



GRENOBLE-INP
PHELMA

Rapport : Projet Contrôle-commande temps réel (CCTR)

Oussama LAMZAOURI
Youssef BOUSSEMID
Yasser KHARJ

3e année SEOCC

2022/2023

Table des matières

1 Introduction	3
2 Prise en main	4
3 Moteurs de la voiture	5
3.1 Principe de fonctionnement	5
3.2 Contrôle de vitesse	5
3.3 Implémentation logicielle	5
4 Contrôle de direction	6
4.1 Théorie du différentiel géométrique :	7
4.2 Implémentation logicielle	7
4.3 Asservissement d'angle :	8
5 Pilotage	8
5.1 Par caméra	8
5.1.1 Le CAN et réglage caméra utilisé	10
5.1.2 Sortie en terminal et résultats	12
5.1.3 Implémentation logicielle du filtre gaussien	12
5.1.4 Résultat de filtrage et interprétation graphique de la courbe d'intensité	13
5.1.5 Implémentation logicielle de la tache "task_Camera"	14
6 Priorités des tâches	14
7 Modélisation sous forme de représentation d'état	15
8 Conclusion	16
9 Code source pour main_LED.c	19
10 Code source pour Task_Steering.c	20
11 Code source pour Task_Camera.c	22
12 Code source pour main_SEOC_CCTR_base.c	24

Glossaire

CAN convertisseur analogique-numérique.. [10](#)

CNA convertisseur numérique-analogique.. [10](#)

servo En programmation de microcontrôleurs, un servomoteur est un moteur électrique qui peut être commandé pour tourner à des angles précis. . [3](#)

temps d'exposition désigne la durée pendant laquelle la lumière est autorisée à entrer dans le boîtier d'un capteur caméra.. [10](#)

1 Introduction

Ce projet a pour but de mettre en pratique les notions théoriques acquises durant le cours de "Réel Contrôle-commande temps réel des systèmes Cyber-Physiques" afin de réaliser un algorithme qui permet à la Voiture (NXP modèle Alamak) de parcourir d'une façon autonome un circuit sans diverger et ceux grâce un asservissement en temps réel.

Dans un premier temps, nous nous sommes familiarisés avec l'environnement temps réel FreeRTOS. Ensuite, nous avons connu les différentes composantes de la voiture, notamment :

- Les deux moteurs qui actionnent les roues arrière.
- La caméra qui aide à connaître la position de la voiture dans la voie.
- Les capteurs de roues jouent un rôle très important, notamment dans le calcul de la vitesse de chaque roue. Ils permettent de savoir si la voiture a atteint la vitesse maximale désignée et sont particulièrement utiles lors de la rotation, qui sera mieux détaillée dans la partie "Contrôle de direction".
- L'alimentation alimente toutes les composantes précédentes pour assurer leur fonctionnement correct.
- Les interrupteurs : ils sont utilisés pour l'unité de microcontrôleur, pour l'alimentation des deux moteurs et enfin pour l'alimentation du **servo**.
- Le microcontrôleur NXP FRDM-KL25Z, qui agit comme le cerveau de la voiture autonome en exécutant les consignes du programme et en récupérant les données fournies par les capteurs et les caméras. Il assure un bon asservissement en veillant à ce que la voiture ne sorte pas de sa voie.

Ainsi, nous avons programmé chacune des parties qui contribuent dans l'asservissement du mouvement de la voiture pour arriver aux résultats souhaités.

2 Prise en main

Dans la partie théorique, nous avons comparé dans un premier lieu les OS temps réel avec les OS temps partagé dans le contexte de gestion d'exécution des processus pour mettre en évidence leurs points forts dans ce projet. Ainsi, nous avons compris les fonctionnalités avancées pour la gestion des tâches qu'offre FreeRTOS, telles que la gestion de la priorité et la synchronisation entre tâches. Ensuite, nous avons compris pourquoi le facteur de stabilité des périodes est important à prendre en compte pour garantir la performance en temps réel.

Dans la partie pratique, nous avons étudié comment ajouter des tâches à notre programme sur la carte FRDM-KL 46 en leur attribuant une priorité. Cette notion est cruciale dans un système d'exploitation temps réel, car elle permet de déterminer l'ordre d'exécution des tâches lorsque plusieurs tâches peuvent être exécutées simultanément. Pour mettre en pratique cette compétence, nous avons créé une tâche appelée "task_LED" Annexe 9 qui a pour objectif de faire clignoter une LED RGB toutes les 25 ms avec différentes couleurs. Nous avons donc codé cette tâche dans le fichier "task_LED.c" Annexe 9 et l'avons appelée dans le fichier "main_SEOC_CCTR_base.c" Annexe 12 en utilisant une constante "task_LED_PRIORITY" Annexe 12 pour définir sa priorité.

3 Moteurs de la voiture

3.1 Principe de fonctionnement

Les moteurs de notre voiture électrique sont alimentés à l'aide d'un pont H ; Le fonctionnement de ce dernier est relativement simple. Il utilise deux transistors pour contrôler le courant qui alimente le moteur. Lorsque le courant est envoyé à travers le moteur dans un sens, il fait tourner le moteur dans une direction, tandis que lorsque le courant est inversé, il fait tourner le moteur dans l'autre direction.

Le pont en H est commandé par un signal binaire provenant d'un "Timer" de fréquence élevée. Ainsi, pour contrôler la puissance du signal PWM envoyé au moteur, il suffit de régler le rapport cyclique et la fréquence du signal.

3.2 Contrôle de vitesse

Pour maintenir une vitesse cible, un capteur mesure la vitesse actuelle de la voiture en comptant le nombre de passages d'aimants sur une roue pendant un intervalle de temps donné.

Cette mesure de vitesse est utilisée pour ajuster le rapport cyclique de la voiture de manière à atteindre la vitesse cible. Cet ajustement est effectué en utilisant une boucle de contrôle appelée "boucle d'asservissement" qui modifie les entrées du système en fonction de la différence entre la vitesse cible et la vitesse actuelle.

