Rapport Projet Robot N°7

Jules Massart et Baptiste Mondion

Juin 2024

1 Cahier des charges

1.1 Explication du contrat

Nous avions le contrat numéro 7 à réaliser. L'objectif final de ce contrat est de :

- Contrôler via bluetooth un robot en le faisant avancer, reculer, aller à droite, et à gauche.
- Fixer la vitesse du robot à 20 cm/s.
- Permettre au robot de contourner un obstacle détecté par son sonar en suivant l'angle d'impact et en longeant l'obstacle.
- Permettre que le robot puisse être démarré et éteint via un bouton d'arrêt d'urgence, à tout moment par mesure de sécurité.
- Vérifier le niveau de batterie et allumer une led lorsque la tension de la carte de contrôle chute en dessous de 3V.
- Etre capable d'envoyer des messages de debug via UART.

1.2 Détails techniques

Ce robot est controllé par une carte NUCLEO-L476RG. Nous nous servirons des composants suivants :

- un sonar
- 4 servo moteurs pour les chevilles droite et gauche.
- Une puce bluetooth

Donner la fiche technique en deuspi

2 Conception

2.1 Fonctionnement général

Le robot démarre après une phase d'initialisation et entre dans une boucle infinie qui démarre ou arrête les roues en fonction de l'état du bouton B1. Il appelle en continu une fonction de détection de murs qui utilise un radar pour détecter et éviter les obstacles. De plus, plusieurs fonctions sont exécutées lors d'interruptions, permettant le contrôle du robot par Bluetooth. Une vérification constante du niveau de tension de la batterie est également effectuée : si cette dernière chute sous un seuil critique, une LED est allumée.

