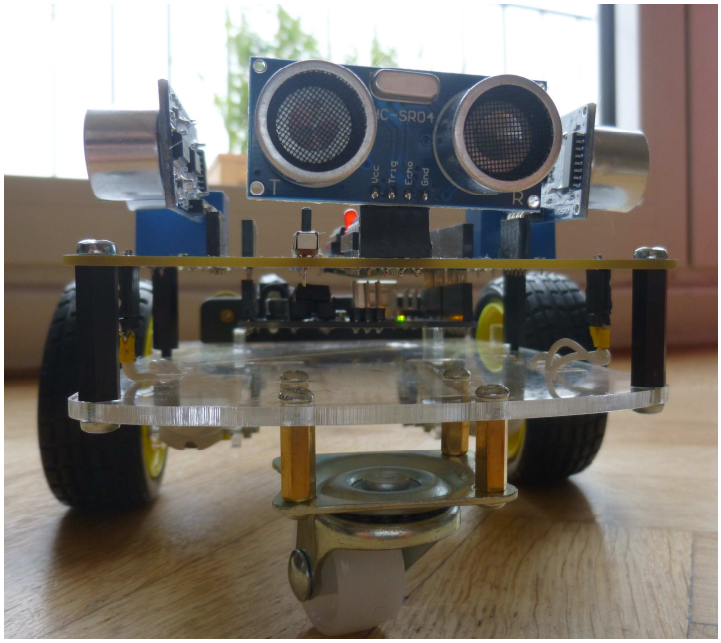
RELUS3



*Le robot est construit sur base d'une carte **ARDUINO UNO**, on l'appelle RELUS3 parce qu'il contient des relais (REL) et trois capteurs à ultrasons (US3).*

Qu'est-ce que RELUS3 peut faire ?

Tout dépendra de la manière dont on va le programmer...
Comme les capteurs à ultrasons sont dirigés suivant trois directions différentes, on doit pouvoir le faire longer un mur.
Les clignotants arrières font du bruit grâce à l'utilisation des relais.
Les deux boutons-poussoirs et les trois LEDS permettent de prévoir plusieurs modes de fonctionnement.

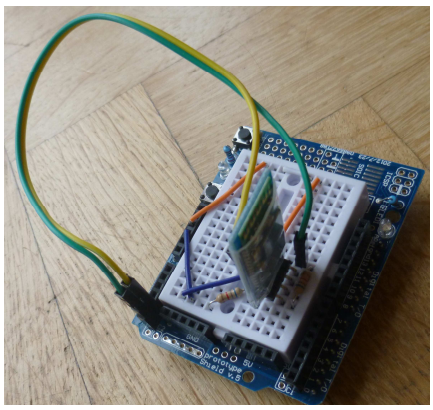
Est-ce que RELUS3 peut faire d'autres choses?

En ajoutant une petite "breadboard" on peut ajouter des fonctionnalités supplémentaires. On peut par exemple ajouter un récepteur bluetooth pour pouvoir télécommander le robot avec un téléphone.

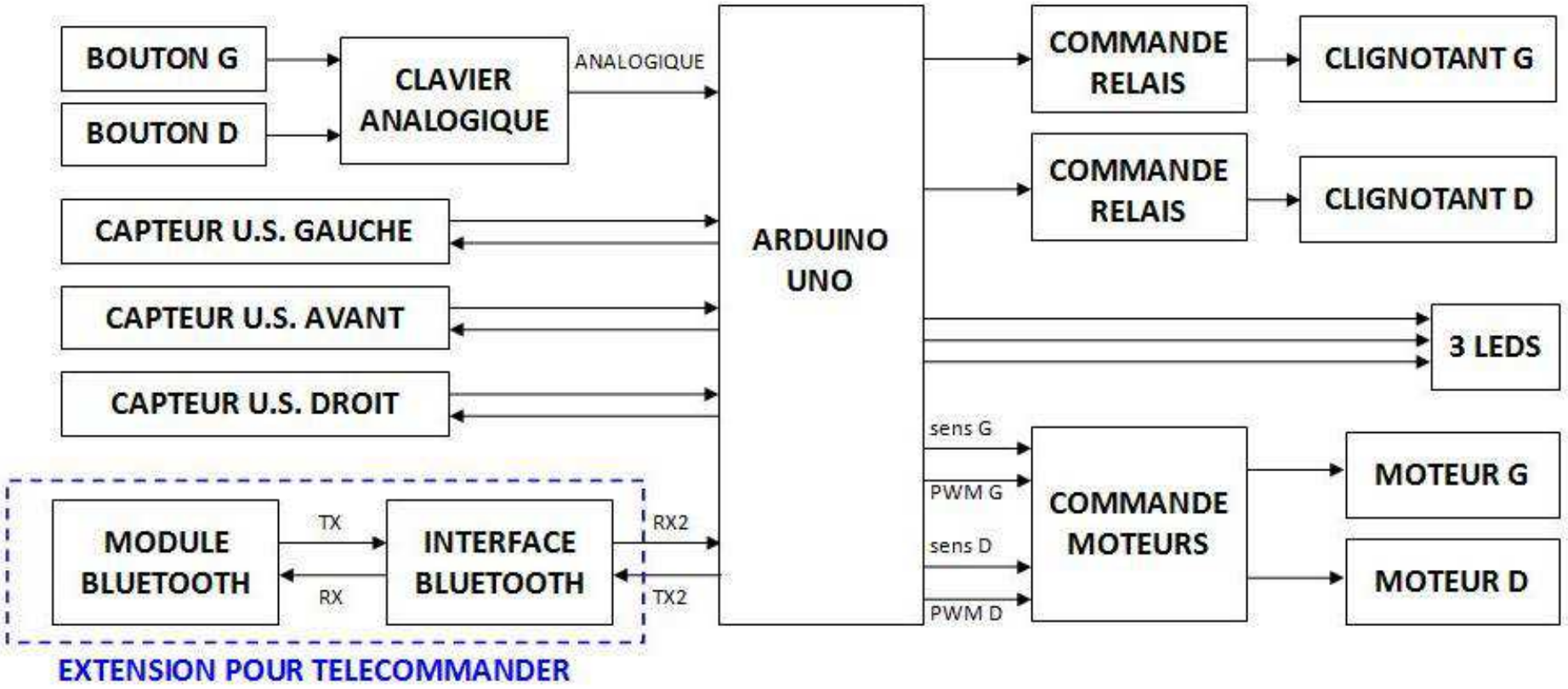
Exemple d'extension :

Le module bluetooth se met dans la "breadboard" puis toute la carte se connecte au robot.

On peut alors télécommander notre RELUS3.



