MINES Saint-Étienne | Institut Mines-Telecom

Conception des systèmes électroniques

Rapport de conception

Rapport Projet robot

Contrat 5 - Le robot suiveur d'une trajectoire filoguidée

Yasmine SALMOUNI
Yassmina BARA

Juin 2024

INSPIRING INNOVATION SINCE 1816

Table des matières

1 Introduction

1.1 Cahier des charges et spécifications - Contrat 5

1.1.1 Objectif du projet

Développement d'un robot capable de suivre une trajectoire prédéfinie à l'aide de la technologie des infrarouges qui pointent vers le bas. Le développement de ce projet sera réalisé avec le logiciel STM32CubeIDE, qui sera utilisé pour programmer la carte Nucleo L476RG associée au robot. Concernant la ligne que le robot doit suivre, du scotch blanc réfléchissant a été choisi en raison de son contraste avec le sol gris, moins réfléchissant. Ce choix assure une meilleure détection par les capteurs infrarouges du robot.

1.1.2 Spécifications techniques

- **PWM (Modulation de largeur d'impulsion)** : Le système doit utiliser une fréquence de 1 kHz pour le contrôle des moteurs.
- **Vitesse** : Le robot doit maintenir une vitesse constante de 10 cm/s.
- **Précision** : La précision du suivi de trajectoire doit être de ± 5 cm.

2 Conception

2.1 Algorigrammes

2.1.1 Main

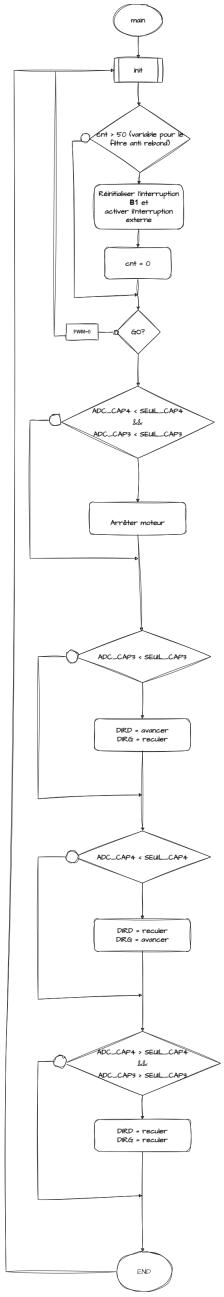

FIGURE 1 – Programme principal

Notons que les deux cateurs arrières sont utilisés pour faire avancer le robot. Donc, lorsque DIRD et DIRG valent RESET (équivalent de reculer sur l'organnigramme), le robot avance.