Le contrôle optimal serait trop complexe à mettre en place, alors une commande "intégrale" est utilisée à la place. La formule de la commande intégrale est la suivante :

$$PWM_{k+1} = PWM_k - K(\omega - \omega_{ref})$$

Avec :

- ω : vitesse angulaire mesurée
- ω_{ref} : vitesse angulaire cible
- K : gain de la boucle de contrôle
- PWM : rapport cyclique de la commande du moteur

3.3 Implémentation logicielle

```

1 //Le facteur K est determine de maniere experimentale
2 #define K 0.01
3 void task_Speed(void *pvParameters) {
4     ...
5     //Acquisition des vitesses calculs par le capteur effet Hall
6     int sGn=speed_2_cnt, sDn=speed_1_cnt;
7     speed_2_cnt =0;
8     speed_1_cnt =0;
9

```

```

10 //La formule de la commande intégrale pour asservir la vitesse des roues sur
11 //une vitesse cible
12 pwm_duty_cycle_d = pwm_duty_cycle_d - K*((float)sDn - speed_target);
13 pwm_duty_cycle_g = pwm_duty_cycle_g - K*((float)sGn - speed_target);
14
15 //Commande sur les moteurs de roues
16 Motor_UpdatePwm((uint16_t)pwm_duty_cycle_g, -1 ,(uint16_t)pwm_duty_cycle_d,
17 -1);
18 ...
19 }
```

4 Contrôle de direction

Après avoir réussi les mouvements avant et arrière, il est crucial de connaître l'angle de rotation optimal pour les roues afin que la voiture maintienne sa position au milieu de la voie, en particulier dans les virages.

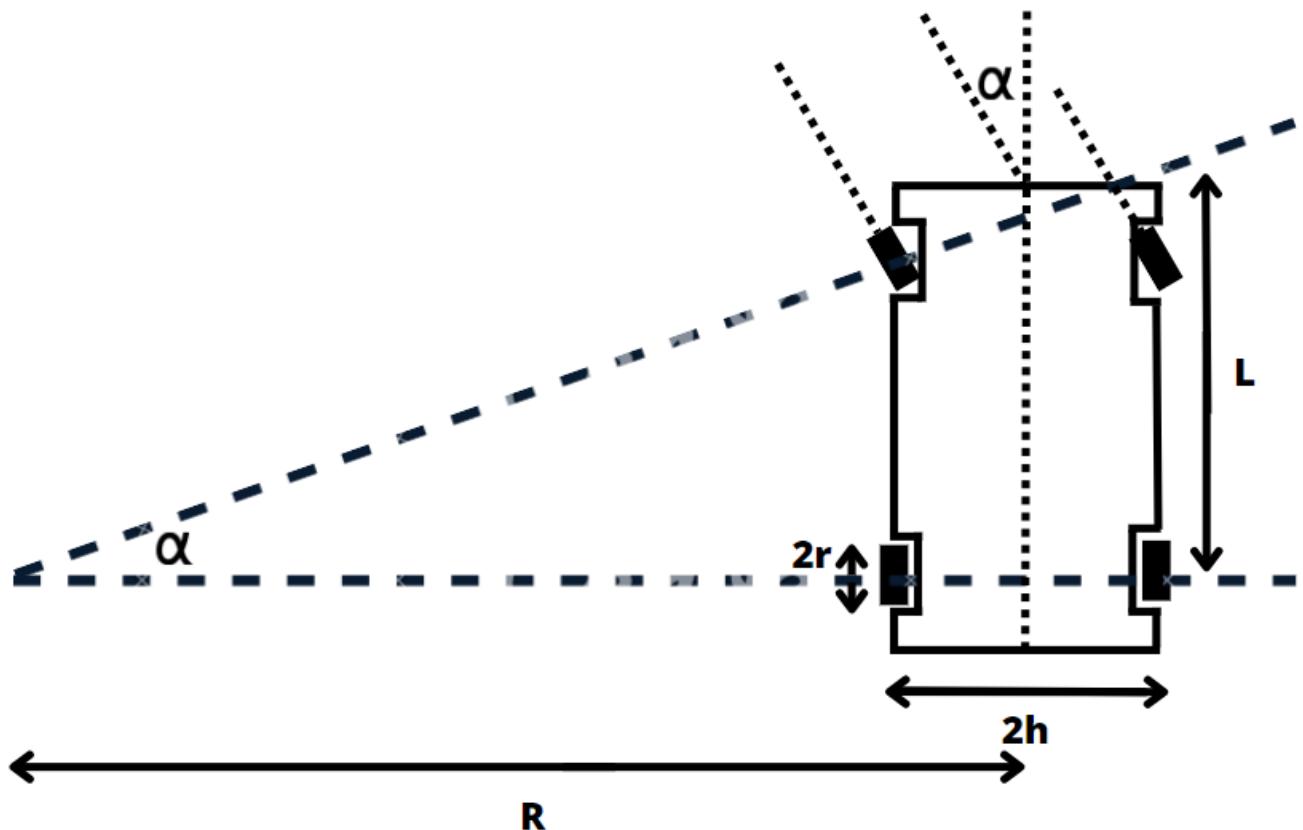


FIGURE 1 – Schéma représentant la rotation de la voiture.

4.1 Théorie du différentiel géométrique :

Compte tenu du fait que les roues gauches et droites ne vont pas à la même vitesse lors des virages à angle α , il est fondamental de calculer le rapport de vitesse pour assurer une circulation stable de la voiture sans déviation. Pour garantir un asservissement optimal de la voiture, il est crucial de calculer le rapport des vitesses angulaires des roues lors de la planification de la trajectoire de la voiture.

Théoriquement, on a :

$$\tan \alpha = L/R$$

Avec :

- α : Angle de rotation
- L : Longueur de la voiture
- R : Rayon de courbure

$$D_g = \alpha(R - h)$$

Avec :

- D_g : Distance entre les roues gauches
- $2 * h$: Largeur de la voiture

$$D_d = \alpha(R + h)$$

Avec :

- D_d : distance entre les roues droites.

