RADIOCOMMANDE ET RECEPTEUR

Esteban Chevalier-Drevon

Matyas Ricci

Groupe F

La radiocommande et son récepteur

1. Appairage de la télécommande:

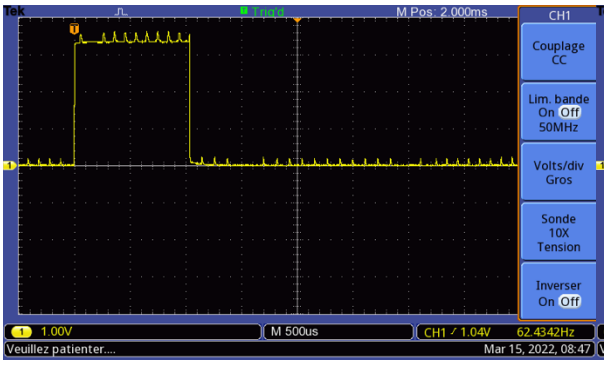
En vous aidant de la documentation présente dans la fiche de synthèse, rédiger la procédure d'appairage de la télécommande et de son récepteur.

1) Vérifier que le robot et la télécommande sont éteints
2) Connecter les broches CH3 et Bind du récepteur ensemble. Connecter aussi l'alimentation au récepteur
3) Allumer le robot
4) Tout en appuyant sur le bouton bind du récepteur, brancher l'alimentation de ce dernier. Vérifier que la led clignote sur le récepteur. Si ce n'est pas le cas, recommencer l'étape 4.
5) Allumer la télécommande tout en maintenant appuyé la touche « Bind » de cette dernière
6) Éteindre la télécommande et le robot
7) Déconnecter les broches CH3 et bind
8) Allumer le robot, la télécommande est désormais appairée

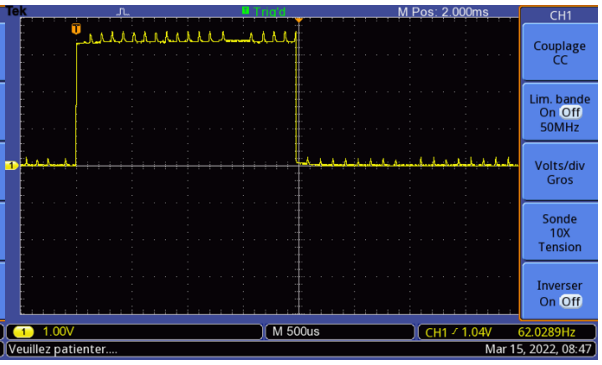
2. Câblage du récepteur et test

Relever les caractéristiques des signaux de chaque voie sur un oscilloscope pour chaque position extrême de la commande associée.