2.1.2 Interruptions

FIGURE 2 – Interruption du bouton B1_Pin

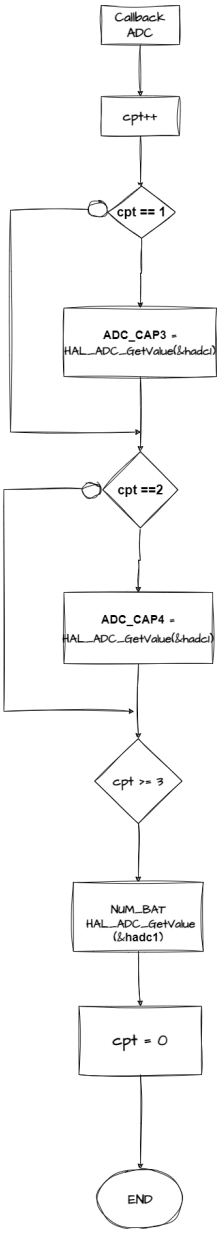

FIGURE 3 – Interruption ADC

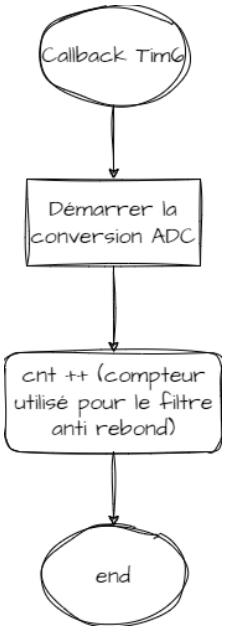

FIGURE 4 – Interruption du timer6

2.1.3 Fonction d'initialisation

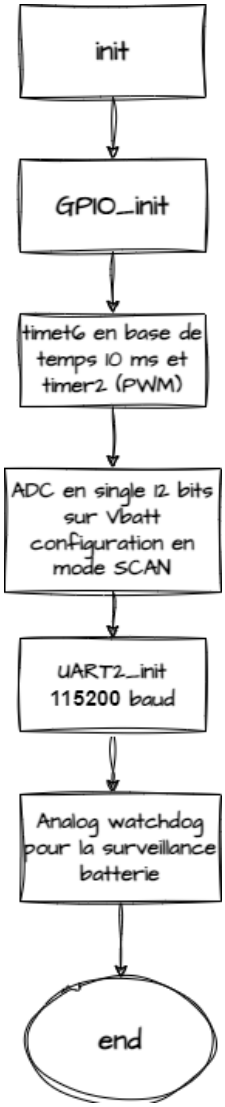

FIGURE 5 – Fonction d'initialisation

3 Réalisation du projet

3.1 Configuration IOC du projet

La configuration IOC globale du projet est la suivante :

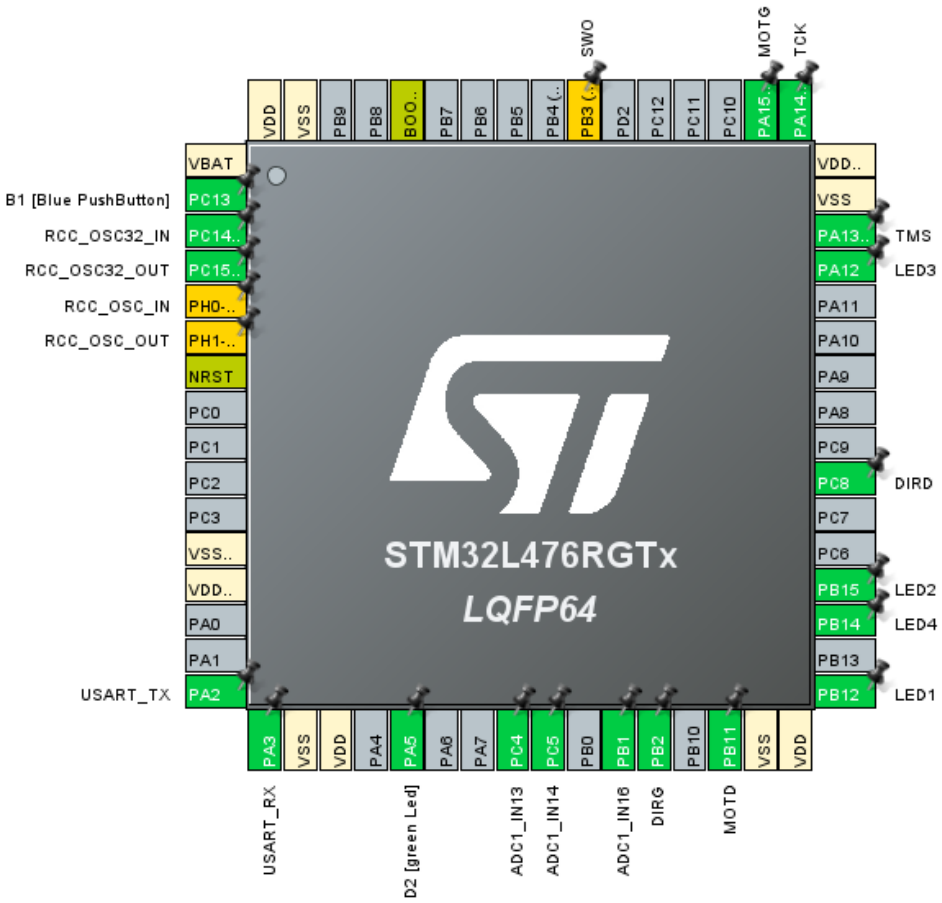

FIGURE 6 – Configuration IOC du projet

3.1.1 Configuration de l'ADC

Le projet implique l'utilisation de capteurs infrarouges qui émettent des rayons infrarouges et mesurent le flux de lumière réfléchie. Cette intensité est ensuite convertie en une valeur numérique. Afin de réaliser cette conversion, il est nécessaire de configurer les capteurs en mode ADC. De plus, il est impératif de surveiller l'état de la batterie, nécessitant également une configuration ADC du pin correspondant.