Alors le rapport de vitesse linéaire :

$$V_g/V_d = (R - h)/(R + h)$$

Avec :

- V_g : Vitesse linéaire des roues gauches.
- V_d : Vitesse linéaire des roues droites.

le rapport de vitesse angulaire :

$$\omega_g/\omega_d = (R - h)/(R + h)$$

Avec :

- ω_g : Vitesse angulaire des roues gauches.
- ω_d : Vitesse angulaire des roues droites.

4.2 Implémentation logicielle

Pour asservir notre voiture lors d'un virage, il est nécessaire tout d'abord de calculer l'angle de rotation de la voiture. Pour ce faire ; il fallait tout d'abord connaître la position de notre

voiture par rapport au centre de la piste. Cette partie est faite dans la partie du pilotage par caméra. Le centre de la piste es calculé au milieu des ectrémités de la piste et le la position de la voiture correspond au bit 64 de la camera (milieu de la plage de bits de la caméra). En contrôllant le steering de la voiture, ona choisit de travailler sur une plage de [-50,50] ce qui correspond à une plage de 100 ; Ceci correspond à une plage de bits de 128 bits. Par relation de trois :

$$angleRot = \cdot \left(\frac{Center - 64}{128} \right) \cdot 100$$

```

1
2 void task_SteeringRoute(void *pvParameters) {
3     const TickType_t xDelay= Ms(5);
4     int ksmall = 2;
5     int kextreme = 4;
6     int k;
7     for (;;)
8     {
9         //Center (centre de la piste) est calcul dans la tache de la camera
10        if(Center <74 && Center>54)k =ksmall;
11        else k=kextreme;
12        //calcul de l'angle de rotation
13        int angleRot = -kextreme*(Center-64)*100/128;
14        //Controle de l'angle de rotation en multipliant par un coefficient
15        //determine par mesure pratique
16        Steering_UpdatePwm_Relative(angleRot);
17        vTaskDelay(xDelay);
18    }

```

4.3 Asservissement d'angle :

La mise à jour brusque faite sur l'angle, nous a causé des rotation aigu en fin de virages, ceci pourra être résolu en ajoutant un filtre de Kalman comme ce qu'on a fait dans la partie de controle de la vitesse des roues. Malheureusement, en essayant d'implémenter ce filtre, cela nous a pris beaucoup de temps et on a pas réussit à avoir les résultats voulu.

5 Pilotage

5.1 Par caméra

Après avoir validé toutes les étapes mentionnées ci-dessus pour le projet, il est temps de passer à l'étape d'auto-pilotage de la voiture par un capteur caméra.



FIGURE 2 – Installation matérielle du capteur caméra

Maintenant, on passe au calcul et à la manipulation du capteur afin de pouvoir rendre la voiture autonome sur la piste.

La caméra utilisée est une caméra ligne analogique donc elle permet l'acquisition d'une ligne de 128 pixels sur laquelle on va se baser pour calculer la position de la voiture sur l'axe de largeur de la piste afin de pouvoir asservir sur cette dernière et permettre à la voiture de réagir dans les virages ou autre changement de piste .

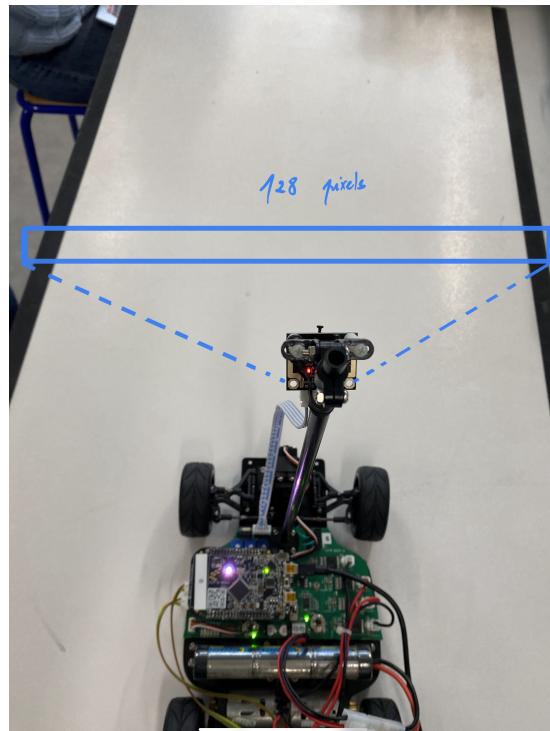


FIGURE 3 – Installation matérielle du capteur caméra

La fonction `Camera_ImageCapture()` permet de récupérer les pixels captés et les stocke dans un buffer passé en paramètre, on l'utilise au sein de la tâche `task_camera` dans le fichier source `task_camera.c`.

5.1.1 Le CAN et réglage caméra utilisé

La conversion analogique/numérique est réalisée dans le micro-contrôleur, par un **CAN**.

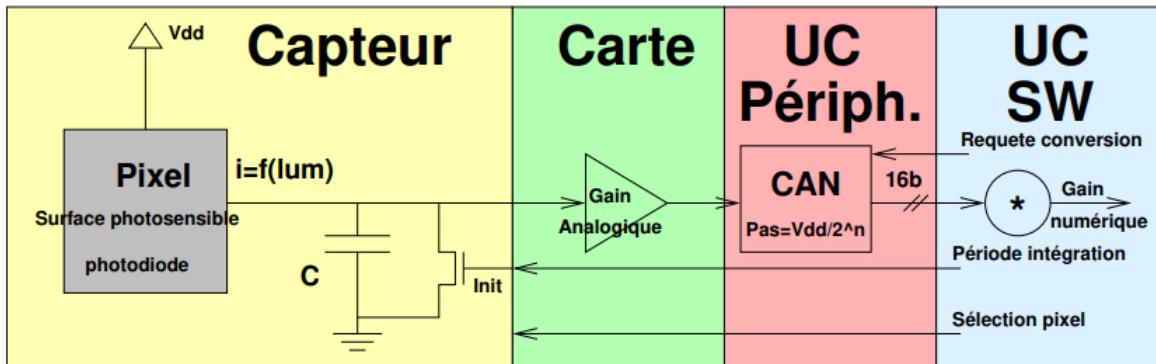


FIGURE 4 – Le convertisseur Analogique numérique utilisé dans le micro-contrôleur.