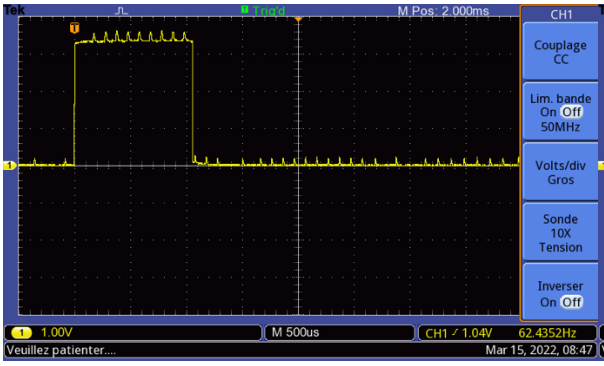
Voie 1 : Minimum



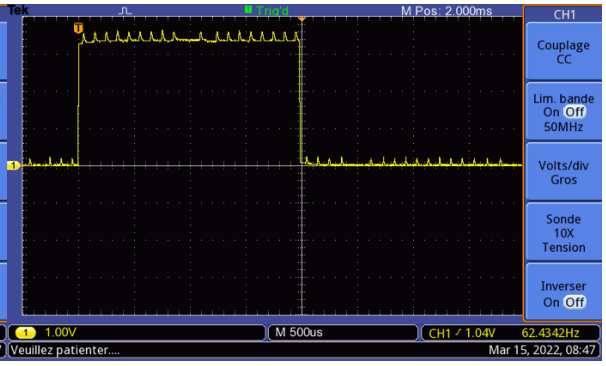
Voie 1 : Maximum



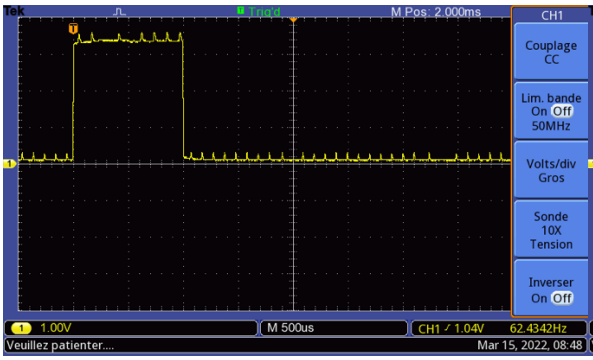
Voie 2 : Minimum



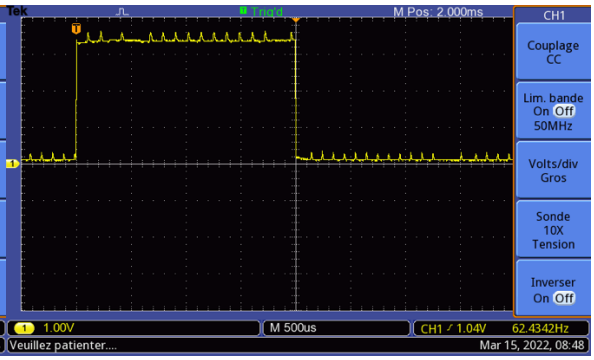
Voie 2 : Maximum



Voie 3 : Minimum



Voie 3 : Maximum



3. Appairage et commande d'un servo moteur

Fonctionnement vérifié et validé pour toutes les voies. Le servo moteur réagit aux commandes émises par la télécommande.

4. Programme 1: Acquisition des signaux de la radiocommande

Afficher sur l'écran LCD (et / ou le moniteur série) les valeurs issues des 3 voies (Th en micro secondes). Pour les voies 1 et 2, régler les trim de la radiocommande pour avoir un point milieu à 1500µs.

On pourra noter les variables *tvitesse* pour la gachette et *tdirection* pour la molette. En résumé les actions à réaliser en fonction de la valeur des temps relevée sont les suivantes :

tvitesse (µs) (*gachette*)	**1000**	**1500**	**2000**
	Recule (Backw)	Arrêt (Stop)	Avance (Forw)
tdirection (µs) (*molette*)	**1000**	**1500**	**2000**
	Gauche	Tout droit	Droite

Programme :

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2); // Addressage de notre écran LCD à l'addresse 0x27

#define VOIE1 8
#define VOIE2 9
#define VOIE3 10

void setup() {
    Serial.begin(9600);
    lcd.init();
    lcd.backlight();
    lcd.setCursor(0,0);
    lcd.print("V1:");
    lcd.setCursor(5,0);
    lcd.print("V2:");
    lcd.setCursor(10,0);
    lcd.print("V3:");
    pinMode(VOIE1, INPUT);
    pinMode(VOIE2, INPUT);
    pinMode(VOIE3, INPUT);