SCHEMA BLOC DE "RELUS3"

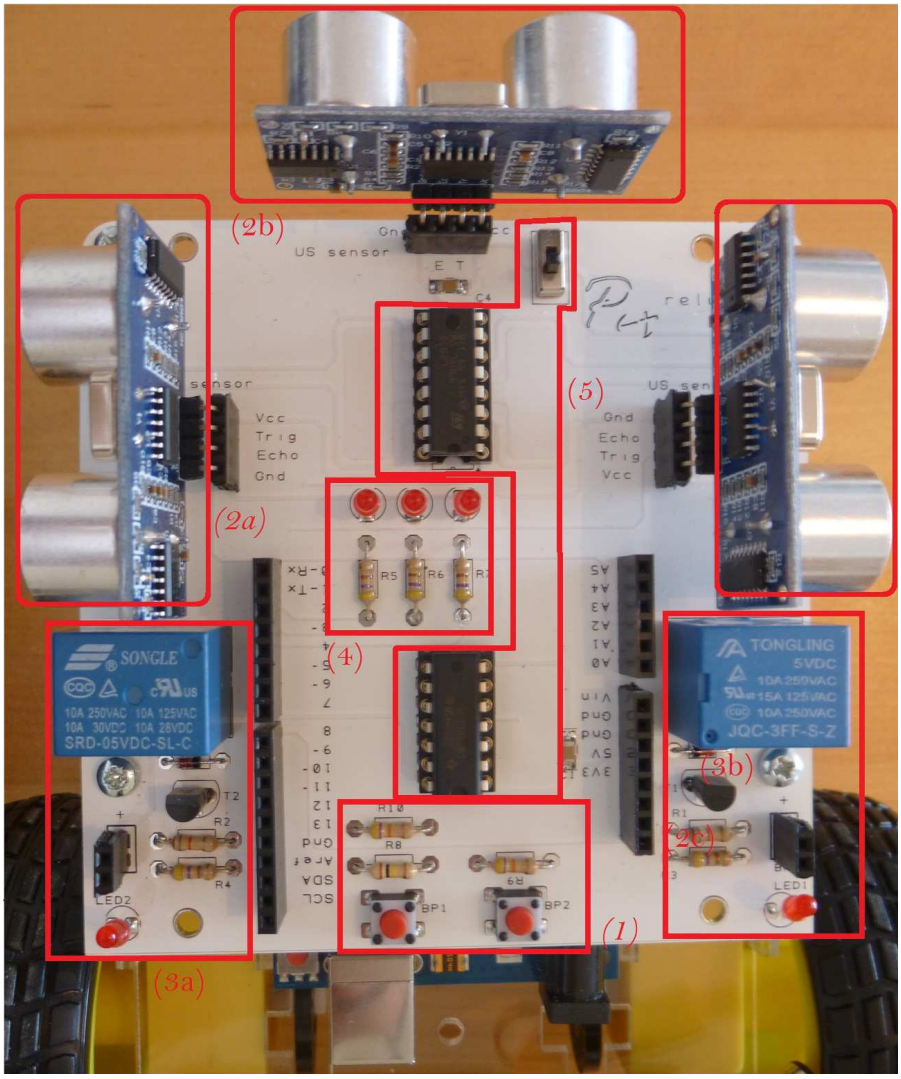


EMPLACEMENT DES BLOCS SUR LE ROBOT

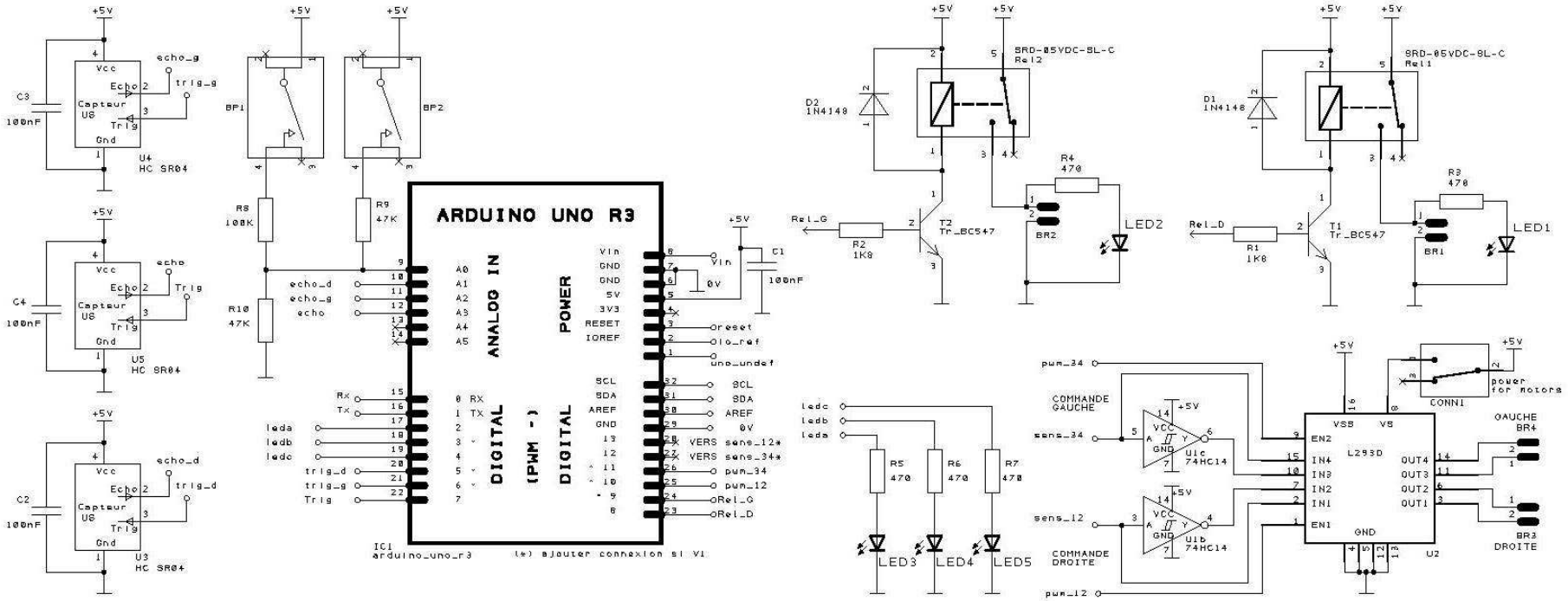
Le schéma bloc montre les différents blocs "fonctionnels" du robot. Les entrées à gauche montrent les éléments qui apportent des informations alors que les sorties à droite reçoivent des commandes de la carte Arduino Uno.