Dans le micro-contrôleur, le **CNA** convertit une tension en un pas de la tension d'alimentation. La résolution peut être choisie (8, 10 ou 12 bits), dans notre cas, on choisit 10 bits, ce qui implique une valeur maximale d'intensité des pixels à 1024.

On s'engage à respecter les contraintes suivantes dans notre projet comme demandé :

- Le réglage des gains et **temps d'exposition** dépend des conditions lumineuses
- La période d'intégration doit être stable, car elle agit comme un gain

En l'occurrence, on choisira un temps d'exposition égal à 5 ms, car après plusieurs essais cela semble fonctionner le mieux pour notre capteur caméra en termes de détection piste. Pour s'y faire, on collecte la luminosité pendant 50 ms par le capteur caméra, mais les données sont stockées dans un buffer (`nullBuffer`) qui sert à se débarrasser de ces données. Ensuite, on collecte pendant 5 ms pour stocker les données dans un buffer (`buffer`) utilisé par la suite pour les différents.

Les constantes `xDelay` et `xDelayMain` sont définies pour représenter respectivement un délai de 50 ms et 5 ms. Les variables `xLastWakeTime` et `xLastWakeTime1` sont utilisées pour conserver le temps dernier de mise en veille de la tâche.

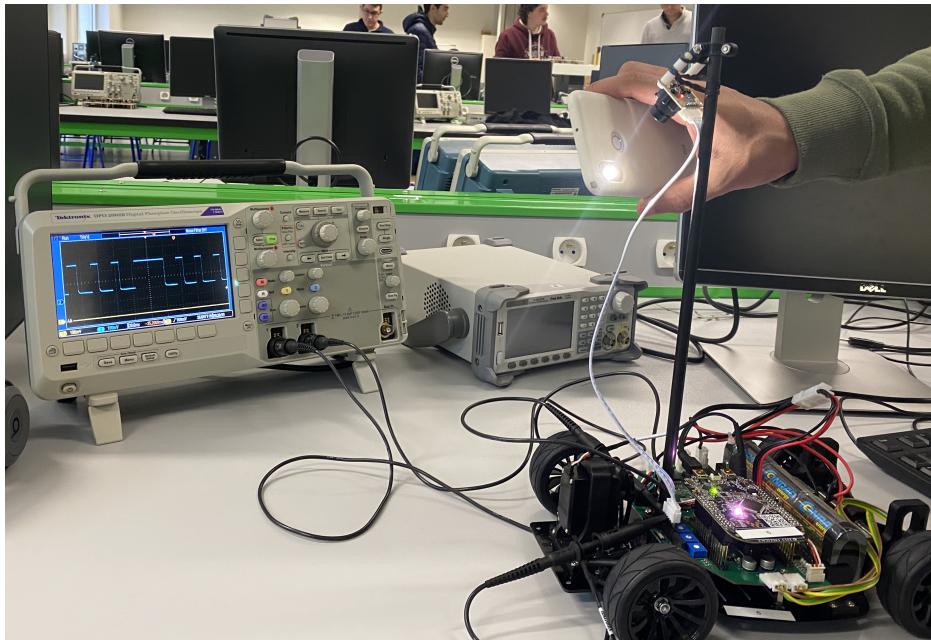


FIGURE 5 – Test 1 du temps d'exposition de la caméra avec oscilloscope

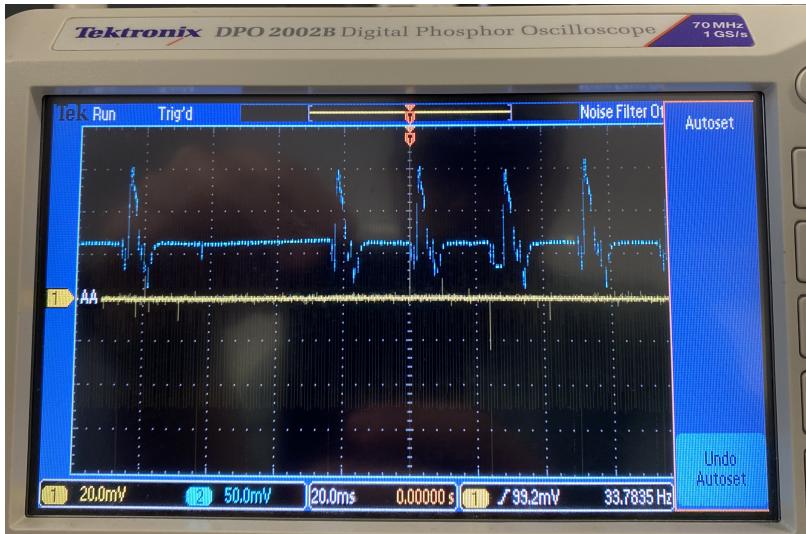


FIGURE 6 – Test 2 du temps d'exposition de la caméra avec oscilloscope

Après avoir branché les pôles de la caméra avec l'oscilloscope, on peut visualiser la `data_out` de notre capteur caméra. On met notre capteur caméra face à un flash pour atteindre la saturation en terme d'intensité des pixels. On remarque un graphique cohérent avec notre implémentation logicielle. Une large collection de donnée de 50 ms correspondante à la première collecte qu'on stocke sur `nullBuffer` puis plusieurs collectes de 5 ms comme implanté.

Pour voir toute l'implémentation logicielle de cette tâche, voir annexe 11.

