

RAPPORT DE PROJET DE RECHERCHE TUTEORE

4^{ème} année Automatique Électronique
Systèmes Embarqués

NXP Cup 2018

Compétition de voiture autonome sur piste



Elie TAILLARDAT Camille PUECH
Anja OLSEN

Juin 2018

Tuteurs : Patrick TOUNSI

Table des matières

Introduction	1
1. Cahier des charges fonctionnel	2
1.1. Principe de la compétition	2
1.2. Rappel des règles de la compétition	3
1.3. Stratégie adoptée	3
2. Organisation et gestion du projet.....	4
2.1. Planning prévisionnel et échéances	4
2.2. Répartition des tâches au sein de l'équipe	4
2.3. Un projet INSA composés de 3 équipes	5
2.4. Un échange privilégié avec NXP Toulouse	5
3. Conception 'hardware' de la voiture	6
3.1. Réception du kit	6
3.2. Difficultés rencontrées et surpassées	7
3.3. Des aides extérieures très précieuses	9
4. Conception 'software' et programmation périphérique	10
4.1. Outil logiciel utilisé et code de base.....	10
4.2. Structure du code	11
4.3. Configuration périphérique.....	12
4.4. Gestion de la voiture par interruption	15
4.4.1. Commande de la vision - caméra	16
4.4.2. Commande de la position – servomoteur	20
4.4.3. Commande de la vitesse - moteurs	22
5. Tests et validations	25
5.1. Conditions de tests	25
5.2. Mise en place d'outils habiles	25
5.3. Situations testées	27
6. Zoom sur la compétition	28
6.1. Organisation et tests préliminaires	28
6.2. Choix final de la stratégie adoptée et résultat	28
Conclusion.....	29
Annexes	30

Introduction

Le présent rapport va détailler dans son ensemble notre réalisation de projet tutoré, qui concernait en l'occurrence, la **participation à la compétition NXP Cup 2018** ayant eu lieu en mars dernier.

La participation à ce projet n'était pas notre premier choix lors de la présentation des sujets disponibles concernant le projet tutoré. En effet, nous avons décidé tous les trois de choisir le sujet concernant la gestion et commande du robot LEGO EV3, rassemblant des compétences de programmation, périphérique*, automatique et électronique. Ce n'est que lorsque la possibilité s'est présentée, à la dernière minute, de participer à ce concours en « remplacement » d'un sujet classique, que nous nous sommes décidés à nous lancer dans ce **challenge**.

Ce projet, rassemblant alors les mêmes compétences à exploiter et développer que le premier sujet que nous avons choisi, était alors l'occasion de réaliser un bon projet, tout en ayant en tête que l'aboutissement était une **compétition** nationale, et qui dit compétition, dit objectif encore plus pertinent et **motivateur**.

Nous allons donc revenir en détails sur les différentes **étapes clefs** de ce projet, en passant par l'organisation, la gestion de projet, la conception hardware* et software*, les différents tests menés, et le déroulé complet de la compétition finale.

Tous les mots suivis d'une astérisque * sont référencés et définis en **Annexe A2**

1. Cahier des charges fonctionnel

1.1. *Principe de la compétition*

Le principe de la NXP Cup est donc de travailler sur une petite voiture aussi bien au niveau **hardware** que **software** afin que la voiture puisse suivre un circuit précis. Le circuit que la voiture doit suivre est propre à chaque compétition, nous le découvrons donc juste au dernier moment. Nous avons cependant quelques informations à propos de ce circuit. La piste est blanche avec des bordures noires. Sa largeur est de 60 cm et il est composé des différentes pièces suivantes :

- Des lignes droites
- Des tournants
- Une bosse
- Une chicane
- Des ralentisseurs
- Un croisement

Le but de cette compétition est d'être **le plus rapide** sur le circuit proposé le jour de la compétition. En effet seul les trois équipes les plus rapides qui arrivent à finir un tour sont qualifiées pour la **finale européenne**.

Les temps de tour sont calculés au centième de seconde près. Chaque équipe à trois essais pour finir un tour, cependant à partir du moment où un tour est réussi, il n'y a plus d'essai possible. En effet, si l'équipe arrive à finir le tour au premier essai alors c'est le temps de cet essai qui est pris en compte. Si la voiture ne s'arrête pas après la ligne d'arrivée, cela entraîne une pénalité de 1 seconde sur le temps réalisé. Enfin, le code ne peut être touché entre chaque essai afin de faire des modifications.



Figure 1 : Exemples de circuits d'entraînements

1.2. Rappel des règles de la compétition

L'année 2017/2018 est une année de transition pour la compétition, en effet cette année deux types de voiture vont concourir dans deux catégories différentes. Nous avons choisi de concourir avec le nouveau modèle de voiture (modèle **Alamak**) afin que le projet puisse continuer l'année prochaine pour d'autres équipes. De plus, concourir avec le nouveau modèle permet de mettre toutes les équipes sur le même pied d'égalité. Afin de pouvoir concourir le 14 mars à Grenoble, nous étions tenu de respecter de nombreuses règles.

Tout d'abord nous étions tenus d'utiliser les kits et les microcontrôleurs fournis par NXP afin d'établir une équité entre les équipes. Nous pouvions apporter des modifications et ajouter des capteurs*, cependant nous devons respecter les quelques règles suivantes :

- Ne pas modifier la distance entre les roues
- Ne pas dépasser les dimensions maximales (25cm x 40cm x 30.5cm)
- Ne pas changer le type de batteries
- Ne pas ajouter de microcontrôleurs auxiliaires
- Ne pas ajouter plus de 16 capteurs
- Ne pas utiliser d'autres techniques de vision que la caméra
- Ne pas utiliser de module bluetooth ou wifi

Ensuite nous étions tenus de fournir un rapport technique et de déposer notre code sur une plateforme « open source » avant la compétition afin que NXP puisse ensuite les publier et les partager au nouveau participant.