2.2 Organigramme

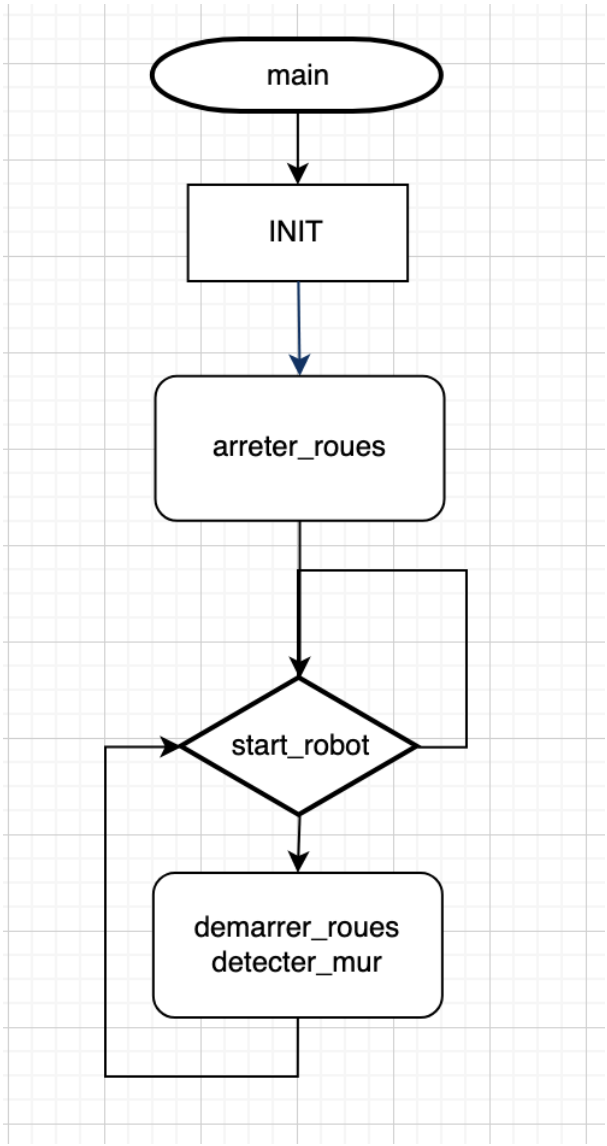

Figure 1: Organigramme de la fonction main

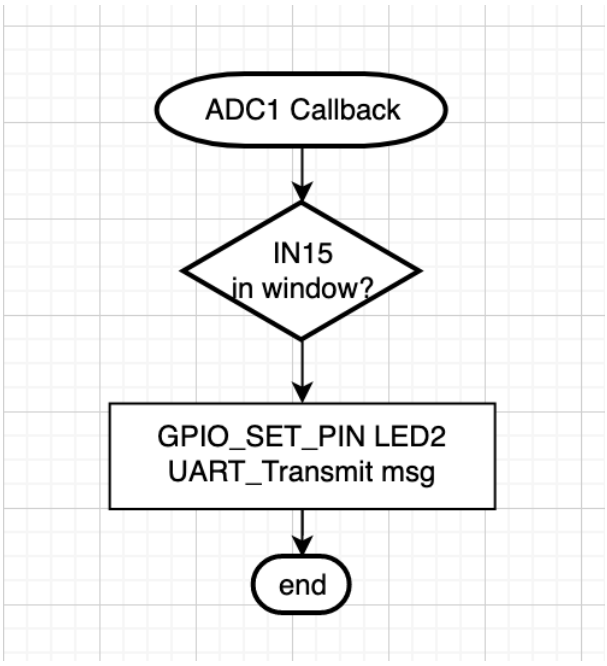

Figure 2: Callback ADC

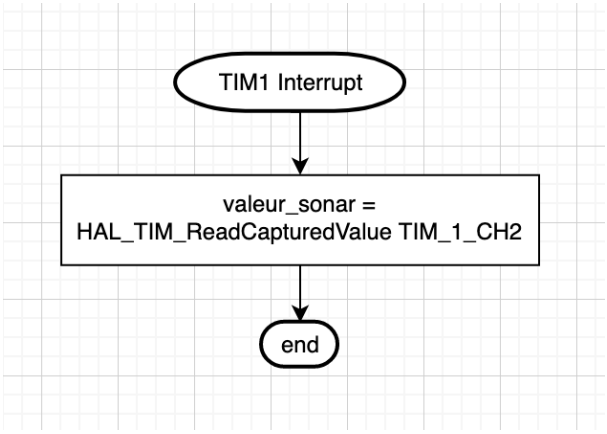

Figure 3: Callback Sonar

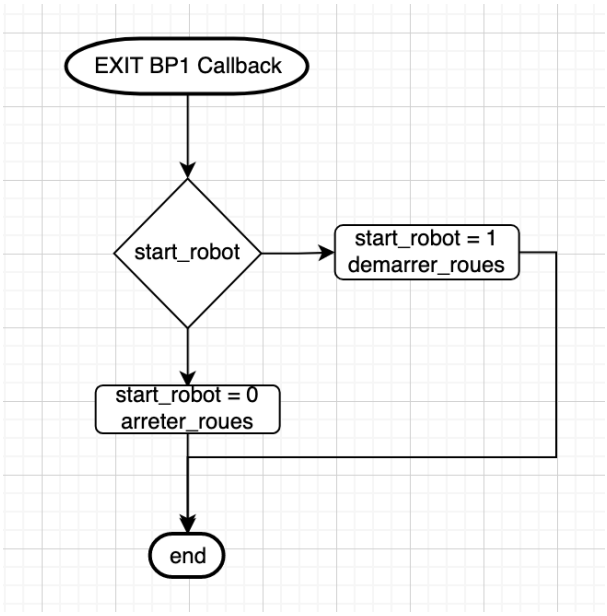

Figure 4: Callback Bouton B1

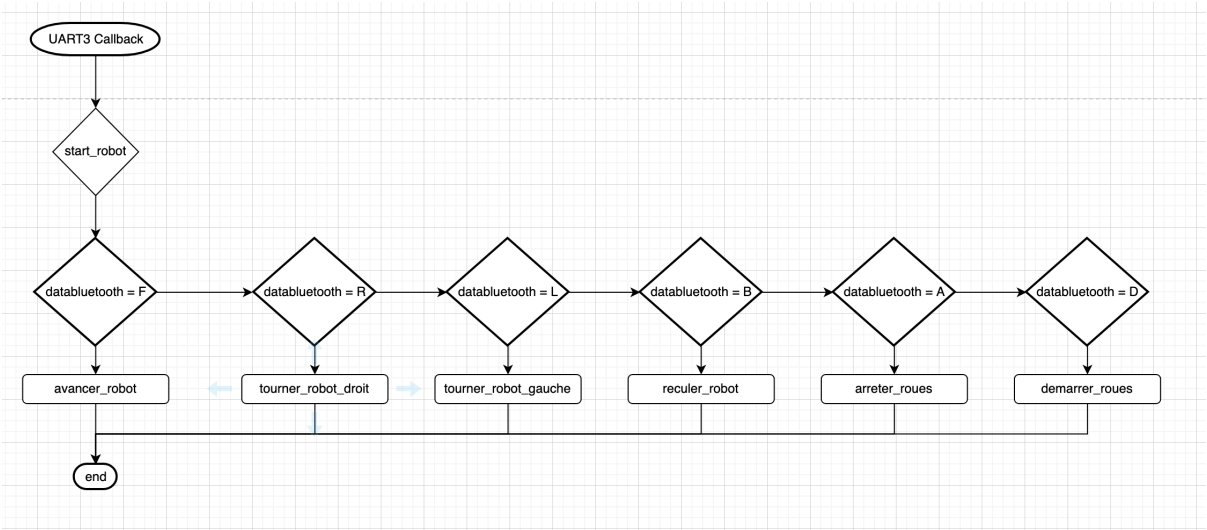

Figure 5: Callback Bluetooth sur l'UART3

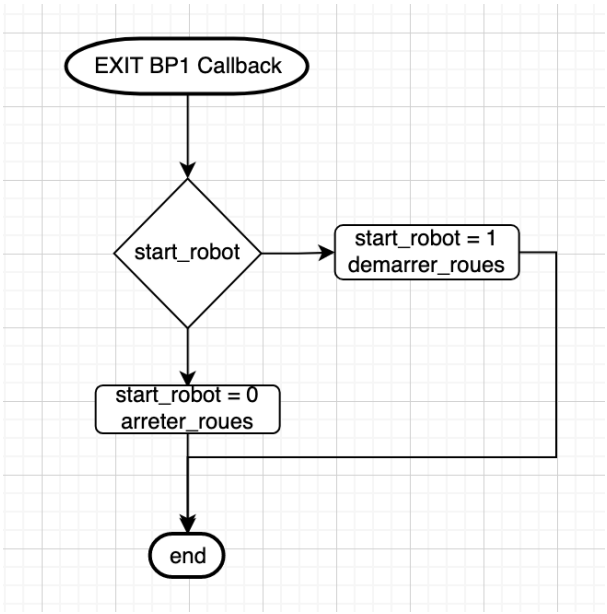

Figure 6: Timer de 100 ms

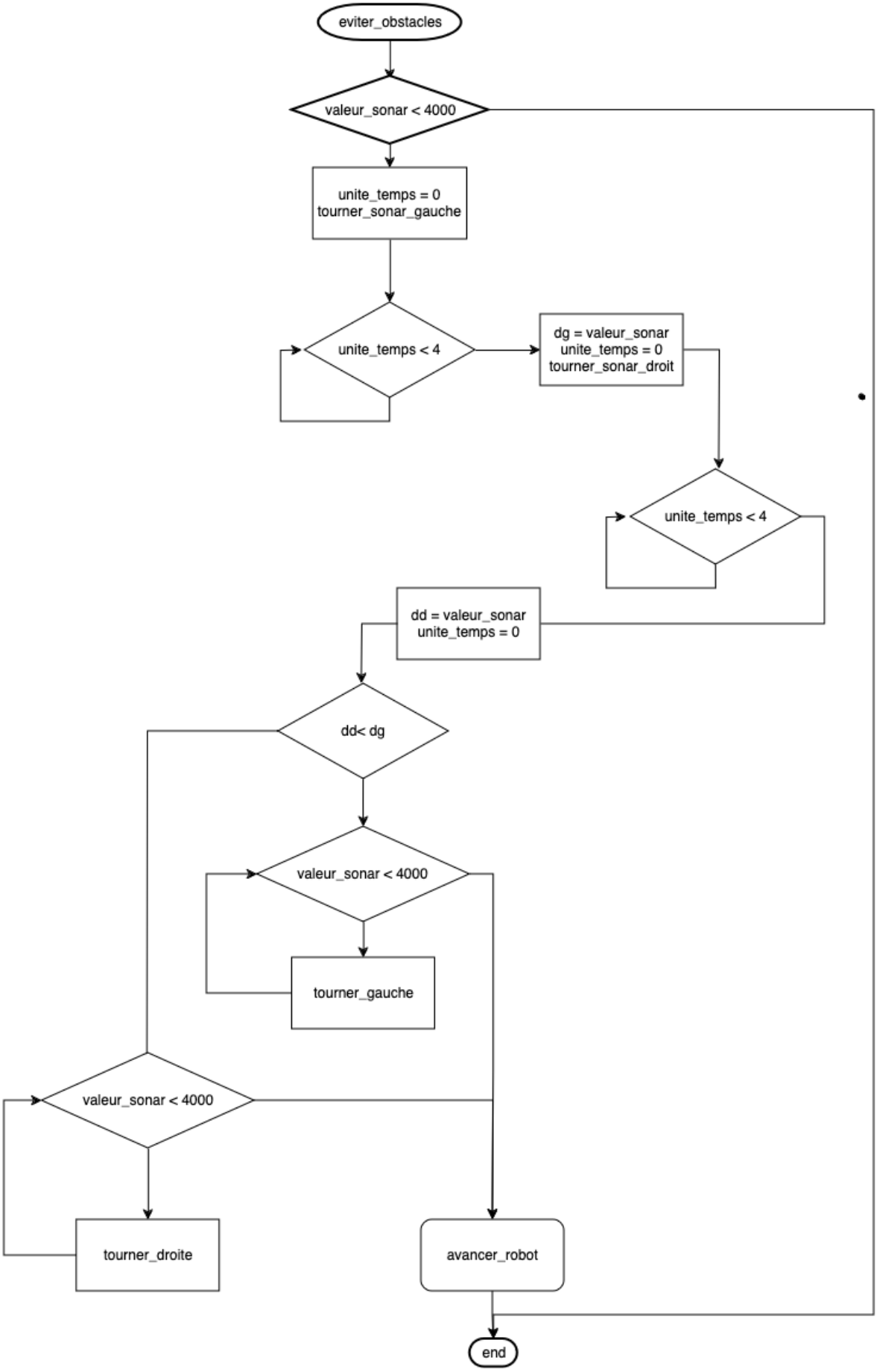

Figure 7: Fonction de detection et d'évitement d'un mur

2.2.1 Détecter un obstacle

On se réfère au schéma page 39. On veut un prescaler pour avoir une période de **37.5 ms** car c'est l'impulsion maximale détectable.

Voici le calcul pour trouver ce prescaler; il y a plusieurs combinaisons possibles de `ARR` et de prescaler, donc on fixe `ARR` à 2^{16}.

```
timer_clock = 1/(80*10**6)
arr = 2**16
periode = 0.0375
prescaler = (periode/(timer_clock*arr)) = 46.8
```

Il faut convertir cette impulsion en une distance.