Configuration des capteurs infrarouges

Les capteurs infrarouges utilisés, en particulier les deux capteurs arrière, doivent être configurés pour convertir l'intensité lumineuse réfléchie en valeurs numériques utilisables. Pour ce faire, chaque capteur est relié à un canal ADC spécifique.

Gestion de la surveillance de la batterie
Le pin associé à la surveillance de la batterie est configuré en mode ADC pour permettre une lecture précise de la tension de la batterie. Cette lecture est essentielle pour garantir le fonctionnement optimal du système sans interruption due à une décharge excessive de la batterie.
Mécanisme de conversion ADC
Le mécanisme de conversion utilise un compteur, identifié dans l'organigramme par `cpt`, qui est incrémenté à chaque interruption de fin de conversion. Ce compteur permet de distinguer les différentes mesures captées par l'ADC. Ainsi, chaque conversion est traitée individuellement dès sa complétion, grâce à la configuration de l'ADC en mode SCAN. Ce mode est particulièrement adapté pour déclencher une interruption après chaque conversion individuelle.
Ordonnancement des conversions
Les conversions sont ordonnées grâce à l'utilisation d'une variable 'rank', qui définit l'ordre de priorité des conversions. Dans notre configuration, la première conversion concerne le capteur 3, la seconde le capteur 4 et la troisième la surveillance de la batterie. Cette organisation permet de structurer efficacement le traitement des données recueillies.
Surveillance de la batterie avec l'analog watchdog
Pour simplifier la surveillance de la batterie, nous avons opté pour l'utilisation d'un analog watchdog. Ce choix permet d'éviter l'écriture de codes extensifs tout en surveillant efficacement les seuils de tension. Avec une tension de référence fixée à 3V et un seuil de 3.3V, les seuils haut et bas de l'analog watchdog sont déterminés en se basant sur une résolution de 12 bits de l'ADC.

$$\begin{aligned} 3.3V &\rightarrow 2^{12} - 1 = 4095 \\ 3V &\rightarrow n \\ n &= \frac{3.4095}{3,3} \end{aligned}$$

Les seuils High et Low de l'*analog watchdog* valent donc 3095 et 3723.

3.1.2 Configuration de la PWM

Le robot est équipé de deux moteurs à chenilles. Selon le cahier des charges, ces moteurs doivent opérer à une fréquence de 1 kHz. Pour atteindre cette spécification, il est nécessaire de déterminer les paramètres optimaux pour le prescaler et l'auto-reload du système de contrôle des moteurs.
Calcul du Prescaler et de l'Auto Reload
Pour configurer correctement les moteurs à la fréquence désirée, nous utilisons la formule suivante afin de calculer les valeurs optimums pour le prescaler et l'auto-reload :

$$\mathrm{PSC} \cdot \mathrm{ARR} \cdot \mathrm{T}_{CLK} = \mathrm{T}_{PWM}$$

où PSC est le prescaler et ARR est la valeur d'auto-reload.
Cette méthode assure que les moteurs fonctionnent de manière efficace et conforme aux exigences du cahier des charges.
La formule appliquée permet de définir les paramètres de façon à ce que la fréquence de sortie corresponde précisément à 1 kHz. Le choix de ces paramètres influence directement la perfor-

mance et la précision du contrôle moteur.