1.3. Stratégie adoptée

Après avoir pris connaissance des différentes règles, nous avons ébauché une première stratégie. Nous avons reçu la voiture environ trois mois avant la compétition, nous avons donc fait le choix de ne **pas modifier le hardware** mis à notre disposition pour se concentrer sur un software plus performant. En effet, nous avons fait le choix de ne pas ajouter de capteurs.

2. Organisation et gestion du projet

2.1. *Planning prévisionnel et échéances*

Une des particularités de notre projet est que depuis le début nous avons une **deadline** imposée par le fait de participer à un concours. En effet, depuis le jour de l'inscription nous savions que la compétition aurait lieu les 14 et 15 mars à Grenoble et que, en conséquence, notre voiture devait être fonctionnelle pour cette date-là. De plus, les kits qui nous ont été envoyé ont pris du **retard** lors de livraison ce qui fait que nous les avons reçus le 25 janvier. À partir de cette date, nous avons passé tout le temps libre que nous avons à travailler sur notre voiture. Pour arriver avec une voiture en parfait état de marche le jour de la compétition, nous nous étions fixés les échéances présentées dans le diagramme de Gantt en fin de rapport (*cf. annexe A1*).

Nous avons eu du mal à suivre parfaitement les échéances présentées dans ce diagramme, mais nous n'avons jamais eu un retard conséquent ce qui nous a permis d'arriver le jour de la compétition avec une voiture **fonctionnelle**.

Une fois la compétition terminée, et sur laquelle nous reviendrons dans une partie ultérieure du présent rapport, nous avons décidé de nous consacrer à la rédaction de l'état de l'art autour de la **sécurité « software » des voitures autonomes**, ainsi qu'à la rédaction de ce rapport. Malheureusement, nous n'avons pas eu le temps de mettre en place les améliorations proposées dans la suite de ce rapport mais nous allons tout mettre en place pour les transmettre aux futures équipes participant à cette compétition.

2.2. *Répartition des tâches au sein de l'équipe*

Avant de commencer le projet, nous avons pensé à distribuer les tâches entre chaque membre du groupe, cependant nous nous sommes vite rendus compte que ce n'était pas la technique qui convenait le mieux à ce projet. En effet, une fois la programmation périphérique réalisés, il restait la partie automatique/commande qui demande beaucoup de tests. Or pour réaliser les tests sans endommager la voiture, nous avons besoin d'être au minimum deux. Nous avons donc travaillé au minimum par binôme en fonction des **disponibilités** de chacun.

Malgré ce fonctionnement, nous avons réussi à bien nous répartir le travail en termes de temps consacré par personne sur le projet. Cette méthode a fonctionné pour notre groupe car nous n'étions que trois et que nous avons réussi à consacrer de manière individuelle le même temps au projet. De plus, le fait de travailler en **binôme** a permis lorsque nous programmions de prendre du recul sur le code est ainsi de debugger* nos programmes de manière plus efficace.

Nous avons donc tous participé à chaque étape du projet ce qui nous a permis de mieux l'appréhender dans son ensemble et donc d'être plus **efficace** dans les moments où nous avons rencontré des problèmes.

2.3. Un projet INSA composés de 3 équipes

Nous ne nous sommes pas lancés seul dans ce projet. En effet, ce sont **3 équipes** de 3 étudiants chacune qui ont participé à la NXP Cup. Nous avons convenu au départ de ne pas partir sur les mêmes solutions hardware, cependant le temps dont nous disposions ne nous a pas permis de différencier ces solutions, nous avons uniquement différencié nos solutions software.

Le fait d'être trois équipes sur le même projet n'a pas toujours été quelque chose de facile. Nous avons réussi à nous aider lorsque nous rencontrions des problèmes d'ordre technique (caméra qui ne marche plus, carte qui grille, ...) ce qui a vraiment été bénéfique au projet. Cependant, nous avons souvent eu du mal à nous **organiser** pour diffuser les informations venant de notre tuteur ou même venant de NXP. Nous avons aussi eu du mal à nous mettre d'accord quant à l'organisation de notre déplacement à Grenoble. Notre équipe s'est souvent trouvée dans le rôle de **leader** quant à nos échanges avec NXP et quant aux décisions à prendre sur le moyen de transport ou encore le logement sur place.

2.4. Un échange privilégié avec NXP Toulouse

La compétition à laquelle nous avons participé dans le cadre de ce projet est organisée par l'entreprise **NXP**. NXP ayant un site sur Toulouse, nous avons eu la chance d'avoir un échange privilégié avec l'entreprise. En effet, nous avons pu rencontrer deux ingénieurs de l'entreprise qui avaient déjà travaillé sur les anciennes voitures de la compétition et qui nous ont apporté une aide précieuse au moment de monter les voitures que nous avons reçu en kit.

De plus, leur expérience sur la compétition nous a permis de nous concentrer sur les fonctionnalités essentielles de la voiture. L'expédition des kits ayant pris du retard, leurs conseils nous ont permis de nous concentrer directement sur les fonctionnalités essentielles au bon fonctionnement. En outre, grâce à ces deux ingénieurs, nous avons eu un retour sur les solutions hardware et software les plus efficaces et nous nous sommes donc dirigés sur les solutions proposées.

Nous avons aussi eu la chance, grâce à Déborah (responsable RH chez NXP), de pouvoir aller nous entraîner dans leur locaux. Locaux dans lesquels était mis à notre disposition une piste **homologuée** pour la compétition. Nous avons donc travaillé trois jeudi après-midi sur place. Malheureusement, ils ont dû renvoyer la piste homologuée afin qu'elle soit disponible pour la compétition au moment où nous en avions le plus besoin.

Le fait d'avoir pu rencontrer ces deux ingénieurs et de nous être entraîné au début sur une piste homologuée à vraiment été un atout majeur pour notre équipe.