Les ultrasons émis par l'émetteur du sonar rebondissent sur l'obstacle et reviennent sur le récepteur. La distance maximale détectable est alors :

```
distance_max = 340 m/s * 37.5 ms / 2 = 6.375 m
```

On peut alors calculer le rapport entre l'impulsion reçue et la distance séparant le robot d'un obstacle.

$$distance\_obstacle = \frac{impulsion\_recue \times distance\_max}{impulsion\_recue\_max} \times \frac{prescaler}{clock}$$

L'application numérique donne un **facteur 100** entre l'impulsion reçue et la distance de l'obstacle. On va donc garder la valeur brute récupérée par le sonar avec la fonction `HAL_TIM_ReadCapturedValue` pour éviter toutes imprécisions.

Toutes les **50 ms**, la variable globale `valeur_sonar` se met à jour via une interruption.

Nous avons aussi ajouté un filtre à **8** dans les Parameter Settings de `TIM1` pour rendre le sonar moins sensible aux mini perturbations. On n'oublie pas d'activer la `capture compare interrupt` sur `TIM1`.

2.2.2 Rotation du sonar

Le servomoteur relié au sonar est sur le Channel 4 du Timer 1. Il peut être contrôlé pour tourner à droite ou à gauche via la fonction `HAL_TIM_SET_COMPARE`.

Selon la documentation, la période du servomoteur est de 37.5 ms. Une impulsion de 1 ms correspond à -90°, 1.5 ms à 0° et 2 ms à +90°. On peut supposer une relation linéaire entre l'angle de rotation et la longueur de l'impulsion, exprimée par la formule suivante :

$$\phi(\theta) = \frac{\theta}{180} + 1$$

$$x = \phi(\theta) \times \frac{ARR}{Période_{PWM}}$$

- x : Valeur du registre de comparaison (CCR) qui détermine la largeur de l'impulsion pour un angle donné.
- $\phi(\theta)$: Temps d'impulsion calculé en millisecondes pour un angle θ donné, selon la relation linéaire $\phi(\theta) = \frac{\theta}{180} + 1$.
- ARR : Valeur du registre Auto-Reload (ARR) du timer, qui définit la période du signal PWM en ticks du timer.
- $Période_{PWM}$: Durée totale d'une période complète du signal PWM en millisecondes.