On peut identifier ci-dessous, sur la photo du circuit soudé, les différents blocs:

(1)	Les deux boutons-poussoirs et les résistances qui forment le clavier analogique.
(2a) (2b) (2c)	Les capteurs ultrasons gauche, avant et droit.
(3a) (3b)	Les clignotants gauches et droits commandés par des relais.
(4)	Les trois LEDS accompagnées de ses résistances de protection
(5)	La commande des moteurs DC



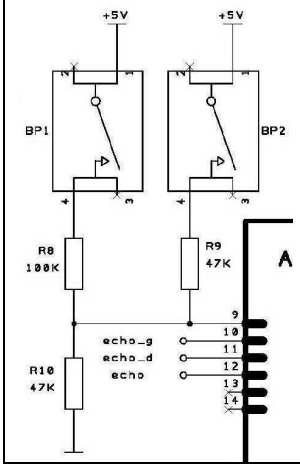
SCHEMA DE PRINCIPE DE "RELUS3"



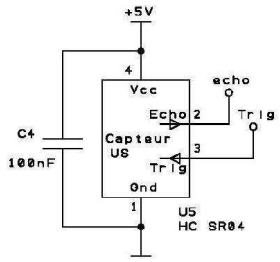
Principe de fonctionnement

Nous expliquerons brièvement le fonctionnement des différents éléments en nous basant sur le schéma bloc et en montrant la partie du schéma de principe qui correspond.

Les boutons gauche et droit

Schéma	Explications
	La patte 9 de l'Arduino Uno va recevoir une tension analogique qui dépendra de la combinaison des boutons-poussoirs. L'Arduino va alors utiliser son convertisseur analogique/numérique interne pour obtenir un nombre. Le programme devra alors analyser ce nombre pour en déduire la combinaison des boutons-poussoirs. La tension analogique qui se trouve sur la patte 9 peut se calculer grâce au principe du diviseur de tension.

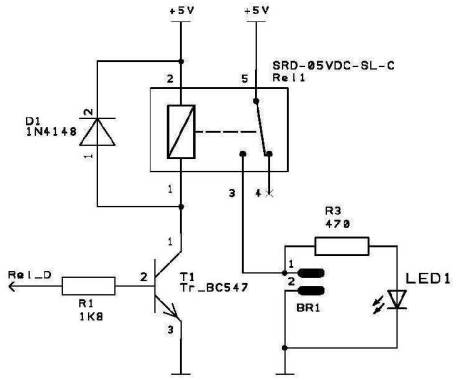
Les capteurs à ultrasons

Schéma	Explications
	Chaque capteur US est connecté à la carte Arduino Uno grâce aux signaux "echo" et "trig". Le programme envoie un signal via le "trig" du capteur qui émet un ultrason; l'ultrason va rencontrer un obstacle, revenir et informer l'Arduino via l' "echo" que l'ultrason est revenu. La durée permet d'estimer la distance.

Les clignotants commandés par des relais

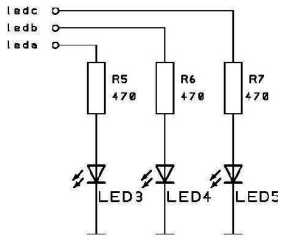
Un relais permet d'actionner un interrupteur en actionnant un électroaimant. Lorsqu'on fait passer un courant électrique dans le bobinage du relais, on crée un aimant qui attire l'interrupteur et on entend un "clic"; de même, lorsqu'on arrête de faire passer du courant par le bobinage, l'aimant s'arrête et l'interrupteur se remet en position de repos, on entend alors un "clac".
L'idée est alors de pouvoir envoyer une commande à partir de l'Arduino qui provoquera un courant ou non au sein du bobinage du relais (et commander le "clic / clac").

Comme l'Arduino ne peut pas fournir directement un courant suffisant au sein du bobinage du relais, on utilise un circuit avec un transistor en commutation entre l'Arduino et le relais.

Schéma	Explications
	L'Arduino va donner un signal numérique (0-5V) sur "Rel_D". Si le signal est à 0V, le transistor T1 ne va pas laisser passer le courant dans le bobinage du relais, on dira que le relais est en position "clac". La LED est alors éteinte. Si le signal est à 5V, le transistor va laisser passer le courant dans le relais qui va générer un aimant et attirer le contact, on dira que le relais passe en position "clic", la LED s'allume.

On voit que le schéma contient aussi une diode (D1) en parallèle sur la bobine du relais. Cette diode sert à protéger le transistor lors de commutations. En effet, les bobines "n'aiment pas" les variations trop brusques de courant, on place donc une diode de protection qu'on appelle diode de roue libre pour éviter ces variations brusques de courant.

Les LEDS

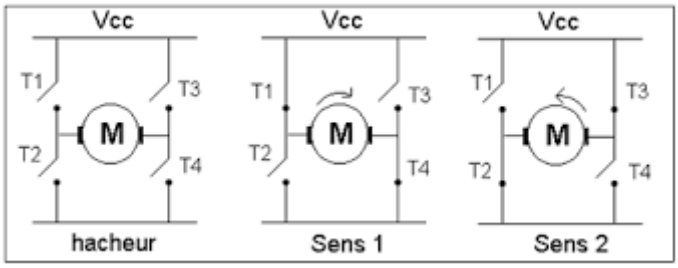
Schéma	Explications
	Chaque signal destiné à une LED est commandé en 0 - 5V. Chaque LED est alors accompagnée d'une résistance série qui va limiter le courant dans celle-ci. Pour par exemple allumer la LED5, on met 5V au signal "ledc", ce qui va allumer la LED avec une tension d'environ 2V et laisser environ 3V aux bornes de R7 (donc, loi d'Ohm, $I \approx 6mA$).

Les moteurs

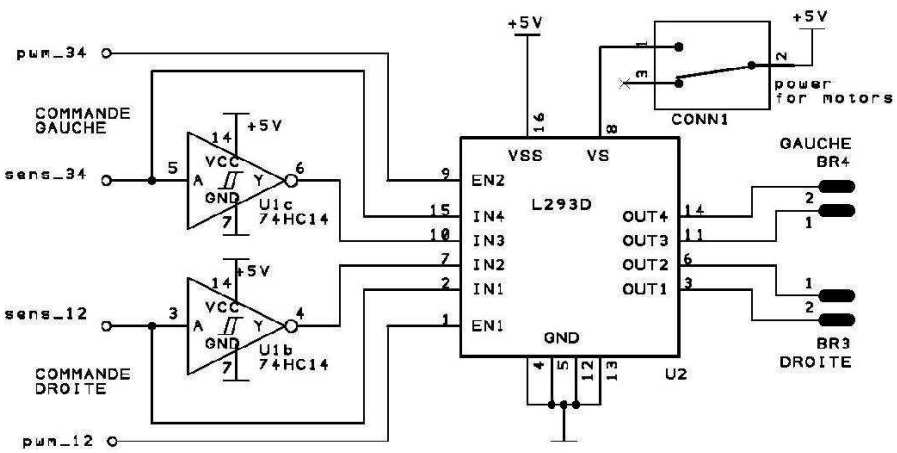
Pour commander un moteur DC, on peut lui appliquer une tension électrique qui va le faire tourner. Le sens de rotation du moteur dépendra de la polarité de la tension électrique et la vitesse du moteur dépendra de la valeur de la tension électrique.
Pour influencer la polarité de la tension électrique aux bornes du moteur, on utilise un circuit qu'on appelle "pont en H"; ce circuit permet également de fournir la puissance nécessaire aux moteurs.