5.1.2 Sortie en terminal et résultats

Ci-dessous, la sortie en terminal après lancement d'un script python "CCTR_Display_Camera.py", permettant l'affichage des 128 pixels récupéré depuis le capteur caméra et la fonction `task_camera_display` sur `task_camera.c`.

```
camera 40 33 44 38 45 37 41 41 43 36 43 32 39 35 38 33 39 37 40 36 36 34 41 35 4
3 38 44 36 47 43 56 58 77 82 98 103 118 125 149 155 178 184 204 215 228 231 244
238 254 246 252 246 254 247 250 245 247 246 251 234 248 240 246 241 248 240 241
232 242 228 236 221 218 209 204 191 187 174 170 154 150 136 132 119 118 107 105
97 97 87 89 76 81 72 68 54 57 49 48 44 45 39 41 36 41 38 42 38 40 40 46 41 50 45
48 42 47 42 47 40 48 33 39 33 34 27 30 52
```

FIGURE 7 – Contenu du buffer de 128 pixels

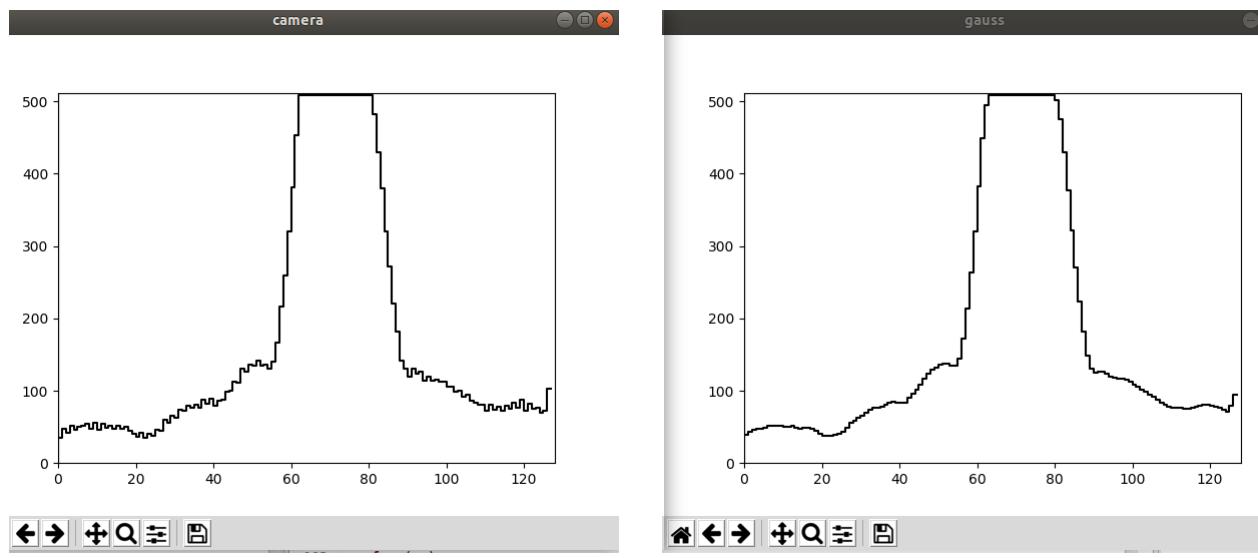


FIGURE 8 – Intensité des pixels récupérés depuis le capteur caméra

On remarque clairement l'adoucissement des bords de la courbe après filtrage gaussien, ce qui va nous permettre d'avoir une distribution gaussienne d'intensité de pixel à traiter.

5.1.3 Implémentation logicielle du filtre gaussien

Pour filtrer notre courbe d'intensité de pixel, on se base sur un filtre gaussien simple à trois valeurs.

```
1 float gauss[3] = {0.25, 0.5, 0.25};
```

Pour le filtrage, on a besoin de faire une convolution entre notre filtre gaussien et notre courbe. Pour s'y faire, on utilise une fonction `filtrage()` qui effectue cette opération avec une fenêtre de convolution de largeur 3 et `gradient()` qui effectue un calcul de gradient avec une fenêtre de largeur 5.

```

1 void filtrage (uint16_t* buffer){
2
3     for(int i=0;i<BUFFER_SIZE;i++){
4         gaussBuffer[i] = (uint16_t)((buffer[MAX(i-1,0)]*gauss[0]) +
5             (buffer[i]*gauss[1]) +
6             (buffer[MIN(i+1,BUFFER_SIZE -1)]*gauss[2]));
7     }
8 }
9
10 void gradient(){
11     for(int i=0;i<BUFFER_SIZE;i++){
12         gradBuffer[i] = (int16_t)((gaussBuffer[MIN(i+2,BUFFER_SIZE -1)] -
13             gaussBuffer[MAX(i-2,0)]));
14     }
15 }
```

Ces deux traitement, filtrage et gradient nous permettent de garder la courbe affichée en temps réel stable, car on élimine toute fluctuation brusque.

