic ultrasonic(PIN_US);

void setup()
{
      Serial.begin(9600);
}
void loop()
{
      int RangeInCentimeters;

      RangeInCentimeters = ultrasonic.Measure();
      Serial.print(RangeInCentimeters);
      Serial.println(" cm");
      delay(500);
}
```

Programme de test du bluetooth

Retour à la lecture du dossier

```
#include <SoftwareSerial.h> //BLE

#define BLE_RX 2
#define BLE_TX 3

char recvChar;
char last_recvChar;

SoftwareSerial BLE(BLE_RX,BLE_TX);

void setupBleConnection()
```

```
{
  BLE.begin(9600);
  delay(100);
  BLE.print("AT+CLEAR");
  delay(100);
  BLE.print("AT+ROLE0");
  delay(100);
  BLE.print("AT+SAVE0");
  delay(100);
  BLE.print("AT+NAMEBATEAU");
}

void task_BLE()
{
if(BLE.available()){

  recvChar = BLE.read();

      if(recvChar == 'z'){
      last_recvChar = 'z';
      Serial.print("reçu : ");
      Serial.println(recvChar);
      }
  }
}

void setup()
{
  Serial.begin(9600);
  pinMode(BLE_RX, INPUT);
  pinMode(BLE_TX, OUTPUT);
  setupBleConnection();
}

void loop()
{
task_BLE();
}
```