Pour influencer la tension aux bornes du moteur, on utilise une astuce qui va modifier la tension moyenne en utilisant un signal numérique.
On va par exemple fournir un signal de type carré dont les niveaux sont 0 - 5V et faire varier le rapport cyclique (fraction de la période dont le signal est au niveau haut).
A titre d'exemple, si on imagine un rapport cyclique de 50%, c'est comme si on demandait au moteur de tourner vite (tension de 5V) pendant 50% du temps et d'être à l'arrêt (tension de 0V) pendant 50% du temps.
Si le commandes AVANCER - STOP - AVANCER - STOP - ... sont assez rapides, le moteur se comportera comme si on lui appliquait la tension moyenne de 2,5V à ses bornes.

Le principe du pont en H schématisé ci-dessous (www.technologuepro.com) montre que le sens de rotation du moteur (M) dépend de l'état des interrupteurs T1 à T4; ces interrupteurs sont en réalité des transistors commandés par des signaux de commande.
On voit qu'on fermant T1 et T4, le moteur tourne dans un sens, en fermant T2 et T3, le moteur tourne dans l'autre sens.



Dans le cadre de notre robot, les ponts en H nécessaires à commander nos deux moteurs DC sont intégrés au sein de notre circuit L293D. Le circuit inverseur 74HC14 permet de prendre en charge la gestion des inversions de polarités de moteurs.

Schéma	Explications
	Les signaux PWM commandent la vitesse. Les signaux SENS commandent le sens des moteurs. BR3 et BR4 sont connectés aux moteurs.

Module d'extension Bluetooth

Pour ajouter des extensions à notre robot, il est pratique d'ajouter une carte dédiée à l'Arduino Uno qui contient une "breadboard" (plaquette d'essais).

Afin de pouvoir commander notre robot à l'aide d'une connexion Bluetooth, on peut ajouter un module Bluetooth sur notre "breadboard" et ajouter le câblage nécessaire comme montré ci-dessous :

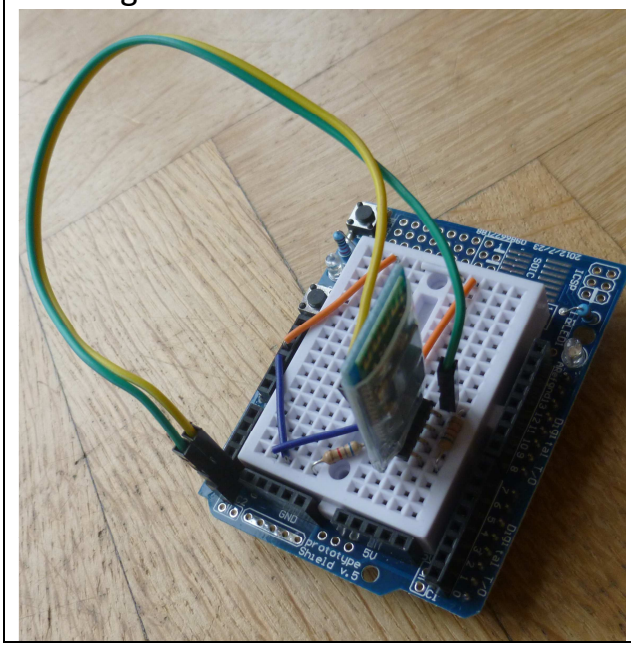
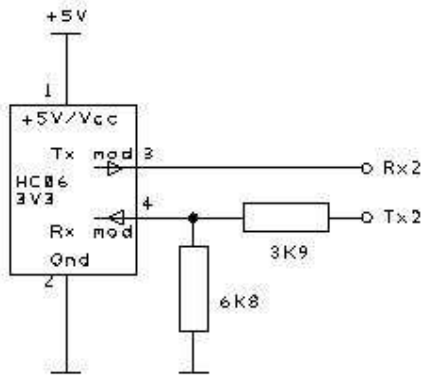
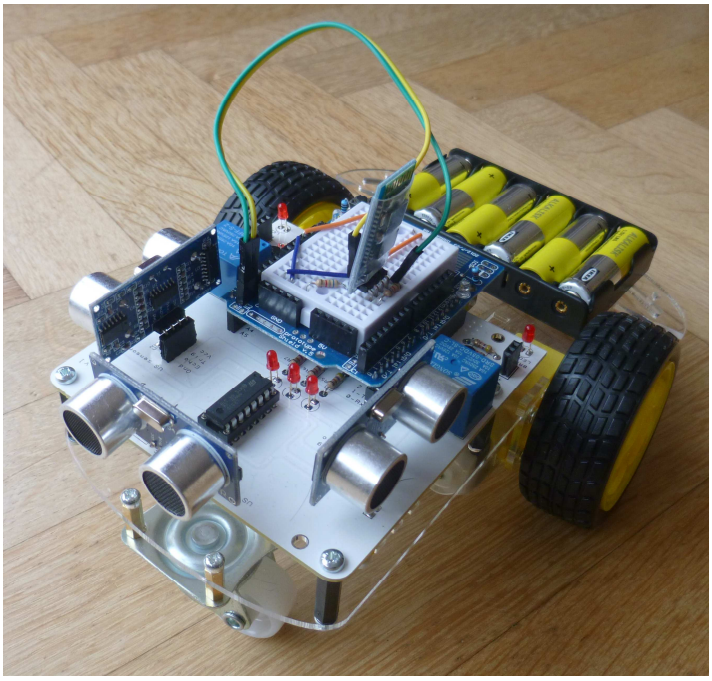
Montage	Descriptif
	Cette plaquette d'essai vient se connecter directement sur l'Arduino Uno. On peut alors connecter les signaux souhaités à la "breadboard". On utilise le module Bluetooth HC-06 qui est un module esclave composé de 4 pattes (pins) d'interconnexion : VCC et GND pour l'alimentation Tx et Rx pour la connexion série 

Schéma du montage	Descriptif
	Les pattes 1 et 2 (VCC et GND) du module Bluetooth sont respectivement connectés au +5V et Gnd de la carte Arduino Uno. Le module fonctionne en 0 - 3,3V pour Rx et Tx : La patte de sortie 3 (Tx) du module peut être connectée telle quelle à une entrée de l'Arduino. La patte d'entrée 4 (Rx) du module recevra une fraction du signal de commande de l'Arduino.