5.1.4 Résultat de filtrage et interprétation graphique de la courbe d'intensité

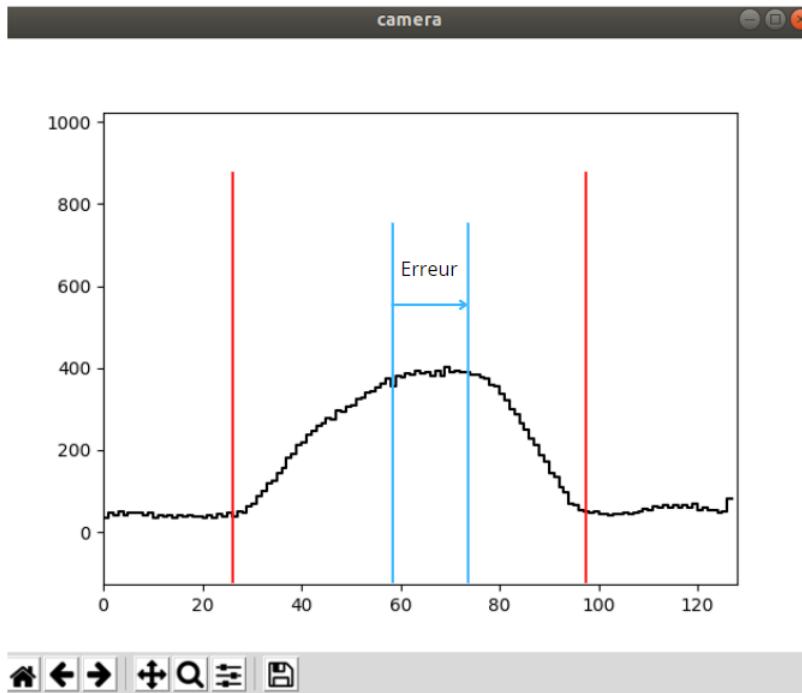


FIGURE 9 – Courbe finale d'intensité des 128 pixels récupérés

D'après la courbe ci-dessus, on peut déterminer graphique les bords de notre piste (les extrémités) ainsi que la piste au milieu. Mais aussi son centre pour le comparer avec centre réel de la piste qui est le pixel 64, la différence entre les deux permet le calcul de l'erreur.

Par cela et par asservissement de la direction et la vitesse, on sera capable de maintenir la voiture au milieu de la piste comme voulu.

L'algorithme sur lequel on se base ainsi que les structures de données sont fournies dans le code source en annexe [11](#).

5.1.5 Implémentation logicielle de la tache "task_Camera"

Le code complet de cette implémentation est fourni en annexe [11](#).

6 Priorités des tâches

La gestion des priorités des tâches en temps réel est cruciale afin de garantir le bon fonctionnement de notre système ; Il faut garantir que les tâches les plus importantes sont exécutées en premier et que les délais critiques sont respectés.

```

1 void task_LED(void *pvParameters);
2 #define task_LED_PRIORITY 0
3
4 void task_Speed(void *pvParameters);
5 #define task_SPEED_PRIORITY 1
6
7 void task_Camera_Display(void *pvParameters);
8 #define task_CAMERA_DISPLAY_PRIORITY 2
9
10 void task_Camera(void *pvParameters);
11 #define task_CAMERA_PRIORITY 3
12
13 void task_Motor(void *pvParameters);
14 #define task_MOTOR_PRIORITY 2
15
16 void task_Steering(void *pvParameters);
17 void task_SteeringRoute(void *pvParameters);
18 #define task_STEERING_PRIORITY 2

```

Pour ce projet, les priorités ne sont pas gérées comme dans le cours avec un ordre descendant (la tache avec la priorité la plus faible est la plus prioritaire) mais dans un ordre ascendant (la tache avec la priorité la plus grande est la plus prioritaire).

- **task_speed** : Pour la tache speed, elle contient task_motor.c aussi. Elle sert à la gestion des vitesses des moteurs et leur aussi. Donc, elle est la moins prioritaire, car elle se base sur les autres pour autres taches, fonctionnement correct.
- **task_Camera_Display** : Elle sert à l'affichage des données récupérées depuis le capteur caméra dans le terminal par l'intermédiaire du script python dédié. Donc moins prioritaire que **task_camera** car elle affiche les données que cette dernière retourne.
- **task_Camera** : La plus prioritaire entre les taches, sert aux différents traitements d'image récupérée depuis le capteur caméra (filtrage gaussien, gradient et double intégration) ayant comme but la détection de piste et ses bords afin de maintenir la voiture sur son milieu.

- `task_SteeringRoute` : sert au contrôle et l'asservissement de la direction de la voiture, donc une priorité inférieure à la tâche caméra et égale aux autres taches.

7 Modélisation sous forme de représentation d'état

En effectuant le même traitement que nous utilisons habituellement dans le cours, nous allons extraire le vecteur d'état afin de déterminer les équations de notre système.

Afin de déterminer notre vecteur d'état, nous avons besoin de connaître les composantes que nous souhaitons contrôler, à savoir l'angle de rotation de la voiture et les vitesses de rotation des roues gauche et droite.

Par conséquent, notre vecteur d'état sera comme suit :

$$X = \begin{bmatrix} \alpha \\ W_d \\ W_g \end{bmatrix}$$

Le vecteur consigne comporte à son tour l'orientation des roues avant et les tensions alimentant les moteurs des roues arrières droite et gauche. Les tensions sont associées aux rapports cycliques dans la voiture réelle.

$$U = \begin{bmatrix} \delta \\ kp_d \\ kp_g \end{bmatrix}$$

Avec :

- δ : angle des servomoteurs.
- kp_d : rapport cyclique des roues droites.
- kp_g : apport cyclique des roues gauches.

Après avoir identifié les équations décrivant de manière appropriée le système que nous étudions, nous allons effectuer sa linearisation en utilisant le système d'équations suivant :

$$\begin{cases} \dot{X} = AX + BU \\ Y = CX + D \end{cases}$$

Avec A, B, C, D des matrices qui doivent être calculées pour obtenir le résultat final.

8 Conclusion

Ce projet visait à mettre en pratique les connaissances acquises en cours de "Contrôle-commande temps réel" pour développer un algorithme qui permettrait à la voiture NXP de parcourir de manière autonome un circuit défini sans dévier de sa voie.

Grâce à notre familiarisation avec l'environnement temps réel FreeRTOS et à notre compréhension des différentes composantes de la voiture, nous avons réussi à atteindre les résultats souhaités. Bien que nous ayons rencontré des défis tels que l'asservissement de la rotation, notre détermination et nos connaissances acquises en cours nous ont permis de les surmonter.

Ce projet a été une excellente occasion pour mettre en pratique nos connaissances théoriques et pour nous préparer pour notre parcours professionnel.

Références

- [1] Support de cours (CCTR) sur Chamilo.
- [2] Support du projet sur Chamilo.