}
void loop() {
    static unsigned long int t0;
    if (millis() - t0 > 300) {
        unsigned long voie1 = pulseIn(VOIE1, HIGH);         // La fonction pulseIn renvoie le temps haut
        unsigned long voie2 = pulseIn(VOIE2, HIGH);         // d'un signal si en 2ème paramètre se trouve
        unsigned long voie3 = pulseIn(VOIE3, HIGH);         // "HIGH". En premier paramètre on entre
        t0 = millis();                                      // le pin associé à notre signal.
        lcd.setCursor(0,1);                                 // Placement du curseur sur la 2ème ligne
        lcd.print((String)voie1 + "  " + voie2 + "  " + voie3 + " "); // Affichage de nos 3 valeurs sur toute la ligne.
    }
}
```

5. Programme 2: Loi de commande – Gestion de la radiocommande:

Acquérir les voies 1 et 2 et traiter les données reçues pour déterminer le mode de fonctionnement (Forw, Backw ou Stop) et la puissance de chaque moteur (à afficher sur le LCD).

Ces deux informations vont jouer sur les **vitesses des moteurs** gauche et droite (**PowerG** et **PowerD**). La gachette va gérer la vitesse et la molette l'écart entre la vitesse de la roue droite/roue gauche.

Pour chacune des variables, le plus simple est de ramener la valeur centrale à 0 :

tvitesse _0 (µs)	**-500**	**0**	**500**
	Recule (Backw)	Arrêt (Stop)	Avance (Forw)
tdirection _0 (µs)	**-500**	**0**	**500**
	Gauche	Tout droit	Droite

Plusieurs solutions sont possibles, nous pouvons choisir d'exprimer la vitesse sous la forme :

POWERG=Vitesse$_{_gachette}$*CoeffG$_{_molette_0}$

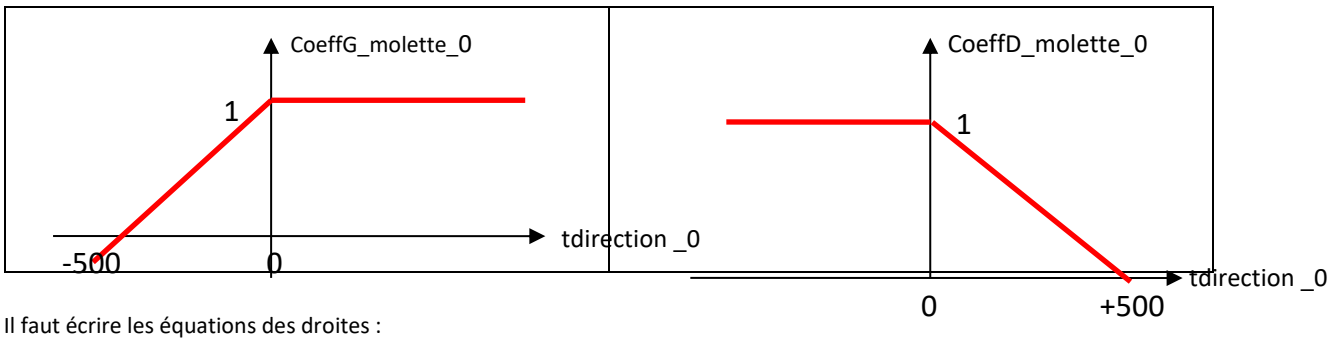
POWERD= Vitesse$_{_gachette}$*CoeffD$_{_molette_0}$

Pour modéliser la gâchette, nous pouvons prendre un modèle linéaire. Par exemple :

tvitesse _0	Vitesse_gachette
500	**100**
250	**50**
0	**0**
-250	**-50**
-500	**-100**

Remarque une vitesse négative signifie qu'il faut reculer

Pour modéliser les coefficients coeffG_molette_0 et coeffD_molette_0, on peut choisir ce type de courbe qui permet de freiner une roue par rapport à l'autre :



Il faut écrire les équations des droites :

- Vitesse_gachette=f (tvitesse _0)
- CoeffG_molette_0=f (tdirection _0)
- CoeffD_molette_0=f (tdirection _0)

On pourra utiliser la fonction *map* qui réalise une opération de proportionnalité.

Programme :

```
void loop() {
    static unsigned long int t0;
    unsigned long voie1 = pulseIn(VOIE1, HIGH);
    unsigned long voie2 = pulseIn(VOIE2, HIGH);
    unsigned long voie3 = pulseIn(VOIE3, HIGH);
    short RFactor = 100;
    short LFactor = 100;
    short RSpeed = 0;
    short LSpeed = 0;
    short mappedVoie2 = map(voie2, 1000, 2010, -100, 100);
    short mappedVoie1 = map(voie1, 1000, 2010, -100, 100);
    if (mappedVoie1 <= -5) { // Turn right
        LFactor = map((100 - abs(mappedVoie1)), 0, 100, 30, 100);
        RFactor = 100;
    } else if (mappedVoie1 >= 5) { // Turn left
        RFactor = map((100 - abs(mappedVoie1)), 0, 100, 30, 100);
        LFactor = 100;
    } else {
        RFactor = 100;
        LFactor = 100;
    }

    if (mappedVoie2 <= -10 || mappedVoie2 >= 10) {
        RSpeed = (RFactor / 100.0) * mappedVoie2;
        LSpeed = (LFactor / 100.0) * mappedVoie2;
    } else {
        RSpeed = 0;
        LSpeed = 0;
    }

    if (RSpeed >=0) {
        cmd_motor(1, 1, RSpeed);
    } else {
        cmd_motor(1, 0, abs(RSpeed));
    }

    if (LSpeed >= 0) {
        cmd_motor(0, 1, LSpeed);
    } else {
        cmd_motor(0, 0, abs(LSpeed));
    }

    if (millis() - t0 > 300) {
        t0 = millis();
        lcd.setCursor(0,0);
        lcd.print((String)voie1 + "  " + voie2 + "  " + voie3 + " ");
        lcd.setCursor(0,1);
        lcd.print((String)LSpeed + "  " + RSpeed + "           ");
    }
}
```

6. Programme 3: Auto/Manu et pilotage des moteurs

Faire le programme qui réalise les actions suivantes :

- Acquérir la voie 3
- Si le mode manu est détecté :
  - Acquérir les voies 1 et 2
  - Traiter les données pour déterminer le mode de fonctionnement (Forw, Backw ou Stop) et la puissance de chaque moteur
  - Exécuter les fonctions permettant de commander les deux moteurs

L'architecture du programme est la suivante :