On doit garder à l'esprit que le module Bluetooth fonctionne en 3,3V alors que la carte Arduino fonctionne 5V. Comme les niveaux de tension sont différents (avec toutefois une masse commune), on utilise un diviseur résistif pour éviter d'arriver avec 5V sur l'entrée du module.
Les signaux appelés ici Rx2 et Tx2 forment un UART (port série) virtuel.

On peut choisir arbitrairement les pattes de l'Arduino qui serviront de Rx2 et Tx2. Dans le montage présenté, Rx2 correspond à A4 et Tx2 à A5.

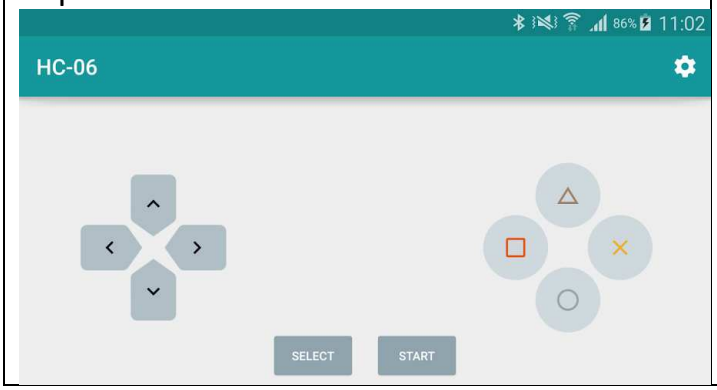
Une fois la carte d'extension câblée correctement, on peut la connecter au dessus de la carte Arduino Uno. On obtient un robot pilotable par Bluetooth.



Comment piloter RELUS3 par Bluetooth ?

On peut utiliser un dispositif doté du Bluetooth pour envoyer des signaux au module HC-06. Une solution simple consiste à installer une application sur "smartphone" qui permettra de piloter le robot.

A titre d'exemple, l'application "Android" nommée "Arduino bluetooth controller" permet d'utiliser l'interface ci-dessous :

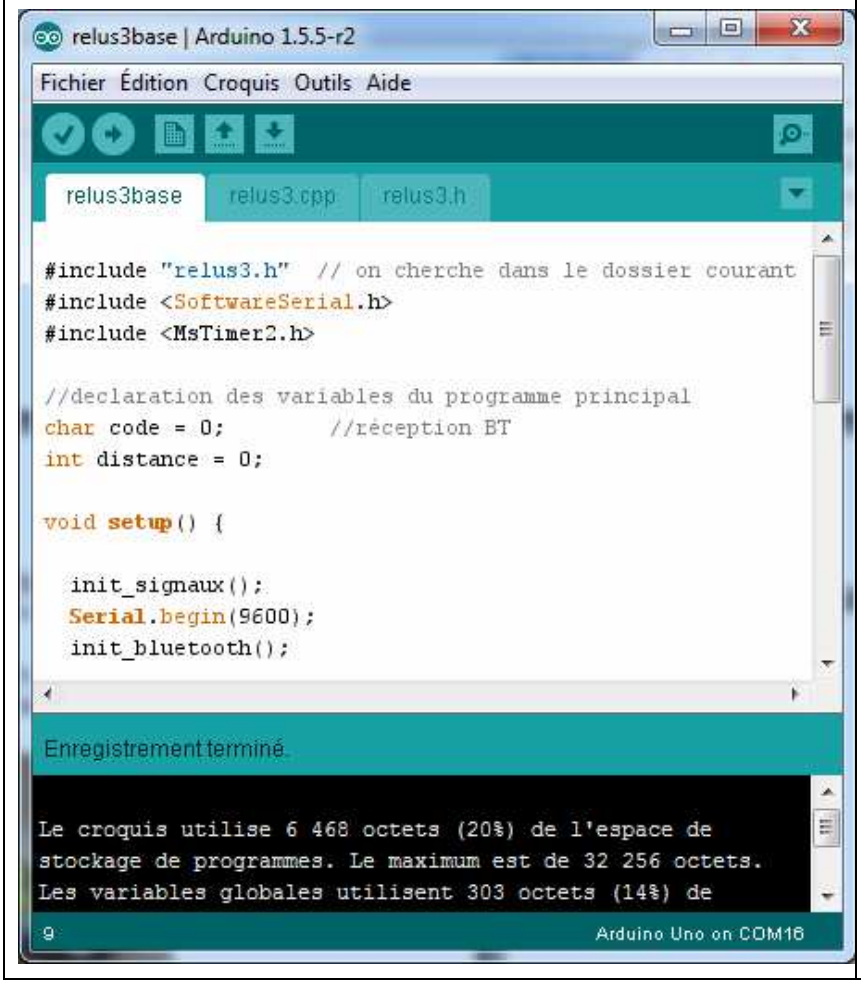
Représentation de l'interface	Paramétrage choisi pour RELUS3 Chaque touche est liée à un caractère.
	 'L' 'U' 'R' 'D'  's'  '1'  'c' 't' 'x' 'o'

Bien entendu les actions à réaliser devront être programmées.

Programmation de RELUS3

La programmation se fait au travers de l'environnement de programmation Arduino. Une librairie dédiée à "RELUS3" a été écrite et est disponible au sein du projet "relus3base". Celle-ci est composée des fichiers "relus3.h" et "relus3.cpp" et peut-être utilisée grâce à la directive #include "relus3.h"

Fenêtre de l'environnement Arduino

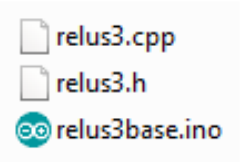


Arborescence

En Arduino,il faut réserver un dossier par projet.

Le nom du programme a une extension .ino et doit se trouver dans un dossier du même nom.

On trouve dans un dossier relus3base Les trois fichiers :



Les fichiers relus3.h et relus3.cpp sont dans le même dossier pour un accès direct via :

#include "relus3.h"

Que contient le fichier relus3.h ?

Ce fichier contient le paramétrage des signaux, les conventions de dialogue par Bluetooth et les dénominations des états de déplacement du robot.
De plus, le fichier contient les déclarations des fonctions de la librairie "relus3".