Annexes

9 Code source pour main_LED.c

```
1 #include "CCTR.h"
2 void task_LED_2(void *pvParameters) {
3     const TickType_t xDelay= Ms(1000);
4     int led_cpt=0;
5     for (;;) {
6         //PRINTF("LED blink %d\r\n", led_cpt);
7         LED_RED_TOGGLE();
8         LED_GREEN_TOGGLE();
9         LED_BLUE_TOGGLE();
10        //PRINTF("A");
11        vTaskDelay(xDelay);
12        led_cpt++;
13        //vTaskSuspend(NULL);
14    }
15 }
```

10 Code source pour Task_Steering.c

```

1
2 #include "CCTR.h"
3 #include <math.h>
4
5 static uint16_t buffer[128];
6 static uint16_t nullBuffer[128];
7 static uint16_t gaussBuffer[128];
8 static int16_t gradBuffer[128];
9 unsigned char indiceGauche;
10 unsigned char indiceDroite;
11 unsigned char Center;
12 /* test CCD Camera */
13 ///////////
14 #define sigma 1
15 #define K 1
16 #define BUFFER_SIZE 128
17 ///////////
18
19 float gauss[3] = {0.25, 0.5, 0.25};
20
21 void filtrage (uint16_t* buffer){
22
23     for(int i=0;i<BUFFER_SIZE;i++){
24         gaussBuffer[i] = (uint16_t)((buffer[MAX(i-1,0)]*gauss[0]) +
25             (buffer[i]*gauss[1]) +
26             (buffer[MIN(i+1,BUFFER_SIZE -1)]*gauss[2]));
27     }
28 }
29
30 void gradient(){
31     for(int i=0;i<BUFFER_SIZE;i++){
32         gradBuffer[i] = (int16_t)((gaussBuffer[MIN(i+2,BUFFER_SIZE -1)] -
33             gaussBuffer[MAX(i-2,0)]) );
34     }
35 }
36
37 void task_Camera(void *pvParameters) {
38     /*const TickType_t xDelay= Ms(50);*/
39     const TickType_t xDelay= Ms(50);
40     const TickType_t xDelayMain= Ms(5);
41     TickType_t xLastWakeTime;
42     TickType_t xLastWakeTime1;
43
44     PRINTF("Task Camera GO\r\n");
45     PRINTF("\r\n");
46     xLastWakeTime = xTaskGetTickCount();
47     xLastWakeTime1 = xLastWakeTime;
48     for (;;)
49     {
50         //xLastWakeTime = xTaskGetTickCount();
51         Camera_ImageCapture(1, nullBuffer);
52         //for(int ii=0; ii<128; ii++) buffer[ii]= 0.5*buffer[ii];

```

```

53     //vTaskDelay(xDelay);
54     vTaskDelayUntil( &xLastWakeTime, xDelay );
55     //xLastWakeTime = xTaskGetTickCount();
56     Camera_ImageCapture(1, buffer);
57     //vTaskDelayUntil( &xLastWakeTime, xDelayMain );
58     for(int ii=0; ii<128; ii++) buffer[ii]= 0.5*buffer[ii];
59     filtrage(buffer);
60     gradient();
61     int maxGauche = 0;//buffer[0];
62     int minDroite = 0;//buffer[65];
63     indiceGauche=0;
64     indiceDroite=0;
65     for(int i=1;i<64;i++){
66         if(gradBuffer[i]>maxGauche && gradBuffer[i]>60){
67             maxGauche = gradBuffer[i];
68             indiceGauche = i;
69         }
70         if(gradBuffer[i+64]<minDroite && gradBuffer[i+64]<-60){
71             minDroite = gradBuffer[i+64];
72             indiceDroite = i+64;
73         }
74     }
75     Center = (indiceDroite+indiceGauche)/2;
76     //gradBuffer[indiceGauche]=200;
77     //PRINTF("\r \n D %d ", maxGauche);
78     //PRINTF("\r \n G %d \r \n", minDroite);
79     //gradBuffer[indiceDroite]=-200;
80     gradBuffer[Center]= 200;
81     vTaskDelayUntil( &xLastWakeTime, xDelayMain );
82 }
83
84 }
85
86
87 void task_Camera_Display(void *pvParameters) {
88     const TickType_t xDelay= Ms(400);
89     PRINTF("Task Camera Display GO\r\n");
90     PRINTF("\n\r");
91     for (;;)
92     {
93         PRINTF("camera ");
94         for(int i=0;i<128;i++)PRINTF("%d ", buffer[i]);
95         PRINTF("\r\n");
96         PRINTF("gauss ");
97         for(int i=0;i<128;i++)PRINTF("%d ", gaussBuffer[i]);
98         PRINTF("\r\n");
99         PRINTF("grad ");
100        for(int i=0;i<128;i++)PRINTF("%d ", gradBuffer[i]);
101        PRINTF("\r\n");
102        vTaskDelay(xDelay);
103    }
104 }
```

11 Code source pour Task_Camera.c

```

1 #include "CCTR.h"
2 #include <math.h>
3
4 static uint16_t buffer[128];
5 static uint16_t nullBuffer[128];
6 static uint16_t gaussBuffer[128];
7 static int16_t gradBuffer[128];
8 unsigned char indiceGauche;
9 unsigned char indiceDroite;
10 unsigned char Center;
11 /* test CCD Camera */
12 ///////////
13 #define sigma 1
14 #define K 1
15 #define BUFFER_SIZE 128
16 ///////////
17
18 float gauss[3] = {0.25, 0.5, 0.25};
19
20 void filtrage (uint16_t* buffer){
21
22     for(int i=0;i<BUFFER_SIZE;i++){
23         gaussBuffer[i] = (uint16_t)((buffer[MAX(i-1,0)]*gauss[0]) +
24             (buffer[i]*gauss[1]) +
25             (buffer[MIN(i+1,BUFFER_SIZE -1)]*gauss[2]));
26     }
27 }
28
29 void gradient(){
30     for(int i=0;i<BUFFER_SIZE;i++){
31         gradBuffer[i] = (int16_t)((gaussBuffer[MIN(i+2,BUFFER_SIZE -1)] -
32             gaussBuffer[MAX(i-2,0)]) );
33     }
34 }
35
36 void task_Camera(void *pvParameters) {
37     /*const TickType_t xDelay= Ms(50);*/
38     const TickType_t xDelay= Ms(50);
39     const TickType_t xDelayMain= Ms(5);
40     TickType_t xLastWakeTime;
41     TickType_t xLastWakeTime1;
42
43     PRINTF("Task Camera GO\r\n");
44     PRINTF("\r\n");
45     xLastWakeTime = xTaskGetTickCount();
46     xLastWakeTime1 = xLastWakeTime;
47     for (;;)
48     {
49         //xLastWakeTime = xTaskGetTickCount();
50         Camera_ImageCapture(1, nullBuffer);
51         //for(int ii=0; ii<128; ii++) buffer[ii]= 0.5*buffer[ii];
52         //vTaskDelay(xDelay);