3. Conception 'hardware' de la voiture

3.1. Réception du kit

Nous avons reçu la voiture dans un **kit en pièces détachées**. Ce kit ne contenant pas le microcontrôleur, nous avons reçu un second paquet avec ces derniers. Le kit que nous avons reçu était composé, comme nous pouvons le voir sur la *figure 2*, des pièces suivantes :

- Un châssis comprenant les 4 roues, les 2 moteurs pour contrôler les roues arrière, le servomoteur* pour la direction
- Une batterie 12V et son chargeur
- Une caméra de type « line scan »
- Une nappe pour relier la caméra au microcontrôleur
- Une carte électronique de puissance
- Une carte permettant de connecter les capteurs au microcontrôleur
- Un mât
- Un kit de vis et de pièces de rechanges



Figure 2 : Kit de la voiture en pièces détachées

La partie électronique de notre voiture est identique à celle présentée sur la *figure 3*, elle est composée de la carte de puissance au premier étage, suivie de la carte entrées/sorties au-dessus, et enfin de la carte de « contrôle » composée du microcontrôleur.

Sur cette image est aussi représentée le module wifi (en haut à gauche) et un écran LCD (à gauche) mais nous ne nous en sommes pas servi.

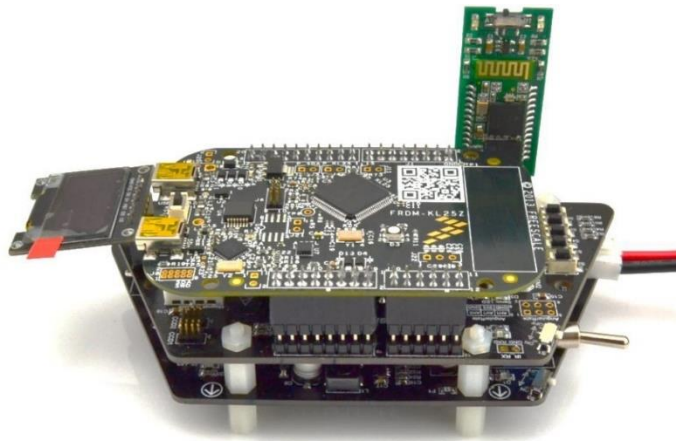


Figure 3 : Assemblage des différentes cartes électroniques

Une fois toutes les pièces reçues, nous nous sommes attelés au **montage** de la voiture. Nous pensions que cela allait être relativement rapide, mais nous nous sommes trompés. En effet, nous avons rencontrés de nombreux **problèmes techniques** qui seront décrits dans la partie suivante.

3.2. Difficultés rencontrées et surpassées

Lors de l'étape de montage, nous avons rencontré de nombreux problèmes. Nous nous sommes rapidement aperçu que le microcontrôleur ne disposait pas de **connecteurs*** nous permettant de la connecter à la carte d'entrée/ sortie. Nous avons donc dû faire appel à M. Martin qui a soudé des connecteurs bien adaptés sur le microcontrôleur. Une fois ce problème réglé, nous nous sommes rendu sur le gitbook mis à disposition des équipes participantes par NXP, pour comprendre comment connecter nos capteurs à la carte d'entrée sortie. Nous avons donc à notre disposition le schéma disponible sur la **figure 4** ci-dessous:

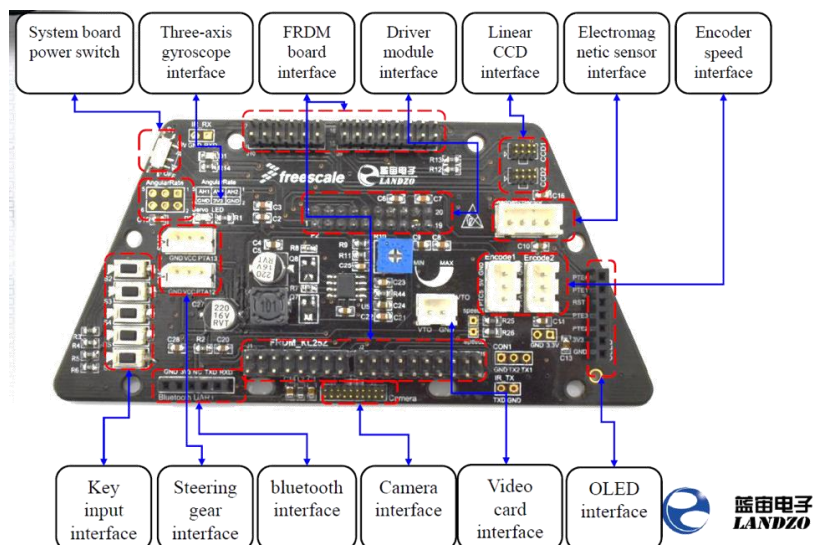


Figure 4 : Schéma des blocs électroniques de la carte intermédiaire

Après lecture de ce schéma, nous avons pu nous apercevoir qu'il y avait de nombreux problèmes de connecteurs. Les entrées femelles fournies sur les capteurs ne correspondaient pas aux entrées mâles mises à disposition sur la carte. Nous avons rencontré ce problème au moment de la connexion sur le servomoteur ; nous disposons du connecteur représenté sur la **figure 5** (1) et nous avons besoin du connecteur représenté à droite (2) :

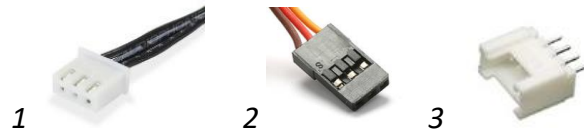


Figure 5 : Connecteur reçu (1), voulu (2) et solution (3)

Comme nous pouvons le voir sur les photos, ces connecteurs ne correspondent pas. Pour résoudre ce problème, avec l'aide de Mr. Martin, nous avons remplacé sur le shield d'entrée/sortie, le connecteur femelle existant par le **connecteur femelle** représenté sur la même figure (3).