En donnant à ARR une valeur maximale de 2^{32}, nous obtenons :

$$\text{PSC} = \frac{T_{PWM}}{\text{ARR} \cdot \text{TCLK}}$$

En prenant $T_{PWM} = 80 \times 10^6$ et $\text{T}_{CLK} = 10^3$, nous trouvons :

$$\text{PSC} = \frac{80 \times 10^6}{2^{32} \cdot 10^3} \approx 0.0186 \times 10^{-5}$$

Ce qui donne $\text{PSC} = 1$.
Pour le calcul de ARR :

$$\text{ARR} = \frac{T_{PWM}}{\text{PSC} \cdot \text{TCLK}}$$

et en utilisant $T_{PWM} = 8 \times 10^6$ et $\text{PSC} = 1$ et $\text{T}_{CLK} = 10^3$:

$$\text{ARR} = \frac{8 \times 10^6}{10^3} = 80000$$

Puisque le moteur gauche est sur le Channel 1 du timer 2 et le moteur droit sur le Channel 4 du timer 2 on active les deux Channels en mode PWM tel que représenté dans la figure suivante :

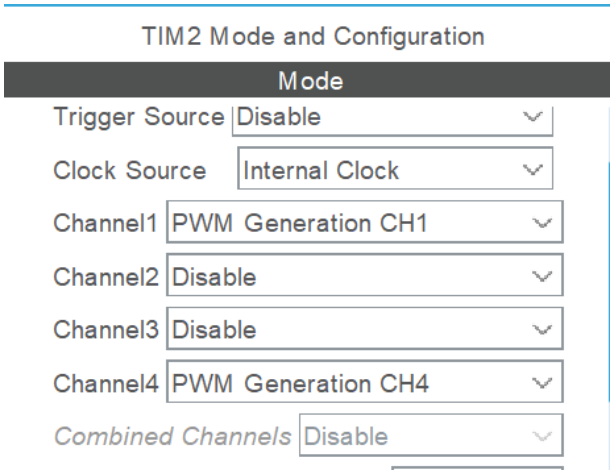

FIGURE 7 – Configuration du timer2

3.1.3 Configuration des LEDs émettrices et du bouton start

Gestion du bouton de démarrage
Le robot est conçu pour démarrer ou s'arrêter en réponse à des interactions avec le bouton bleu situé sur la carte Nucleo. Pour répondre à cette exigence, le bouton doit être configuré en mode interruption. Cette configuration permet au robot de réagir immédiatement à l'activation du bouton, facilitant ainsi un contrôle efficace et réactif de son fonctionnement. Lorsqu'une

pression est détectée, le robot démarre ses opérations et une seconde pression arrête toutes les activités, garantissant une manipulation intuitive et sécurisée.

Configuration des LEDs émettrices

Les LEDs émettrices du robot, utilisées pour la navigation et d'autres fonctionnalités, doivent émettre de la lumière de manière continue pendant le fonctionnement du robot. Afin de garantir une activation sans délai des LEDs, celles-ci sont configurées en mode GPIO output. Cette configuration est essentielle pour permettre un allumage immédiat des LEDs dès que le robot est activé. En maintenant les LEDs allumées tout au long du fonctionnement, nous nous assurons que le robot bénéficie d'une visibilité optimale sans interruption, ce qui est vital pour ses opérations de navigation et d'interaction avec l'environnement.

3.2 Développement du code

3.2.1 Les Callback du projet

Dans notre projet, quatre callbacks principaux ont été implémentés pour répondre à diverses fonctionnalités :

- **Callback de l'analog watchdog :** Ce callback est activé lorsque la tension de la batterie descend en dessous d'un seuil prédéfini. Il a pour rôle d'allumer la LED LD2 (configurée en sortie GPIO), servant ainsi d'indicateur visuel pour prévenir l'utilisateur que la batterie est faible.
- **Callback du timer :** Déclenché toutes les 10 millisecondes, ce callback est essentiel pour initier les conversions ADC à intervalles réguliers, garantissant ainsi une surveillance continue des paramètres surveillés.
- **Callback lié au bouton start :** Ce callback est exécuté à chaque pression sur le bouton bleu. Il modifie l'état de la variable `start` en utilisant un XOR avec 1, permettant ainsi de basculer entre l'état actif et inactif du robot.
- **Callback de fin de conversion ADC :** Il est appelé après chaque conversion complète par l'ADC, et s'occupe de plusieurs tâches critiques : il incrémente le compteur `cpt`, stocke la valeur de l'ADC en fonction de ce compteur, et réinitialise le compteur après trois conversions pour préparer le cycle suivant.