Détails des termes :

1. $\phi(\theta)$: Cette partie de la formule convertit l'angle θ en un temps d'impulsion spécifique selon la relation linéaire établie.
2. ARR : L'Auto-Reload Register (ARR) détermine le nombre total de ticks pour une période complète du PWM. Il est utilisé pour définir la fréquence du PWM.
3. $Période_{PWM}$: La période totale du PWM en millisecondes, basée sur la fréquence du timer et la valeur du prescaler.

Cette formule permet de déterminer la valeur du registre de comparaison (x), ajustant ainsi la largeur de l'impulsion en fonction de l'angle souhaité pour le servomoteur, en respectant la période définie par l'ARR du timer.

```
ARR = 2**16
PWM_period = 37.5
theta = 0
phi_theta = (theta / 180.0) + 1.5
x = phi_theta * (ARR / PWM_period)
```

Ce qui donne :

θ	x
0°	2600
45°	3058
-45°	2184

Table 1: Valeurs de x en fonction de θ

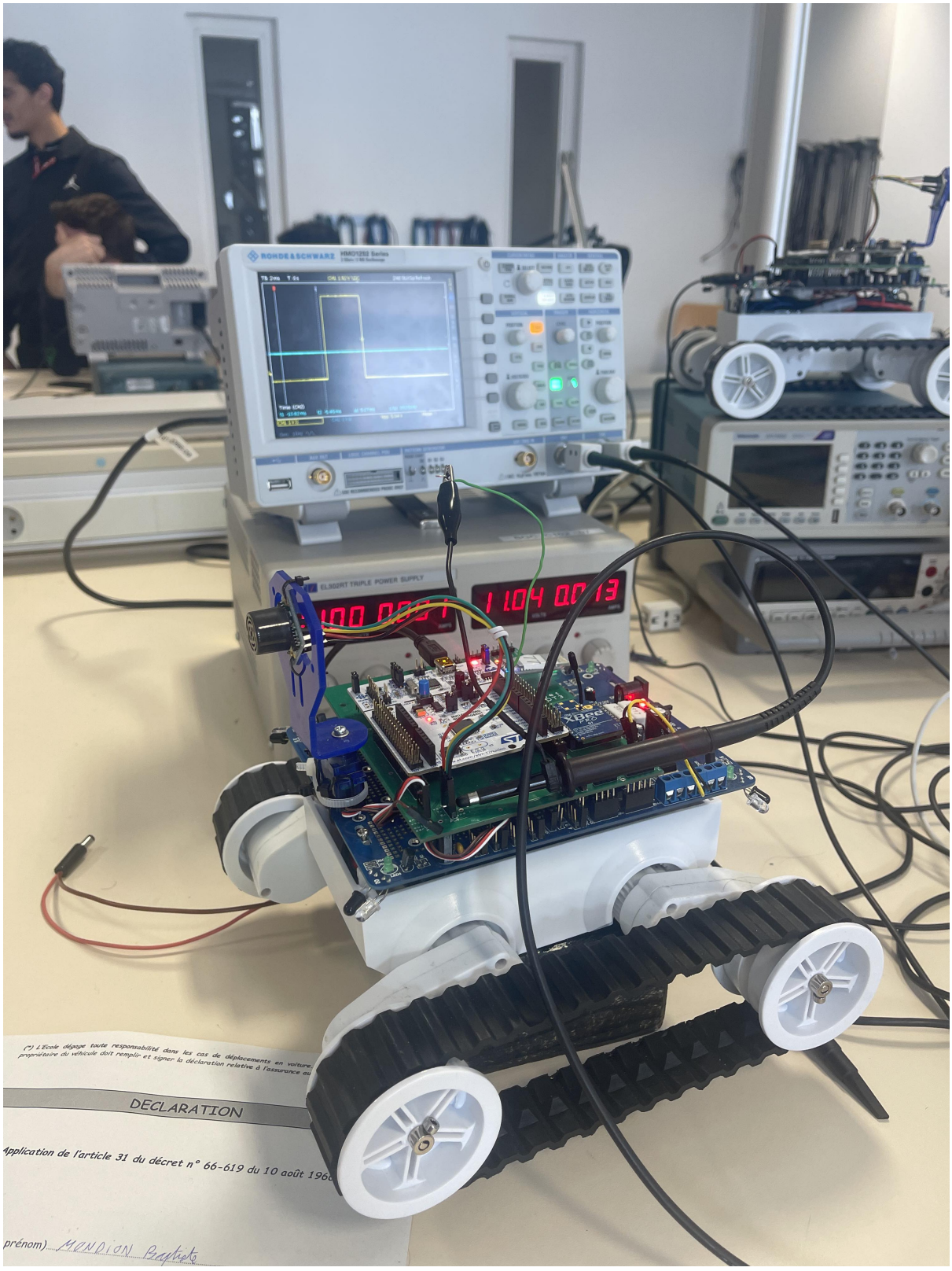

Figure 8: Capture de l'impulsion sur `triger_sonar` à l'oscilloscope

2.3 Les chenilles

L'énoncé nous précise qu'il faut une PWM à **1 kHz** ainsi qu'une vitesse de **20 cm/s**. Voici les pins à configurer:

- `DIRG` sur `PB2` en mode `GPIO_Output` qui gère la direction de la chenille gauche.
- `DIRD` sur `PC8` en mode `GPIO_Output` qui gère la direction de la chenille droite.
- `PWMD` sur `PB11` en mode Channel 4 du Timer 1 (`TIM1_CH4`) qui mesure la largeur de l'impulsion reçue.
- `PWMG` sur `PA15` en mode Channel 1 du Timer 1 (`TIM1_CH1`) qui mesure la largeur de l'impulsion reçue.

On va utiliser le couple `ARR = 80 000` et `prescaler = 1`. Selon la fiche technique, le robot est capable d'aller à 1 km/h soit **28 cm/s**. Par un produit en croix, on trouve un rapport cyclique de la PWM à **71%**, soit **57 000**. Empiriquement, on trouve plutôt **38 000** et **36 000** pour la chenille droite et gauche, du moins pour le robot 59.

On fixe le rapport cyclique de la PWM à **100%**, on peut donc faire avancer le robot à **20 cm/s** avec:

```
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 40000);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 40000);