Une fois ce problème de connecteur réglé, nous avons dû faire face à un nouveau problème de connecteur, cette fois plus compliqué à résoudre, au niveau de la caméra. Nous disposons en effet sur la carte, comme le montre la **figure 6**, du connecteur représenté sur la gauche et sur la caméra du connecteur représenté à droite :

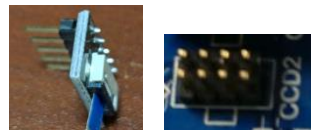


Figure 6 : Connecteurs de caméra

Encore une fois nous pouvons voir ici que les deux types de connecteurs ne sont pas du tout compatibles l'un avec l'autre. Ce problème a été rencontré par l'ensemble des équipes participant à la compétition avec le nouveau modèle de voiture. NXP nous a alors envoyé un message pour nous prévenir que nous devrions recevoir des **adaptateurs** permettant de connecter la caméra cependant la compétition approchant à grand pas, nous avons, avec l'aide de M. Martin, fabriqué notre propre adaptateur. En effet, NXP a mis à disposition de toutes les équipes le schéma **figure 7** expliquant comment connecter les différentes broches :

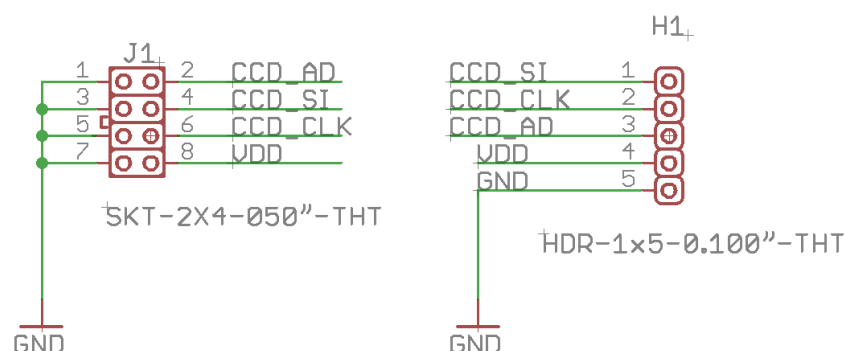


Figure 7 : Schéma de connexion des pins

Nous avons donc simplement relié à l'aide de fils les différents pins concernés, et avons pu finalement brancher notre caméra.

Une fois la voiture montée et les capteurs connectés, nous avons pu alimenter les différentes cartes électroniques. Cependant, nous avons constaté un dysfonctionnement des switch permettant d'alimenter la carte (system board power switch), nous avons donc décidé de les remplacer par un **point de soudure** afin que ces dernières soit toujours en mode **ON**. Ceci nous a aussi permis d'éviter que les cartes s'éteignent subitement lors du passage de la voiture sur les bosses.

Nous avons fait face à un dernier problème d'ordre technique, la pin de sortie permettant d'alimenter le servomoteur ne fonctionnait pas. Cette dernière ne délivrait pas un bon signal de **PWM***. Heureusement que deux sorties nous permettaient de commander le servomoteur, nous avons donc simplement changé le câble de place et reconfiguré les périphériques afin d'utiliser la deuxième sortie permettant de commander le servomoteur.

Ces différents problèmes ont réduit le temps que nous souhaitions passer à implémenter le software de notre voiture. Ils nous ont permis néanmoins de nous rendre compte que le hardware est aussi une part importante du travail d'ingénieur en automatique, notamment le **débogage** à l'aide de l'oscilloscope.

3.3. Des aides extérieures très précieuses

Comme il est possible de le constater dans la partie précédente, sans l'aide précieuse de M. Martin, nous n'aurions certainement pas réussi à résoudre certains de nos problèmes. En effet, les points à souder étaient à 1mm les uns des autres. Cette aide a été un avantage énorme face aux compétiteurs des autres écoles qui ont souvent perdu du temps à résoudre ces problèmes d'ordre technique. Nous tenions donc à le **remercier** dans ce rapport.

Nous tenions aussi à remercier les deux ingénieurs de chez NXP qui nous ont délivré de précieux conseils mais qui nous ont surtout permis de nous rendre compte rapidement que si notre servomoteur ne marchait pas, c'était tout simplement car la PIN de sortie de notre microcontrôleur était défectueuse et qu'elle ne délivrait pas une PWM comme attendue.

Nous tenions aussi à remercier toutes les équipes qui ont contribué à l'écriture du gitbook et qui nous ont permis de résoudre plus rapidement les nombreux problèmes que nous avons rencontrés. De même, le fait de participer à la compétition à trois équipes nous a permis de nous **entraider** lorsque nous faisons face à des problèmes communs et ainsi accélérer le processus de résolution.

Sans l'aide précieuse des personnes mentionnées ci-dessus, nous n'aurions certainement pas réussi à mener ce projet à terme.

4. Conception ‘software’ et programmation périphérique

4.1. Outil logiciel utilisé et code de base

Nous avons alors pu commencer la programmation du microcontrôleur embarqué grâce à un environnement de développement (IDE) *figure 8*, Code Warrior, conçu également par NXP. Nous avons d’abord opté pour l’IDE Kinetis, mais suite à de nombreux problèmes d’installations, nous avons changé d’avis pour utiliser CodeWarrior.



Figure 8 : IDEs utilisées pour coder la ‘board’

Il a fallu un peu de temps pour mettre toutes les **librairies*** en place, mais une fois l’environnement complètement installé, nous avons pu allumer une LED de la carte et donc vérifier le bon fonctionnement basique de notre carte et la bonne compilation réalisée par l’IDE sans erreurs due à une mauvaise installation.

Un **code de base** faisant appel aux principales fonctionnalités du véhicule et des exemples de configuration périphérique a été donné sur le forum NXP par un des organisateurs. En effet, un des problèmes très récurrents durant ce projet (et dont nous verrons vraiment les issues en détails lors de la phase de compétition en fin de rapport), était l’utilisation d’un nouveau modèle de voiture cette année, et donc aussi de carte et circuit embarqué. Ce code de base ne marchait pas dans un premier temps, et il a fallu pour nous déboguer d’entrée.