Ces callbacks jouent un rôle vital dans la régulation du comportement du robot, en assurant que les données nécessaires sont correctement traitées et que l'état du système est mis à jour en temps réel.

3.2.2 Implémentation du code principal

Après avoir détaillé les fonctions de chaque callback, nous aborderons le cœur de l'implémentation dans le code principal de notre projet. La figure ci-dessous illustre la structure principale du programme :

```
HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_SET);
if(start){
    HAL_TIM_Base_Start_IT(&htim6);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 23000);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 23000);
    if (ADC_CAP4<SEUIL_CAP4 && ADC_CAP3<SEUIL_CAP3){
        while(1){
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
            if(ADC_CAP4>SEUIL_CAP4 || ADC_CAP3>SEUIL_CAP3) break;
          }
      }
    else if(ADC_CAP3<SEUIL_CAP3){
        HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin, GPIO_PIN_RESET);
    }
    else if(ADC_CAP4<SEUIL_CAP4){
        HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin, GPIO_PIN_SET);
    }
    else if(ADC_CAP4>SEUIL_CAP4 && ADC_CAP3>SEUIL_CAP3){
        HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin, GPIO_PIN_RESET);
    }
}
else{
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
}
```

FIGURE 8 – Partie principale du main

Une procédure de test spécifique est définie pour évaluer le comportement des moteurs du robot. Initialement, les LEDs émettrices sont activées, suivies par le lancement du timer 6. Les moteurs sont ensuite mis en marche avec un rapport cyclique ajusté à 23000, basé sur des mesures expérimentales effectuées avec un chronomètre et un mètre.
La logique de traitement des signaux captés par les capteurs 3 et 4 est mise en œuvre comme suit :

- Lorsque la valeur détectée par le capteur 3(capteur gauche) est inférieure au seuil établi, indiquant la présence de gris, le moteur gauche recule tandis que le moteur droit avance, permettant ainsi au robot de se réaligner correctement avec la ligne.
- Si c'est le capteur 4 (capteur droit) qui détecte une valeur inférieure au seuil, les moteurs ajustent leurs mouvements de façon inverse : le moteur gauche avance et le moteur droit recule.
- Si les deux capteurs détectent le blanc, cela indique que le robot doit continuer à avancer sans ajustement.

— En cas de détection du gris par les deux capteurs, le robot entre dans une phase de correction jusqu'à ce que le blanc soit détecté par l'un des capteurs infrarouges, signalant la fin de la ligne et l'arrêt du robot.

Ces opérations sont effectuées à condition que la variable `start` soit activée (valeur de 1). Si `start` est désactivée (valeur de 0), les moteurs s'arrêtent, plaçant le robot en état de pause.

4 Tests

4.1 Observation du signal PWM

Cette partie du rapport explore l'expérience conduite pour observer et mesurer le signal PWM généré par le microcontrôleur STM32 à l'aide d'un oscilloscope. Notre objectif était de confirmer le rapport cyclique et la fréquence du signal.

4.1.1 Matériel Nécessaire

Les instruments utilisés pour cette expérience incluaient un microcontrôleur STM32, un oscilloscope de Keysight Technologies, une sonde d'oscilloscope, et des câbles de connexion nécessaires pour établir les liaisons électriques adéquates.