```

```

53     vTaskDelayUntil( &xLastWakeTime, xDelay );
54     //xLastWakeTime = xTaskGetTickCount();
55     Camera_ImageCapture(1, buffer);
56     //vTaskDelayUntil( &xLastWakeTime, xDelayMain );
57     for(int ii=0; ii<128; ii++) buffer[ii]= 0.5*buffer[ii];
58     filtrage(buffer);
59     gradient();
60     int maxGauche = 0;//buffer[0];
61     int minDroite = 0;//buffer[65];
62     indiceGauche=0;
63     indiceDroite=0;
64     for(int i=1;i<64;i++){
65         if(gradBuffer[i]>maxGauche && gradBuffer[i]>60){
66             maxGauche = gradBuffer[i];
67             indiceGauche = i;
68         }
69         if(gradBuffer[i+64]<minDroite && gradBuffer[i+64]<-60{
70             minDroite = gradBuffer[i+64];
71             indiceDroite = i+64;
72         }
73     }
74     Center = (indiceDroite+indiceGauche)/2;
75     //gradBuffer[indiceGauche]=200;
76     //PRINTF("\r \n D %d ", maxGauche);
77     //PRINTF("\r \n G %d \r \n", minDroite);
78     //gradBuffer[indiceDroite]=-200;
79     gradBuffer[Center]= 200;
80     vTaskDelayUntil( &xLastWakeTime, xDelayMain );
81 }
82
83 }
84
85
86 void task_Camera_Display(void *pvParameters) {
87     const TickType_t xDelay= Ms(400);
88     PRINTF("Task Camera Display GO\r\n");
89     PRINTF("\n\r");
90     for (;;)
91     {
92         PRINTF("camera ");
93         for(int i=0;i<128;i++)PRINTF("%d ", buffer[i]);
94         PRINTF("\r\n");
95         PRINTF("gauss ");
96         for(int i=0;i<128;i++)PRINTF("%d ", gaussBuffer[i]);
97         PRINTF("\r\n");
98         PRINTF("grad ");
99         for(int i=0;i<128;i++)PRINTF("%d ", gradBuffer[i]);
100        PRINTF("\r\n");
101        vTaskDelay(xDelay);
102    }
103 }
```

12 Code source pour main_SEOC_CCTR_base.c

```

1 #include "CCTR.h"
2
3 void task_LED(void *pvParameters);
4 #define task_LED_PRIORITY 0
5
6 void task_Speed(void *pvParameters);
7 #define task_SPEED_PRIORITY 1
8
9 void task_Camera_Display(void *pvParameters);
10 #define task_CAMERA_DISPLAY_PRIORITY 2
11
12 void task_Camera(void *pvParameters);
13 #define task_CAMERA_PRIORITY 3
14
15 void task_Motor(void *pvParameters);
16 #define task_MOTOR_PRIORITY 2
17
18 void task_Steering(void *pvParameters);
19 void task_SteeringRoute(void *pvParameters);
20 #define task_STEERING_PRIORITY 2
21
22 /*
23 * @brief Application entry point.
24 */
25
26 int main(void) {
27     /*
28         Fonctions d'initialisation des diffrentes entres/sorties
29         et des protocoles de communication
30     */
31     BOARD_init_all();
32     PRINTF("Hello World CCTR SEOC Base\n\r***** Test MOTOR *****\n\r");
33     /* Cration d'une tache avec pile statique, priorit statique */
34     /*xTaskCreate(task_Motor, "Task_Motor",
35                 configMINIMAL_STACK_SIZE + 100, NULL,
36                 task_MOTOR_PRIORITY, NULL);*/
37     xTaskCreate(task_Speed, "Task_Speed",
38                 configMINIMAL_STACK_SIZE + 100, NULL,
39                 task_SPEED_PRIORITY, NULL);
40     xTaskCreate(task_SteeringRoute, "Task_Steering",
41                 configMINIMAL_STACK_SIZE + 100, NULL,
42                 task_STEERING_PRIORITY, NULL);
43     /*xTaskCreate(task_Camera_Display, "Task_Camera_Display",
44                 configMINIMAL_STACK_SIZE + 100, NULL,
45                 task_CAMERA_DISPLAY_PRIORITY, NULL);*/
46     xTaskCreate(task_Camera, "Task_Camera",
47                 configMINIMAL_STACK_SIZE + 100, NULL,
48                 task_CAMERA_PRIORITY, NULL);
49     vTaskStartScheduler();
50     /* Enter an infinite loop, but should never arrive here */
51     for (;;) {
52         ;

```

```

53     return 0 ;
54 }
55
56 /*
57 * Tache priodique pour faire clignoter la LED
58 */
59 int led_cpt=0;
60 void task_LED(void *pvParameters) {
61     const TickType_t xDelay= Ms(250);
62     led_cpt=0;
63
64     for (;;)
65     {
66         LED_RED_TOGGLE();
67         vTaskDelay(xDelay);
68         LED_GREEN_TOGGLE();
69         vTaskDelay(xDelay);
70         LED_BLUE_TOGGLE();
71
72         PRINTF("ous %d\r\n", led_cpt);
73         //PRINTF("y %d\r\nz %d\r\n", led_cpt % 100, 100-(led_cpt %100));
74         vTaskDelay(xDelay);
75         led_cpt++;
76     }
77 }
```