Nous avons passé beaucoup de temps à comprendre le code de base, afin de pouvoir en tirer le maximum et pouvoir ensuite développer nos stratégies propres dans différentes fonctions. La première visite chez NXP, durant laquelle nous étions le seul groupe présent au complet, nous avons pu avoir l’aide de deux ingénieurs dans cette démarche. Et cette aide n’a pas été de trop comme nous avons pu l’écrire précédemment, car même eux ont eu de très grandes difficultés à comprendre une partie du code et à figurer d’où notre principal problème venait. Nous avons pu utiliser à ce moment-là un oscilloscope afin de visualiser les différentes sorties associées à la commande des roues et du servo-moteur. Des pins étaient en effet mal configurés, et la broche 12 censée être utilisée pour le servomoteur avait un mauvais fonctionnement, ce qui nous a déporté sur la broche 13 pour gérer cette fonctionnalité.

Une fois que le code de base fonctionnait parfaitement et que nous avons pris en main chaque ligne du programme, nous avons pu commencer à **intégrer nos différentes corrections**.

4.2. Structure du code

Nous avons divisé le code en 3 fichiers différents : *main.c*, *carSkills.c* et *carSkills.h*. Nous allons dans un premier temps présenter le *carSkills.h* pour passer en revue toutes les fonctionnalités que nous avons pu implémenter dans la voiture.

Ce fichier est composé de plusieurs `#define` concernant notamment les valeurs définies pour nos correcteurs (gains du PID par exemple), des configurations périphériques raccourcies, et des fonctions que nous allons implémenter :

```
-----

#ifndef CARSKILLS_H_ /* CARSKILLS_H_ guards */
#define CARSKILLS_H_

// Define threshold for black line recognition
#define THRESHOLD 100 // Difference between B & W

// Defines for Direction PD Servo Control Loop
#define KP 40 // Proportional coefficient
#define KDP 45 // Differential coefficient
#define MOTOR_SPEED 200
#define MOTOR_SPEED_MIN 90
#define KV (2/1000)
#define KS (200*MOTOR_SPEED/2000)
#define KSD 0

// Defines for LineScan Camera
#define TAOS_DELAY asm ("nop") // minimal delay time
#define TAOS_SI_HIGH GPIOB_PDOR |= (1<<8) // SI on PTB8
#define TAOS_SI_LOW GPIOB_PDOR &= ~(1<<8) // SI on PTB8
#define TAOS_CLK_HIGH GPIOB_PDOR |= (1<<9) // CLK on PTB9
#define TAOS_CLK_LOW GPIOB_PDOR &= ~(1<<9) // CLK on PTB9

// Defines for led states
#define LED_RED_OFF GPIOD_PDOR |= GPIO_PDOR_PDO(1<<18)
#define LED_RED_ON GPIOD_PDOR &= ~GPIO_PDOR_PDO(1<<18)
#define LED_GREEN_OFF GPIOD_PDOR |= GPIO_PDOR_PDO(1<<19)
#define LED_GREEN_ON GPIOD_PDOR &= ~GPIO_PDOR_PDO(1<<19)
#define LED_BLUE_OFF GPIOD_PDOR |= GPIO_PDOR_PDO(1<<1)
#define LED_BLUE_ON GPIOD_PDOR &= ~GPIO_PDOR_PDO(1<<1)

#define BLACK_LINE_RIGHT 126
#define BLACK_LINE_LEFT 1

// Functions declaration
void ImageCapture(void);
int sumCurrentLine(int imageData[128]);
int cntPics(int imageData[128]);
int detectEndOfRace(int pics_cnt, int oldImageSum, int currentImageSum);
int findRightBlackLine(int imageData[128]);
int findLeftBlackLine(int imageData[128]);
int findRoadMiddle(int blackLineRight, int blackLineLeft);
int setServoPosition(int servoPositionOld, int diffOld, int diff);

#endif /* CARSKILLS_H_ */
```

Nous reviendrons aux différents coefficients correcteurs dans les parties suivantes. On peut retrouver aussi dans ce fichier les différents **#define** concernant les configurations périphériques qui se feront dans la procédure **void ImageCapture(void)** et des raccourcis afin de changer l'état des LEDs de manière plus facile avec l'utilisation de masque, que nous avons pu voir durant l'UF du premier semestre consacrée à la programmation périphérique sur puce STM32.

On retrouve de même le seuil **THRESHOLD** défini pour distinguer le blanc du noir sur le circuit, valeur que l'on doit adapter un peu en fonction de l'éclairage ambiant de la pièce, et de la réflexion sur la piste comme nous avons pu avoir avec les bouts de piste disponibles du l'INSA.

Enfin, on peut retrouver les différentes fonctions que nous allons implémenter dans le définir dans **carSkills.c** et ensuite appeler dans l'interruption* qui gèrera le comportement du véhicule. Il s'agit de fonctions basiques dont nous aurons besoin par la suite : gestion de la caméra, détection de lignes du circuit (gauche et droite), commande de position du servomoteur, détection de pics de différence de luminosité, et détection de fin de course.