```
void loop()
 {  int k,…;
   currentMillis = millis();
   if((currentMillis - previousMillisRadiocommande) > intervalRadiocommande)
      { // Acquisition Radiocommande
       }

   if (Manu)
     { //Gestion Radiocommande
        cmd_motor(right, SensD, powerD) ;
        cmd_motor(left, SensG, powerG) ;
     }
    else
     {
     }
}
```

Programme :

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2); // Addressage de notre écran LCD à l'addresse 0x27

#define VOIE1 8
#define VOIE2 9
#define VOIE3 10

#define In1 2
#define In2 3
#define In3 4
#define In4 7
#define EnA 5
#define EnB 6

#define DROIT 1
#define GAUCHE 0
#define STOP 0
#define AVANCE 1
#define RECUL 0

void cmd_motor(bool motor, bool direction = 0, short power = 0);

void setup() {
    Serial.begin(9600);
    lcd.init();
    lcd.backlight();
    pinMode(VOIE1, INPUT);
    pinMode(VOIE2, INPUT);
    pinMode(VOIE3, INPUT);
    pinMode(In1, OUTPUT);
    pinMode(In2, OUTPUT);
    pinMode(In3, OUTPUT);
    pinMode(In4, OUTPUT);
    pinMode(EnA, OUTPUT);
    pinMode(EnB, OUTPUT);

}
void loop() {
    static unsigned long int t0;
    unsigned long voie3 = pulseIn(VOIE3, HIGH);
    if (voie3 > 1500) { // mode Manuel
        unsigned long voie1 = pulseIn(VOIE1, HIGH);
        unsigned long voie2 = pulseIn(VOIE2, HIGH);
        short RFactor = 100;
        short LFactor = 100;
        short RSpeed = 0;
        short LSpeed = 0;
        short mappedVoie2 = map(voie2, 1000, 2010, -100, 100);
        short mappedVoie1 = map(voie1, 1000, 2010, -100, 100);
        if (mappedVoie1 <= -5) { // Turn right
            LFactor = map((100 - abs(mappedVoie1)), 0, 100, 30, 100);
            RFactor = 100;
        } else if (mappedVoie1 >= 5) { // Turn left
            RFactor = map((100 - abs(mappedVoie1)), 0, 100, 30, 100);
            LFactor = 100;
        } else {
            RFactor = 100;
            LFactor = 100;
        }

        if (mappedVoie2 <= -10 || mappedVoie2 >= 10) {
            RSpeed = (RFactor / 100.0) * mappedVoie2;
            LSpeed = (LFactor / 100.0) * mappedVoie2;
        } else {
```

```
            RSpeed = 0;
            LSpeed = 0;
        }

        if (RSpeed >=0) {
            cmd_motor(1, 1, RSpeed);
        } else {
            cmd_motor(1, 0, abs(RSpeed));
        }

        if (LSpeed >= 0) {
            cmd_motor(0, 1, LSpeed);
        } else {
            cmd_motor(0, 0, abs(LSpeed));
        }

        if (millis() - t0 > 300) {
            t0 = millis();
            lcd.setCursor(0,0);
            lcd.print((String)voie1 + "  " + voie2 + "  " + voie3 + " ");
            lcd.setCursor(0,1);
            lcd.print((String)LSpeed + "  " + RSpeed + "           ");
        }
    } else {

    }
}


void cmd_motor(bool motor, bool direction = 0, short power = 0) {
    float kfactor = 1;
    if (motor) { // Moteur droit
        power = (power * 2.55) * kfactor;
        if (direction) { // marche avant
            digitalWrite(In1, LOW);          // On paramètre le pont en H afin que le moteur droit
            digitalWrite(In2, HIGH);         // soit configuré de manière à avancer.
            analogWrite(EnB, power);         // On applique une PWM de rapport cyclique définit par la variable power pour le moteur droit.
        } else { // Marche arrière
            digitalWrite(In2, LOW);          // On paramètre le pont en H afin que le moteur droit soit
            digitalWrite(In1, HIGH);         // configuré de manière à reculer.
            analogWrite(EnB, power);         // On applique une PWM de rapport cyclique définit par la variable power pour le moteur droit.
        }
    } else { // Moteur gauche
        power = (power * 2.55);
        if (direction) { // Marche avant
            digitalWrite(In3, HIGH);         // On paramètre le pont en H afin que le moteur gauche soit configuré
            digitalWrite(In4, LOW);          // de manière à avancer
            analogWrite(EnA, power);         // On applique une PWM de rapport cyclique définit par la variable power pour le moteur droit.
        } else { // Marche arrière
            digitalWrite(In3, LOW);          // On paramètre le pont en H afijn que le moteur gauche soit configuré
            digitalWrite(In4, HIGH);         // de manière à reculer.
            analogWrite(EnA, power);         // On applique une PWM de rapport cyclique définit par la variable power pour le moteur gauche.
        }
    }
}
```