4.1.2 Configuration Initiale

Pour commencer, nous avons programmé le microcontrôleur pour qu'il démarre la modulation de largeur d'impulsion (PWM) sur le canal spécifié et réglé le comparateur pour atteindre la valeur désirée :

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 23000);
```

Nous avons ensuite préparé l'oscilloscope en connectant la sonde directement au pin PWM du microcontrôleur et attaché le clip de mise à la terre à un pin de masse (GND). L'oscilloscope a été configuré pour un temps de base de 500.0 µs/div, une tension de 1.00 V/div, et un déclenchement sur le front montant pour capturer clairement le signal PWM.

4.1.3 Observation et Analyse

L'observation directe sur l'oscilloscope a révélé un signal PWM bien défini oscillant entre 0V et la tension maximale du microcontrôleur, avec des variations du rapport cyclique. Les mesures effectuées ont montré que le rapport cyclique était de 28.75%, indiquant que le signal reste à l'état haut pour environ 28.75% du temps total de chaque cycle. De plus, la fréquence du signal a été enregistrée à 501.50 Hz, correspondant au nombre de cycles que le signal complète en une seconde.

4.1.4 Calculs théoriques

Nous avons également effectué des calculs pour comparer les observations avec les valeurs théoriques :

$$\text{Duty Cycle} = \frac{23000}{80000} \times 100 = 28.75\%$$

Les résultats théoriques et expérimentaux concordent parfaitement, validant ainsi l'efficacité de notre configuration et nos méthodes de mesure.

5 Options supplémentaires

5.1 Le filtre anti-rebond

Lors de l'appui sur le bouton start, une interruption du bouton est censée se déclencher. Cependant, plusieurs interruptions successives du bouton peuvent se produire, ceci étant dû aux fronts montants successifs. Cela justifie l'utilisation d'un filtre anti-rebond.
Le filtre anti-rebond permet de ne prendre en compte que l'interruption du premier front montant. Les autres fronts sont ignorés pendant une durée de 500 ms. On suppose que la durée entre deux appuis successifs de l'utilisateur est supérieure à 500 ms. Dans le cas contraire, l'appui suivant ne sera pas pris en compte.

5.1.1 Développement du filtre anti-rebond

Le code ci-dessous montre la fonction de rappel (callback) pour la gestion des interruptions du bouton :

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    //now=HAL_GetTick();
    if(GPIO_Pin == B1_Pin){
        start = start ^ 1;
        HAL_NVIC_DisableIRQ(EXTI15_10_IRQn);
        cnt = 0;
    }
}
```

Lors de l'appui sur le bouton start, cette fonction est déclenchée, ce qui commute l'état de la variable `start` et désactive toutes les autres interruptions pendant 500 ms en utilisant la fonction `HAL_NVIC_DisableIRQ(EXTI15_10_IRQn)`. Le compteur de temps `cnt` est réinitialisé à 0.
Ensuite, le code suivant dans la boucle principale réactive les interruptions après 500 ms :

```
if(cnt > 50){
    HAL_GPIO_EXTI_CLEAR_IT(B1_Pin);
    HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
    HAL_GPIO_EXTI_CLEAR_IT(B1_Pin);
    // start = start ^ 1;
    cnt = 0;
}
```

Dans cette section du code, une fois que `cnt` dépasse 50, les interruptions sont réactivées en utilisant la fonction `HAL_NVIC_EnableIRQ(EXTI15_10_IRQn)`.
La fonction `HAL_GPIO_EXTI_CLEAR_IT(B1_Pin)` est utilisée pour ignorer toute interruption supplémentaire.
En somme, lors d'un appui sur le bouton start, la fonction de rappel du bouton est déclenchée et l'état de la variable `start` est commuté. Toutes les autres interruptions sont désactivées pendant 500 ms à l'aide de la fonction `HAL_NVIC_DisableIRQ(EXTI15_10_IRQn)`. Le timer6

est utilisé pour incrémenter le compteur de temps `cnt`. Après 500 ms, les interruptions sont réactivées avec `HAL_NVIC_EnableIRQ(EXTI15_10_IRQn)`
et `HAL_GPIO_EXTI_CLEAR_IT(B1_Pin)` pour ignorer toute interruption supplémentaire. Ce mécanisme permet de filtrer efficacement les rebonds du bouton.

6 Conclusion

Applications industrielles potentielles du robot suiveur de ligne

L'adoption potentielle des robots suiveurs de ligne dans divers secteurs industriels pourrait entraîner des améliorations significatives en termes d'efficacité et de réduction des coûts.