4.3. Configuration périphérique

Nous allons donc maintenant faire une description assez succincte de la partie **int main(void)** du fichier **main.c**, car elle n'est utile qu'à la configuration par masquage de chaque broche que nous allons utiliser sur la carte, des GPIO, des clocks... En effet, la partie qui gèrera la voiture se trouve dans l'interruption **void FTM1_IRQHandler()** qui sera détaillée en plusieurs parties au cours des trois prochaines sous-sections de ce rapport. La description des lignes suivantes se trouve sous forme de commentaires insérés directement dans le code (et en anglais oui, une bonne habitude à prendre), ce qui a été aussi très utile pour nous rappeler de manière claire et rapide certaines informations au cours du projet.

```
-----  
#include "derivative.h" /* include peripheral declarations */  
  
int main(void) {  
    /* Clocks configuration */  
    // configure clock to 48 MHz from a 8 MHz crystal  
    MCG_C2 = (MCG_C2_RANGE0(1) | MCG_C2_EREF0_MASK);  
    // configure the oscillator settings  
    MCG_C1 = (MCG_C1_CLKS(2) | MCG_C1_FRDIV(3));  
    // divider for 8 MHz clock  
    for (i = 0 ; i < 24000 ; i++) // wait for OSCINIT to set  
    // now in FBE mode  
    MCG_C6 |= MCG_C6_CME0_MASK;  
    // enable the clock monitor  
    MCG_C5 |= MCG_C5_PRDIV0(1);  
    // set PLL ref divider to divide by 2  
    temp_reg = MCG_C6;  
    // store present C6 value (as CME0 bit was previously set)  
    temp_reg &= ~MCG_C6_VDIV0_MASK;
```

```

// clear VDIV settings
temp_reg |= MCG_C6_PLLS_MASK | MCG_C6_VDIV0(0);
// write new VDIV and enable PLL
MCG_C6 = temp_reg;
// update MCG_C6
for (i = 0 ; i < 4000 ; i++) // wait for PLLST status bit to set
// now in PBE mode
SIM_CLKDIV1 = (SIM_CLKDIV1_OUTDIV1(1) | SIM_CLKDIV1_OUTDIV4(1));
// core clock, bus clock div by 2
MCG_C1 &= ~MCG_C1_CLKS_MASK;
// switch CLKS mux to select the PLL as MCGCLKOUT
for (i = 0 ; i < 2000 ; i++) // Wait for clock status bits to update
// now in PEE mode, core and system clock 48 MHz, bus and flash clock
24 MHz
// turn on all port clocks
SIM_SCGC5 = SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTB_MASK |
SIM_SCGC5_PORTC_MASK | SIM_SCGC5_PORTD_MASK | SIM_SCGC5_PORTE_MASK;
// turn on all TPM clocks
SIM_SCGC6 |= SIM_SCGC6_TPM0_MASK | SIM_SCGC6_TPM1_MASK |
SIM_SCGC6_TPM2_MASK;
// turn on ADC0 clock
SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;

/* GPIOs configuration */

// set ADC inputs to ADC inputs
PORTC_PCR2 |= PORT_PCR_MUX(0); // Camera 1 PTC2 ADC0_SE11

// set GPIOs to GPIO function
PORTC_PCR0 |= PORT_PCR_MUX(1); // PTC0 Push Button S2
PORTB_PCR18 |= PORT_PCR_MUX(1); // PTB18 LED red
PORTB_PCR19 |= PORT_PCR_MUX(1); // PTB19 LED green
PORTD_PCR1 |= PORT_PCR_MUX(1); // PTD1 LED blue
PORTB_PCR8 |= PORT_PCR_MUX(1); // PTB8 Camera SI
PORTB_PCR9 |= PORT_PCR_MUX(1); // PTB9 Camera Clock
PORTA_PCR4 = 0; // Set PTA4 Pin Control Reg to 0
PORTA_PCR4 |= PORT_PCR_MUX(1); // PTA4 Motor 1 In 1 (direction)
PORTA_PCR5 |= PORT_PCR_MUX(3); // PTA5 Motor 1 In 2 (speed)
// PTA5 TPM0_CH2
PORTC_PCR8 |= PORT_PCR_MUX(1); // PTC8 Motor 2 In 1 (direction)
PORTC_PCR9 |= PORT_PCR_MUX(3); // PTC9 Motor 2 In 2 (speed)
// PTC9 TPM0_CH5

// set GPIO outputs to outputs
GPIOB_PDDR |= (1<<18); // PTB18 LED red
GPIOB_PDDR |= (1<<19); // PTB19 LED green
GPIOB_PDDR |= (1<<1); // PTD1 LED blue
GPIOB_PDDR |= (1<<8); // PTB8 Camera SI
GPIOB_PDDR |= (1<<9); // PTB9 Camera Clock
GPIOA_PDDR |= (1<<4); // PTA4 Motor 1 In 1
GPIOA_PDDR |= (1<<5); // PTA5 Motor 1 In 2 (direction)
GPIOC_PDDR |= (1<<8); // PTC8 Motor 2 In 1 (direction)
GPIOC_PDDR |= (1<<9); // PTC9 Motor 2 In 2
GPIOA_PDDR |= (1<<13); // PTA13 Servo Motor

// set GPIO input to input
GPIOC_PDDR &= ~ (1<<0); // PTC0 Push Button S2
// update LED states
LED_RED_OFF;
LED_GREEN_OFF;
LED_BLUE_ON;