```

On mesure la vitesse sur **2 mètres** et on trouve en moyenne **9.6 secondes**, soit un écart relatif de **5%**. Or, la précision demandée est de $\pm$**2 cm/s**, soit **10%**. Nous sommes donc dans le seuil de tolérance. Nous gérons les fonctions suivantes via `HAL_GPIO_WritePin` pour changer la direction des chenilles :

- **Tourner à droite**
- **Tourner à gauche**
- **Avancer**
- **Reculer**

Pour tourner à droite, par exemple, nous faisons avancer la chenille de gauche et reculer la chenille droite. Nous pouvons donc contrôler le robot à présent.

2.4 Connexion Bluetooth

Nous avons ajouté l'option de contrôler le robot via Bluetooth en utilisant les commandes implémentées précédemment.

Les étapes sont les suivantes :

1. Activer le Bluetooth sur l'`UART3` (Universal Asynchronous Receiver-Transmitter) à un Baud rate de **9600 bits seconde**.
2. Configurer le Pin `PC11` en mode `USART3_RX`.
3. Activer l'interruption globale pour recevoir des commandes envoyées par un client.
4. Enregistrer le caractère reçu dans la variable char `dataBluetooth` en utilisant :

```
HAL_UART_Receive_IT(&huart3, &dataBluetooth, 1);
```

Dans le callback de l'`UART3`, **HAL_UART_RxCpltCallback** , nous pouvons utiliser une disjonction de cas pour exécuter les commandes en fonction du caractère reçu.

- Exemple de cas pour avancer :

```
if (dataBluetooth == 'A') {
    avancer();
}
```

- Exemple de cas pour reculer :

```
if (dataBluetooth == 'R') {
    reculer();
}
```

Chaque commande reçue déclenche l'exécution de la fonction correspondante pour contrôler le robot. On utilise l'application Android, `Arduino RC Controller` pour télécommander notre robot.

2.5 Monitoring de la batterie

Pour contrôler la batterie, nous utilisons le convertisseur analogique numérique (ADC) pour analyser la tension d'entrée de la carte. Les étapes sont les suivantes :

1. Activer l'`ADC1` sur l'input 15 (`IN15`) en "single ended".
2. Activer les interruptions ainsi que l'analog watchdog1.
3. Choisir un seuil HIGH et LOW.