Pour paramétrer les signaux, on trouve des lignes comme :

```
#define LED_C     4
```

Ceci nous permet de commander la LED_C en l'appelant par son nom.

Dans le même esprit, les autres signaux sont associés aux pattes de l'Arduino auxquelles ils sont connectés. On trouve alors les noms de signaux suivants :

LED_A, LED_B, LED_C, REL_G, REL_D, RX2, TX2, PWM_D, PWM_G, SENS_G, SENS_D, BP, TRIG, _G, TRIG_D, ECHO, ECHO_G, ECHO_D

Pour paramétrer les conventions de dialogue par Bluetooth, on utilise également des directives *#define*, on trouve alors des lignes telles que :

```
#define UP       'U'
```

On augmente la lisibilité du programme en associant à chaque caractère de commande bluetooth un nom évocateur. On trouve les appellations suivantes :

UP, DOWN, LEFT, RIGHT, SELECT, START, CARRE, TRIANGLE, CROIX, ROND, NO_BT

Pour gérer le déplacement du robot, la librairie utilise une variable d'état qui mémorise l'état actuel de déplacement du robot. C'est comme si on prenait note du déplacement actuel du robot pour mieux modifier son déplacement lorsqu'une nouvelle commande de pilotage bluetooth survient.

Il a été décidé que les états de déplacements possibles sont :

arrêt, vers l'avant, vers l'arrière, gauche, droite, avant gauche, avant droite, arrière gauche et arrière droite

Comme sur base d'une boussole, les dénominations respectives sont :

STOP, N, S, O, E, NO, NE, SO, SE

Enfin, le fichier *relus3.h* contient toutes les déclarations des fonctions de la librairie, à savoir :

Déclarations des fonctions	Petit descriptif
`char bp1(void);` `char bp2(void);`	Lecture des boutons-poussoirs BP1 et BP2

`int dist(int , int);`
`int dist_US(void);`
`int dist_US_G(void);`
`int dist_US_D(void);`
	Lecture des distances des trois capteurs ultrasons

`void init_signaux(void);`
`void init_bluetooth(void);`
`char lire_bluetooth(void);`
	Paramétrage des entrées / sorties
	Gestion du Bluetooth

Déclarations des fonctions	Petit descriptif

`void arret(void);`
`void avance(void);`
`void recule(void);`
`void gauche(void);`
`void droite(void);`
`void avance_gauche(void);`
`void avance_droite(void);`
`void recule_gauche(void);`
`void recule_droite(void);`
	.
	.
	Gestion du déplacement du robot en agissant sur les signaux PWM_D, PWM_G, SENS_G, SENS_D
	.
	.

`void action_carre(void);`
`void action_triangle(void);`
`void action_croix(void);`
`void action_rond(void);`
`void action_select(void);`
`void action_start(void);`
`void action_manette_directionnelle(char);`
`void action_bluetooth(char);`
	.
	Actions liées à une commande bluetooth d'un "bouton"
	.
	Action liée à la "manette bluetooth"
	Action liée à une réception bluetooth

Que contient le fichier relus3.cpp ?

Ce fichier contient l'implémentation des fonctions de la librairie, autrement dit il précise ce qu'il faut faire lorsqu'on veut exécuter une fonction.

Certaines fonctions devront être modifiées/complétées (voire écrites), notamment celles liées au déplacement du robot. En effet, si on souhaite par exemple faire rouler le robot tout droit, il faut donner des valeurs aux signaux PWM_D et PWM_G qui doivent être adaptées à chaque robot.

Il n'est pas possible de donner des valeurs aux signaux PWM_D et PWM_G qui sont valables pour tous les robots car chaque moteur est différent, les frottements des roues sont différents,...
En réalité, il n'est même pas possible de faire en sorte que le robot aille tout à fait droit sans utiliser un système en boucle fermée[*].

Nous allons nous contenter de décrire le rôle des fonctions de la librairie sans entrer dans les détails de l'implémentation.

[*] Un système en boucle fermée mesure en permanence ce qu'on essaie de réguler pour influencer la grandeur de réglage de manière à atteindre la consigne.

• *Les fonctions bp1() et bp2()*

Ces fonctions vont lire l'entrée analogique au travers de laquelle les boutons sont interconnectés et déterminer si le bouton précisé est enfoncé ou non.
Si par exemple le bouton-poussoir 1 est enfoncé, la fonction bp1() renverra la valeur 1, sinon elle renverra la valeur 0.

On écrira par exemple :

```
if( bp1() )
  {
   //...
  }
```

```
while (! bp2() )
 {
   //...
  }
```

• *La fonction dist()*

La fonction dist() sert d'utilitaire aux fonctions dist_US(), dist_US_G() et dist_US_D(). On ne devra pas l'utiliser directement.
Elle sert à renvoyer la distance en [cm] d'un capteur US dont les signaux sont donnés en paramètre.

• *Les fonctions dist_US(), dist_US_G() et dist_US_D()*

Ces fonctions vont légèrement intégrer la mesure du capteur ultrasons précisé et renvoyer un entier (normalement compris entre 5 et 100) pour indiquer la distance de l'obstacle dans son " axe de vision".

On écrira par exemple :

```
//...
d = dist_US_G();
if( d >=5 && d < 20 )
  {
   //...
  }
```

```
//...
d = dist_US();
while( d >30 )
  {
   //...
  d = dist_US();
  }
```

• *La fonction init_signaux()*

Cette fonction ne sera appelée qu'une fois au début du programme pour préciser l'assignation "entrée" ou "sortie" des différents signaux.

• *La fonction init_bluetooth()*

Cette fonction ne sera appelée qu'une fois au début du programme pour initialiser les paramètres de la connexion bluetooth.

• *La fonction lire_bluetooth()*

Cette fonction sera appelée à chaque fois que l'on veut examiner s'il y a une réception bluetooth à traiter. La fonction renverra un caractère (qui correspond à un code reçu) qui devra être communiqué comme paramètre à la fonction action_bluetooth().

On écrira alors :

```
//...
code = lire_bluetooth();
action_bluetooth(code);
//
...
```

• *Les fonctions de déplacement du robot : arret(), avance(), ...*

Ces fonctions seront appelées à chaque fois qu'on veut influencer la manière dont le robot se déplacera. Ces fonctions pourront être appelées directement au sein du programme principal (si le robot est en mode "autonome") ou au travers des fonctions qui prennent en charge les actions de pilotage bluetooth du robot (si le robot est en mode télécommandé).
En mode "télécommandé", la fonction *action_manette_directionnelle()* se charge de mettre à jour l'état de déplacement du robot sur base du caractère reçu par la connexion bluetooth. Cette fonction prend en charge une variable qui contient l'état de déplacement du robot de manière à effectuer un déplacement en accord avec la façon dont on pilote le robot.