```

```

// set PWM outputs
PORTA_PCR13 |= PORT_PCR_MUX(3); // Servo Motor 1 Out PTA13 TPM1_CH1

// configure TPM clock source to be 48 MHz
SIM_SOPT2 |= SIM_SOPT2_TPMSRC(1); // MCGFLLCLK clock or MCGPLLCLK/2
SIM_SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK; // use MCGPLLCLK/2 clock when
running from 8 MHz and PLL

// set TPM prescaler before enabling the timer
TPM0_SC |= 3; // prescaler for TPM0 (Motor) is 8
TPM1_SC |= 3; // prescaler for TPM1 (Servo) is 8

// TPM modulo register, set frequency
TPM0_MOD = 600; // modulo TPM0 (Motor), periode = 0.10 ms (10000 Hz)
TPM1_MOD = 30000; // modulo TPM1 (Servo), periode = 10 ms (100 Hz)

// set TPM clock mode to enable timer
TPM0_SC |= TPM_SC_CMOD(1); // enable TPM0 (Motor)
TPM1_SC |= TPM_SC_CMOD(1); // enable TPM1 (Servo)

// configure TPM channel outputs high-true pulses
TPM0_C2SC = 0x28; // TPM0 channel1 Motor 1 In 1 speed left
TPM0_C5SC = 0x28; // TPM0 channel5 Motor 2 In 2 speed right
TPM1_C1SC = 0x28; // TPM1 channel1 Servo 1

// TPM channel value registers, sets duty cycle
TPM1_C1V = 7000; // TPM1 channel1 Servo 1 ca. 1.5 ms (middle)

// initial configuration of motors
TPM0_C2V = 100; // TPM0 channel1 left Motor 1 In 1 slow forward
TPM0_C5V = 100; // TPM0 channel5 right Motor 2 In 2 slow forward
GPIOA_PDOR &= ~(1<<4); // Set PTA4 left Motor 1 In 2 forward
GPIOC_PDOR &= ~(1<<8); // Set PTC8 right Motor 2 In 1 forward

// configure interrupts in TPM1
TPM1_SC |= TPM_SC_TOIE_MASK; // enable overflow interrupt in TPM1 (10
ms rate)

// ADC0 clock configuration
ADC0_CFG1 |= 0x01; // clock is bus clock divided by 2 = 12 MHz

// ADC0 resolution
ADC0_CFG1 |= 0x08; // resolution 10 bit, max. value is 1023

// ADC0 conversion mode
ADC0_SC3 = 0x00; // single conversion mode

// enable interrupts 18 (TPM = FTM1) in NVIC, no interrupt levels
NVIC_ICPR |= (1 << 18); // clear pending interrupt 18
NVIC_ISER |= (1 << 18); // enable interrupt 18

// enable interrupts globally
asm (" CPSIE i "); // enable interrupts on core level

// Main loop
for(;;) { // endless loop
    // do nothing
}
return 0;
}

```


4.4. Gestion de la voiture par interruption

Le contrôle du comportement de la voiture se fait à l'intérieur d'une **interruption** qui se déclenche toutes les 10 ms lors du dépassement du seuil fixé dans TPM1 avec le « timer » associé. Cette interruption est structurée comme suit :

```
void FTM1_IRQHandler() { // TPM1 ISR

    TPM1_SC |= 0x80;    // clear TPM1 ISR flag

    /* Bloc "Vision"
       - détection fin de course
       - détection des lignes et comparaison

    * Bloc "Position"
       - gestion position servomoteur
       - détection des lignes et comparaison

    * Bloc "Vitesse"
       - gestion vitesse en fonction des virages
       - différentiel sur les roues
       - accélération en 'ligne droite'
    */
}
```

Nous allons donc détailler chaque bloc dans les parties suivantes. Nous avons préféré le faire sous cette forme plutôt que de mettre tout le code en annexe, afin de pouvoir suivre un **raisonnement logique** et suivre les consignes que la voiture reçoit pas à pas.

Les variables globales sont les suivantes :

```
int i; // counter for loops
int temp_reg; // temporary register
int imageData[128]; // array to store the LineScan image

int blackLineRight; // position of the black line on the right
int blackLineLeft; // position of the black line on the left
int roadMiddle = 0; // calculated middle of the road

int diff = 0; // actual difference from line to middle pos
int diffOld = 0; // previous difference from line to middle pos

int servoPosition = 0; // actual pos of the servo relative to middle
int servoPositionOld = 0; // old servo position

int diffCmd; // differential applied to the rear wheels
int speedCmd; // speed to be deduced depending on the angle
int ajustedSpeed; // same as above but after precautions

int cntMotorsOff = 0; // counts for how long the motors are cut
int cntHighSpeed = 0; // "" for how long the car is at high speed
int stopMotors = 0; // boolean to know if motors are off

int currentImageSum = 0; // sum of all integers in the imageData array
int oldImageSum = 0; // to compare with the new one and detect end
int endOfRace = 0; // boolean to know if the race is over
```

4.4.1. Commande de la vision - caméra

```
/* Bloc "Vision" */
ImageCapture(); // Capture Line Scan Image

if (!endOfRace) {
    // Detect end of race
    // replace the previous value by the current one
    oldImageSum = currentImageSum;
    currentImageSum = sumCurrentLine(imageData);
    pics_cnt = cntPics(imageData);
    endofRace = detectEndOfRace(pics_cnt, oldImageSum, currentImageSum);
    // return 1 if true

    // Find black line on the right side
    blackLineRight = findRightBlackLine(imageData);
    // Find black line on the left side
    blackLineLeft = findLeftBlackLine(imageData);
    // Find middle of the road, 64 for strait road
    roadMiddle = findRoadMiddle(blackLineRight, blackLineLeft);

    // Store old value
    diffOld = diff; // store old difference
    // Find difference from real middle
    diff = roadMiddle - 64; // calculate actual difference
    // plausibility check
    if (abs (diff - diffOld) > 50) diff = diffOld;

    ...
}
```

La première étape consiste à capturer une image grâce à la caméra, qui ne détecte que le niveau de luminosité entre deux points. La procédure **ImageCapture** (visible en détail plus loin) renvoie le résultat dans la variable `int imageData[128]`, qui est un tableau de 128 entiers représentant différentes valeurs de luminosité (plus c'est sombre, plus la valeur est petite).

Ensuite, l'ensemble des opérations effectuées dans l'interruption ne se fera que si la variable `endOfRace` n'est pas déjà à `true` (1), c'est-à-dire que la voiture n'a pas encore franchie la ligne d'arrivée, symbolisée par deux pointillés en milieu de route. Notre algorithme détectant la fin de course est assez simple à expliquer, et se base sur deux paramètres. On somme tout d'abord la totalité des valeurs entières représentant les niveaux de luminosité capturées par la caméra - `sumCurrentLine(imageData)` - et on regarde combien de « pics » sont présents au sein de ce tableau par comparaison successives des valeurs : on remarque ainsi la présence de pointillées.