- Pour le seuil HIGH, nous le laissons au maximum : **4096** car la résolution est de **12 bits**.
- La tension par défaut pour la Nucleo 476RG est **3.3V**, donc **4096** représente **3.3V**.
- **3V** est donc représenté par **3723** par simple produit en croix.
- Une interruption est déclenchée via la fonction **HAL_ADC_LevelOutOfWindowCallback** si la tension d'entrée sort de cette fenêtre.
- Nous pouvons ensuite convertir la valeur de l'ADC en une tension via un nouveau produit en croix.

2.6 Bouton Démarrer/Arrêter

Cette partie a été rapide à réaliser car tout a fonctionné du premier coup et nous n'avons pas compris l'intérêt des filtres anti-rebonds car ce bouton bleu fonctionnait très bien. On associe le big blue button à `PC13` en mode `GPIO_EXTI13`. La fonction **"HAL_GPIO_EXTI_Callback"** met la variable `start_robot` à **1** si cette dernière est à **0** et **1** sinon. La valeur de `start_robot` est vérifiée avant d'exécuter une commande Bluetooth et avant de démarrer les roues.

2.7 Debug via le terminal sur UART2

L'**UART2** est configuré à 115200 bits/s. On peut récupérer les messages envoyés via la fonction **HAL_UART_Transmit** sur une console directement dans STM32CubeIDE via un port COM ou bien via PuTTY.

2.8 Horloge

On a remplacé les **"HAL_Delay"** utilisés dans les premiers essais par des boucles `while` grâce à une horloge de période **100 ms** sur le `TIM7` avec un prescaler à **122**.

```
timer_clock = 1/(80*10**6)
arr = 2**16
periode = 0.1
prescaler = (periode/(timer_clock*arr)) = 122.07
```

3 Réalisations et Tests

4 Conclusion

4.1 Améliorations

Nous avons décelé deux améliorations majeures que nous n'avons hélas pas eu le temps d'implémenter:

- Nous avons remarqué que la vitesse des chenilles diminuait lorsque la batterie se déchargeait. Nous avons penser à relever la vitesse absolue avec un rapport cyclique de **100%** tous les **0.5 V** de **7.2 V** à **5.6 V** afin de créer une fonction d'ajustement grâce à la tension de l'ADC. Cette méthode n'est pas forcément pertinente, car la vitesse dépend aussi d'autres facteurs, coefficient de résistance du sol, usure, etc... Il aurait fallut faire un asservissement PID. Pour assurer que notre robot maintienne une vitesse constante, nous aurions dû utiliser une méthode de régulation basée sur l'ajustement dynamique du rapport cyclique de la PWM. La procédure aurait été la suivante :

- **Mesure de la vitesse réelle** : Un capteur aurait été utilisé pour mesurer la vitesse réelle du robot en temps réel.
- **Calcul de l'erreur relative** : Nous aurions calculé l'erreur relative entre la distance attendue et la distance réelle mesurée. Cette erreur aurait été donnée par la formule :

$$Erreur = \frac{Vitesse\_réelle}{Vitesse\_attendue} \tag{1}$$

- **Ajustement du rapport cyclique** : En fonction de l'erreur relative, nous aurions ajusté le rapport cyclique de la PWM pour minimiser cette erreur. Le rapport cyclique aurait été multiplié par $\frac{1}{Erreur}$.
- **Boucle de rétroaction** : Ce processus aurait été exécuté de manière continue dans une boucle de rétroaction, permettant des ajustements constants pour maintenir la vitesse désirée.

Cette méthode aurait permis une régulation efficace de la vitesse du robot, en assurant qu'il s'adapte aux variations de charge et aux conditions de la batterie pour atteindre la vitesse cible aussi précisément que possible.

- Le radar renvoyait parfois de faux positifs. Nous aurions pu prendre trois mesures consécutives et vérifier que les trois dépassaient le seuil de **40 000** avant de rentrer dans une boucle, car parfois, le robot s'arrêtait dans le vide.