• *Les fonctions action_manette_directionnelle() et action_bluetooth()*

La fonction *action_manette_directionnelle()* sert d'utilitaire et n'est pas prévue pour être modifiée ni appelée directement. En effet, elle est appelée au travers de la fonction *action_bluetooth()* dans le cas où le caractère de réception bluetooth correspond à une action de la manette directionnelle.

La fonction *action_bluetooth()* se charge d'appeler la fonction qui prend en charge l'action liée à la commande de réception bluetooth. Cette fonction n'est pas prévue pour être modifiée mais simplement appelée après avoir récupéré le caractère de commande bluetooth à l'aide de la fonction *lire_bluetooth()* (voir ci-avant).

• *Les autres fonctions d'actions du robot*

La fonction action_bluetooth() se charge d'appeler la fonction qui gère l'action liée à la commande de réception bluetooth.

Nous avons expliqué les actions liées à la manette directionnelle mais il reste les actions liées aux autres "touche", à savoir :

"Touche" de la télécommande	Déclaration de la fonction associée
Le carré	`void action_carre(void);`
Le triangle	`void action_triangle(void);`
La croix	`void action_croix(void);`
Le rond	`void action_rond(void);`
La "touche" select	`void action_select(void);`
La "touche" start	`void action_start(void);`

Ces actions sont prévues pour être complétées / modifiées.
Il faut cependant faire attention si vous désirez agir sur le déplacement du robot; il faut prévoir de mettre à jour la variable globale ETAT_DEPLACEMENT pour qu'elle contienne l'information correcte de déplacement du robot.

Que contient le fichier relus3base.ino ?

Avec l'environnement Arduino, le programme principal se trouve au sein d'un fichier d'extension ".ino" et s'appelle un "skecth".
Lorsqu'on crée un nouveau "sketch", il apparaît une fenêtre avec deux fonctions vides, la fonction *setup()* et la fonction *loop()* :

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

C'est en réalité dans un autre fichier caché (écrit en C++) que sont appelées ces fonctions. Ce fichier se charge d'appeler la fonction *setup()* une fois au début du programme et la fonction *loop()* au sein d'une boucle infinie.
Autrement dit, notre "skecth" contiendra les actions initiales au sein du "setup()" et les actions à répéter au sein du "loop()".

Le contenu du "sketch" relus3base illustre un exemple didactique d'utilisation de certaines fonctions.

Contenu du "sketch" relus3base :

```
#include "relus3.h"  // on cherche dans le dossier courant
#include <SoftwareSerial.h>
#include <MsTimer2.h>

//declaration des variables du programme principal
char code = 0;         //réception BT
int distance = 0;

void setup() {

  init_signaux();
  Serial.begin(9600);  // pour utiliser la console
  init_bluetooth();
  arret();

  while(! bp1())
    {/*bloc d'action vide */ }

  while(dist_US()>30)
    {
    avance();
    }
   //obstacle à 30cm max
   arret();
   delay(2000);

   digitalWrite(REL_D, HIGH);
   digitalWrite(LED_A, HIGH);
   delay(150);
   digitalWrite(REL_D, LOW);
   digitalWrite(LED_A, LOW);
   delay(150);
}

void loop()
  {
    code = lire_bluetooth();
    action_bluetooth(code);
  }
```

Commentons brièvement le programme :

"Sketch" relus3base	Petit descriptif

`#include "relus3.h"`
`#include <SoftwareSerial.h>`
`#include <MsTimer2.h>`
	On inclus les librairies nécessaires

`char code = 0;` `int distance = 0;`	On déclare les variables
`void setup() {`	On rentre dans le setup()

`init_signaux();`
`Serial.begin(9600);`
`init_bluetooth();`
`arret();`
	On initialise les signaux, la console, le bluetooth et on arrête le robot.

`while(! bp1())` `{/*bloc d'action vide */ }`	On attend l'enfoncement de BP1

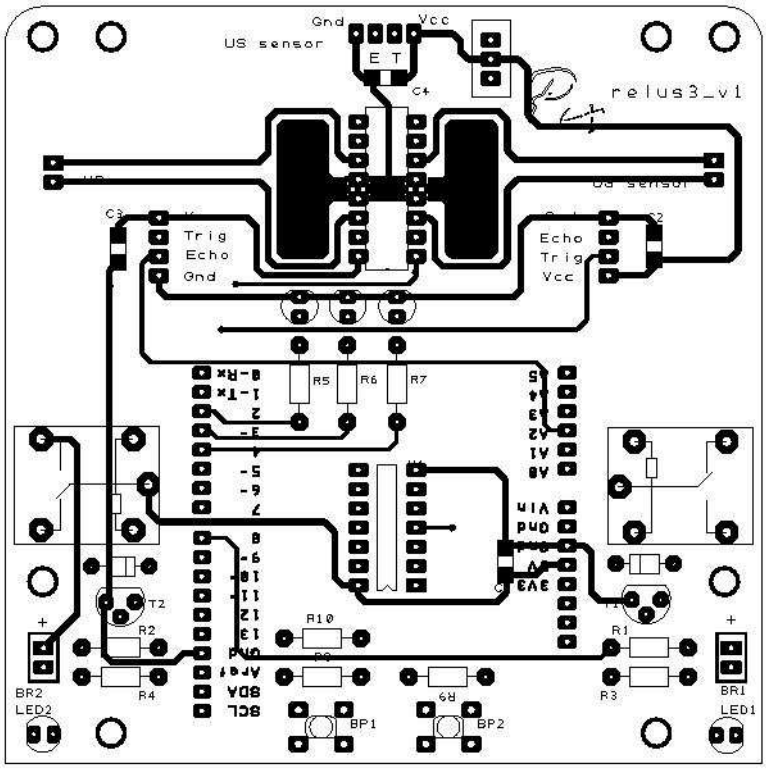
`while(dist_US()>30)`
`{`
`avance();`
`}`
`//obstacle à 30cm max`
`arret();`
`delay(2000);`
	On avance tant qu'on n'a pas d'obstacle inférieur ou égal à 30 cm
	
	Après un obstacle on arrête le robot et on attend 2 secondes.