En logistique, par exemple, l'intégration de ces robots dans les entrepôts pourrait simplifier le transport des marchandises. Ils pourraient naviguer de manière autonome, suivant des lignes au sol pour optimiser les opérations de stockage et de distribution, réduisant potentiellement les besoins en main-d'œuvre.

Dans le secteur manufacturier, l'utilisation envisagée de robots suiveurs de ligne pourrait rendre la production plus flexible. Ils pourraient transporter des composants d'un poste à l'autre, permettant des adaptations rapides aux changements de la demande du marché sans interruptions coûteuses.

L'agriculture pourrait aussi bénéficier de ces technologies. Imaginons des robots effectuant des tâches de surveillance, de pulvérisation de pesticides et de récolte, optimisant ainsi la couverture du terrain et minimisant les gaspillages.

En matière de surveillance et de sécurité, l'utilisation de robots suiveurs de ligne pourrait renforcer la sécurité dans les installations industrielles ou commerciales. Ils pourraient patrouiller selon des itinéraires prédéfinis et être équipés de capteurs pour détecter toute activité suspecte.

Les services de nettoyage dans les lieux publics tels que les aéroports et les hôpitaux pourraient aussi voir une révolution avec l'arrivée de robots suiveurs de ligne. Ceux-ci pourraient suivre des parcours prédéterminés pour assurer un nettoyage systématique et constant, réduisant le besoin en personnel de nettoyage.

Enfin, dans le secteur de la santé, les robots d'assistance pourraient améliorer la logistique interne des hôpitaux en prenant en charge la distribution de médicaments et d'échantillons de laboratoire, permettant au personnel soignant de se concentrer davantage sur les soins aux patients.

7 Annexes

7.1 La fonction main

```
    int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration
   ----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
   Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */
  HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin, GPIO_PIN_RESET);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if(cnt>50){
      __HAL_GPIO_EXTI_CLEAR_IT(B1_Pin);
      HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
      __HAL_GPIO_EXTI_CLEAR_IT(B1_Pin);
     // start=start^1;
      cnt=0;
    }
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, GPIO_PIN_SET);
    if(start){
      HAL_TIM_Base_Start_IT(&htim6);
      HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

```
50        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 23000);
51        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
52        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 23000);
53        if (ADC_CAP4<SEUIL_CAP4 && ADC_CAP3<SEUIL_CAP3){
54            while(1){
55              __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
56              __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
57              if(ADC_CAP4>SEUIL_CAP4 || ADC_CAP3>SEUIL_CAP3) break;
58             }
59          }
60        else if(ADC_CAP3<SEUIL_CAP3){
61          HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_SET);
62          HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin, GPIO_PIN_RESET);
63        }
64        else if(ADC_CAP4<SEUIL_CAP4){
65          HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_RESET);
66          HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin, GPIO_PIN_SET);
67        }
68        else if(ADC_CAP4>SEUIL_CAP4 && ADC_CAP3>SEUIL_CAP3){
69          HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_RESET);
70          HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin, GPIO_PIN_RESET);
71        }
72      }
73      else{
74        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
75        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
76      }
77      /* USER CODE END WHILE */
78
79      /* USER CODE BEGIN 3 */
80    }
81    /* USER CODE END 3 */
82  }
```

7.2 Les callbacks

```
    /* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
  //now=HAL_GetTick();
  if(GPIO_Pin==B1_Pin){
    start=start^1;
    HAL_NVIC_DisableIRQ(EXTI15_10_IRQn);
        cnt=0;
  }
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  if(htim->Instance==TIM6){
    HAL_ADC_Start_IT(&hadc1);
    cnt++;
  }

}
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc){
  if(hadc->Instance==ADC1){
    cpt++;
    if(cpt==1){
      ADC_CAP3=HAL_ADC_GetValue(&hadc1);
    }
    if(cpt==2){
      ADC_CAP4=HAL_ADC_GetValue(&hadc1);
    }
    if(cpt>=3){
      NUM_BAT=HAL_ADC_GetValue(&hadc1);
      cpt=0;
    }
  }
}
```

Table des figures