initialisation

```
// Initialisation de la biblioth que HAL
HAL_Init();

// Initialisation des GPIO
MX_GPIO_Init();

// Initialisation des timers
MX_TIM1_Init();
MX_TIM2_Init();
MX_TIM6_Init();

// Initialisation des p riph riques UART
MX_USART3_UART_Init();
MX_USART2_UART_Init();

// Initialisation de l'ADC
MX_ADC1_Init();

// D marrage des timers en mode base et PWM
HAL_TIM_Base_Start(&htim2);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);

// D marrage des interruptions pour le timer et l'input capture
HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_2);
HAL_TIM_Base_Start_IT(&htim6);

// Configuration initiale des GPIO
HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(trig_sonar_GPIO_Port, trig_sonar_Pin, GPIO_PIN_SET);

// R ception UART en interruption
HAL_UART_Receive_IT(&huart3, &dataBluetooth, 1);

// Initialisation de la variable de d marrage du robot
start_robot = 0;

// D marrage de l'ADC avec gestion d'erreur
if (HAL_ADC_Start_IT(&hadc1) != HAL_OK) {
    Error_Handler();
}

// Envoi d'un message via UART
char msg[] = "Robot d marr ";
HAL_UART_Transmit(&huart2, (uint8_t *) msg, sizeof(msg), HAL_MAX_DELAY);

// Arr ter les roues (fonction personnalis e)
arreter_roues();
```

main.c

```
void main(void){
/*inititialisation mentionnee ci-dessus*/
  while (1){
    if(start_robot){
      demarrer_roues();
      detecter_mur();
    }
  }
}
```

detecter_mur

```
void detecter_mur(){
    if(valeur_sonar < 4000){
        unite_temps = 0;
        arreter_roues();
        while(unite_temps < 4){
            sonar_tourner_droite();
        }
        uint32_t dd = valeur_sonar;
        unite_temps = 0;
        while(unite_temps < 4){
            sonar_tourner_gauche();
        }
        uint32_t dg = valeur_sonar;
        sonar_mettre_milieu();
        demarrer_roues();
        if(dd > dg){
          while(valeur_sonar < 4000){
                robot_tourner_droit();
            }
        }
        else{
             while(valeur_sonar < 4000){
                 robot_tourner_gauche();
             }
        }
        avancer_robot();
    }
}
```

commandes

```
1
2  void avancer_robot(){
3      HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
4      HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
5  }
6
7  void reculer_robot(){
8      HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_RESET);
9      HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_RESET);
10 }
11
12 void demarrer_roues(){
13     __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,80000);
14     __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 80000);
15 }
16
17 void arreter_roues(){
18     __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
19     __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
20 }
21
22
23 void sonar_tourner_droite(){
24     __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_4, 2400-700);
25 }
26
27 void sonar_tourner_gauche(){
28     __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_4, 2400+700);
29 }
30
31 void sonar_mettre_milieu(){
32   __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_4, 2400);
33 }
34
35 void robot_tourner_droit(){
36     HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_RESET);
37     HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
38 }
39
40 void robot_tourner_gauche(){
41       HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
42       HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_RESET);
43 }
```

Bluetooth_interrupt

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
  if (huart->Instance == USART3 && start_robot) {
      switch (dataBluetooth) {
          case 'F':
              avancer_robot();
              break;
          case 'R':
            robot_tourner_droit();
              break;
          case 'L':
            robot_tourner_gauche();
              break;
          case 'A':
             arreter_roues();
             break;
          case 'B':
             reculer_robot();
             break;
          case 'D':
             demarrer_roues();
             break;
          default:
              break;
      }
  }

  HAL_UART_Receive_IT(&huart3, &dataBluetooth, 1);
}
```

Sonar_interrupt

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){
  if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2){
    valeur_sonar = (uint32_t) HAL_TIM_ReadCapturedValue(&htim1, TIM_CHANNEL_2);
  }

}
```

100ms_interrupt

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM6){
        //periodElapsed=1;
        unite_temps++;
    }
}
```

Start/Stop_interrupt

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if (GPIO_Pin == B1_Pin) {
        uint32_t current_time = HAL_GetTick();
        if ((current_time - last_interrupt_time) > DEBOUNCE_DELAY) {
            last_interrupt_time = current_time;
            if (start_robot == 0) {
                start_robot = 1;
                demarrer_roues();
            } else if (start_robot == 1) {
                start_robot = 0;
                arreter_roues();
            }
        }
    }
}
```

ADC_interrupt

```
void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef *hadc) {
    if (hadc->Instance == ADC1) {
        HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        uint32_t adc_value = HAL_ADC_GetValue(hadc);
        float tension_batterie = ((float)adc_value / 4095) * 3.3;
        char msg[50];
        snprintf(msg, sizeof(msg), "Niveau batterie : %.2f V\n", tension_batterie);
        HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), HAL_MAX_DELAY);
        HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
    }
}
```