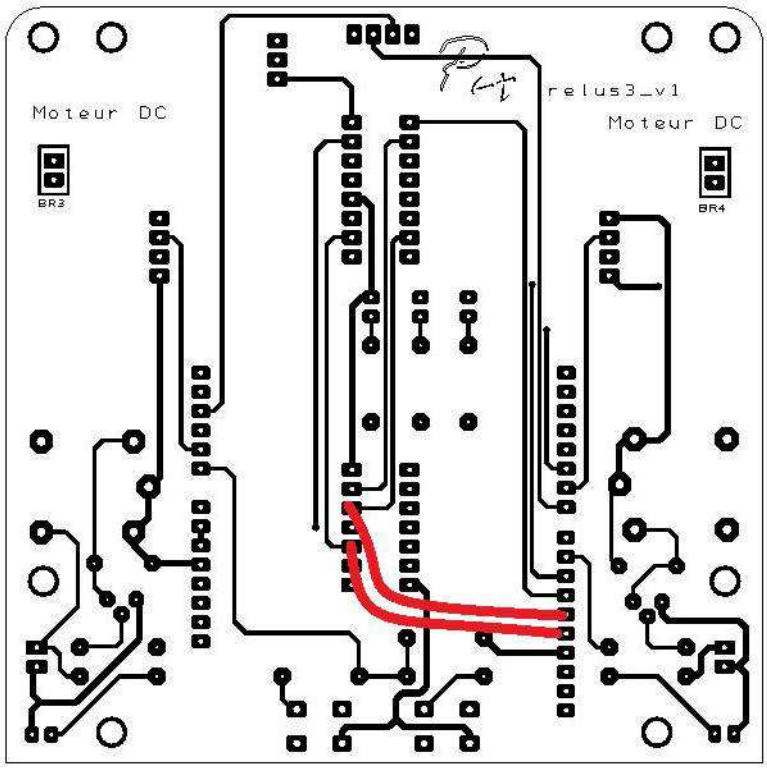
`digitalWrite(REL_D, HIGH);`
`digitalWrite(LED_A, HIGH);`
`delay(150);`
`digitalWrite(REL_D, LOW);`
`digitalWrite(LED_A, LOW);`
`delay(150);`
	On fait des petites actions liées au relais droit et à la LED A

`}`	On sort du setup()
`void loop()` `{`	On rentre dans le loop()

`code = lire_bluetooth();`
`action_bluetooth(code);`
`}`
	On gère une commande bluetooth en permanence.

DESSIN DU CIRCUIT IMPRIME (PCB)

Sérigraphie "TOP" et Cuivre "TOP"


Sérigraphie "BOTTOM" et Cuivre "BOTTOM" + pontages à réaliser


Relus3_v1 : Liste des composants pour inventaire / partie principale

Type de composants	Valeur/ référence	Ref schéma	Commentaire / description
Résistances	470 Ω	R3, R4	Protection des LEDS "clignotants" des relais
	470 Ω	R5, R6, R7	Protection des 3 LEDS de "debug" de la carte
	1,8 kΩ	R1, R2	Résistances de base du transistor en commutation
	47 kΩ	R9, R10	Résistances du clavier analogique
	100 kΩ	R8	Résistance du clavier analogique
Condensateurs	100 nF	C1	Découplage de l'alimentation de l'Arduino (CMS 1206)
	100 nF	C2, C3, C4	Découplage des capteurs US (ultrasons) (CMS 1206)
Diodes	1N4148	D1, D2	Diodes de protection du relais
LEDS	Rouge 3mm ou 5mm	LED1, LED2	LEDS "clignotants" des relais
	Au choix	LED3, LED4, LED5	LEDS de "debug" de la carte
Transistors	BC547	T1, T2	Bipolaires NPN en commutation pour les relais
CI	74 HC14	U1	Circuit inverseur (DIL16) pour la commande des moteurs
	L293D	U2	Ponts en H (DIL16) pour les moteurs
Capteurs	HC SR04	U3, U4, U5	capteurs US (ultrasons)
Relais	SRD-05VDC-SL-C	Rel1, Rel2	Relais pour les "clignotants"
Boutons-poussoirs	4 pins / N.O.	BP1, BP2	Boutons de commande
Interrupteur	3 contacts	CONN1	ON/OFF de la commande moteurs
connecteur	Femelle 2 contacts	BR1, BR2	Pour extension extérieure des relais
	Femelle 2 contacts	BR3, BR4	Connexion des moteurs gauche (BR4) et droit (BR3)
	Femelle 4 contacts	U3, U4, U5	1 rangée, 4 contacts
	Connecteurs arduino	IC1	1 rangée, double orientation (1x6, 2x8, 1x10)
Arduino	Arduino uno R3	IC1	Carte de développement Arduino Uno
Divers (hors PCB)	Base roulante		Kit base roulante (Châssis+moteurs DC+roues+roue libre)
	Mâle 2 contacts		Connexion des moteurs gauche (BR4) et droit (BR3)
	Support piles		Support 6*AA
	Jack male		Jack DC mâle 2,1mm x 5,5mm
	Câble multibrins		Câblage Jack DC et moteurs
	Entretoises, écrous, vis		4 entretoises M3/ 20mm + vis M3 + écrous M3

LISTE DES COMPOSANTS PRINCIPAUX

Relus3_v1 : Liste des composants pour inventaire / matériel supplémentaire

Type de composants	Valeur/ référence	
Supports DIL14 et DIL16	Supports DIL pour U1 et U2	
PCB relus_3	Circuit imprimé relus3 (dessiné par J. Pochet)	
Gaine thermorétractable	1mm / 2mm / 3 mm	
Extension "télécommande" :		
Plaque d'essai arduino uno	PCB compatible arduino Uno avec petite breadboard	
Module bluetooth esclave	HC06	
Deux résistances différentes	3,9 kΩ et 6,8 kΩ : cf transformation 5V ↔ 3V3	
Matériel de soudure :		
Fer à souder + support	Environ 25W avec panne assez fine	
Soudure	Avec plomb si possible / diamètre 0,6mm	
Eponge naturelle	Naturelle pour supporter la chaleur du fer à souder	
Petite pince coupante	Petite pour précision	
Petite pince à long bec	Le plus fin le mieux	
Pompe à dessouder		
Gabarit pliage composants	Pour plier des résistances, ...	
Matériel électrique :		
Piles AA	1,5 V ou 1,2V rechargeable NiMH ou NiCd	
Chargeur de piles	6*AA pour NiMH et NiCd	
Multimètre	Tension, continuité, mesure de résistance et capacités	
Alimentation	DC stabilisé variable de 0 à 15 V, 2A	
Câbles "banane" et croco	banane ↔ croco, croco ↔ croco, banane ↔ banane	
Breadboard	Plaque d'essai classique	
Câbles monobrins	Noir/rouge/couleurs + ponts pour breadboards	
Câbles signaux	Câbles ♀↔♂, ♀↔♀, ♂↔ ♂ pour connexions arduino	

COMPOSANTS SUPPLEMENTAIRES