Notre fonction `detectEndOfRace(pics_cnt, oldImageSum, currentImageSum)` prenant en argument le nombre de pics détectés, la somme de la précédente image, et la somme actuelle permet alors de déterminer si la ligne d'arrivée a été détectée. En effet, cette fonction (détaillée en fin de cette sous-partie également) renvoie `true` (1) si le nombre de pic est supérieur à 4, et que la somme des luminosités est inférieure à celle précédemment

calculée, signifiant bien qu'une ligne noire en pointillée a été détectée, le noir renvoyant à la caméra de plus faibles valeurs entières.

La suite de cette partie de code permet de trouver la ligne noire à gauche de la route, celle à droite, et ainsi trouver le milieu de la piste pour la voiture. Ces fonctions détaillées en annexe de cette sous-partie, ont le même principe : elles parcourent le tableau `imageData` et renvoie l'indice où la détection de noir a été faite :

- Entre **0 et 64** pour la gauche - `blackLineLeft`
- Entre **64 et 128** pour la droite - `blackLineRight`

La position de la voiture est alors une **moyenne** des deux. Cependant, les cas extrêmes sont aussi traités, à savoir si uniquement une des lignes est trouvée, ou si aucune n'est détectée (signifiant que la voiture est déjà en milieu de piste). Dans ces cas-là, il faut recentrer la voiture, et la fonction `findRoadMiddle` ajuste la valeur de `roadMiddle` pour la voiture.

Enfin, on calcule la différence dans la variable `diff` qu'il existe entre cette valeur trouvée et le vrai milieu de la route, à savoir la valeur 64. Un test est réalisé avec l'ancienne valeur de différence `diffOld` pour vérifier que la valeur reste cohérente, et dans le cas inverse, reprendre l'ancienne valeur trouvée. La valeur de `diff` sera utilisée par la suite.

Le détail des fonctions appelées est disponible en page suivante ; elles sont présentes dans le fichier ***carSkills.c***, et détaillées grâce aux commentaires inclus directement dans le code.

```

void ImageCapture(void) {
    unsigned char i;
    ADC0_CFG2 |= 0x10; // select B side of the MUX
    TAOS_SI_HIGH;
    TAOS_DELAY;
    TAOS_CLK_HIGH;
    TAOS_DELAY;
    TAOS_SI_LOW;
    TAOS_DELAY;
    // inputs data from camera (first pixel)
    ADC0_SC1A = 11; // set ADC0 channel 11
    // wait until ADC is ready
    while ((ADC0_SC1A & ADC_SC1_COCO_MASK) == 0);
    imageData[0] = ADC0_RA;
    TAOS_CLK_LOW;

    for (i = 1; i < 128; i++) {
        TAOS_DELAY;
        TAOS_DELAY;
        TAOS_CLK_HIGH;
        TAOS_DELAY;
        TAOS_DELAY;
        // inputs data from camera (one pixel each time through loop)
        ADC0_SC1A = 11;
        // wait until ADC is ready
        while ((ADC0_SC1A & ADC_SC1_COCO_MASK) == 0);
        imageData[i] = ADC0_RA;
        TAOS_CLK_LOW;
    }
    TAOS_DELAY;
    TAOS_DELAY;
    TAOS_CLK_HIGH;
    TAOS_DELAY;
    TAOS_DELAY;
    TAOS_CLK_LOW;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int sumCurrentLine(int imageData[128]) {
    int currentImageSum = 0;
    for (i = 0; i < 127; i++) {
        currentImageSum += imageData[i];
    }
    return currentImageSum;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int cntPics(int imageData[128]) {
    int compareData = THRESHOLD;
    int pics_cnt = 0;
    for (i = 0; i < 127; i++) {
        imageDataDifference[i] = abs (imageData[i] - imageData[i+1]);
        if (imageDataDifference[i] > compareData) {
            compareData = imageDataDifference[i];
            pics_cnt++; // increment pics_cnt value
        }
    }
    return pics_cnt;
}

```

```

int detectEndOfRace(int pics_cnt, int oldImageSum, int currentImageSum) {
    if (pics_cnt > 4 && currentImageSum < oldImageSum * 0.9)
        // 0.9 = security coefficient
        return 1;    // the end of the race is detected
    else
        return 0;
}

////////////////////////////////////

int findRightBlackLine(int imageData[128]) {
    int compareData = THRESHOLD;
    int blackLineRight = BLACK_LINE_RIGHT;
    int imageDataDifference[128];
    // array to store the PineScan pixel difference
    for (i = 64; i < 127; i++) {
        // calculate difference between the pixels
        imageDataDifference[i] = abs (imageData[i] - imageData[i+1]);
        if (imageDataDifference[i] > compareData) {
            compareData = imageDataDifference[i];
            blackLineRight = i;
        }
    }
    return blackLineRight;
}

////////////////////////////////////

int findLeftBlackLine(int imageData[128]) {
    int compareData = THRESHOLD;
    int blackLineLeft = BLACK_LINE_LEFT;
    int imageDataDifference[128];
    // array to store the PineScan pixel difference
    for (i = 64; i > 0; i--) {
        // calculate difference between the pixels
        imageDataDifference[i] = abs (imageData[i] - imageData[i-1]);
        if (imageDataDifference[i] > compareData && i > 2) {
            compareData = imageDataDifference[i];
            blackLineLeft = i;
        }
    }
    return blackLineLeft;
}

////////////////////////////////////

int findRoadMiddle(int blackLineRight, int blackLineLeft) {
    roadMiddle = (blackLineLeft + blackLineRight) / 2;

    // if a line is only on the the left side
    if (blackLineRight > 124)
        roadMiddle = blackLineLeft + 50;
    // if a line is only on the the right side
    if (blackLineLeft < 3)
        roadMiddle = blackLineRight - 50;
    // if no line on left and right side
    if ((blackLineRight > 124) && (blackLineLeft < 3))
        roadMiddle = 64;

    return roadMiddle;
}

```

4.4.2. Commande de la position – servomoteur

Le servomoteur dirige la direction des roues avant de la voiture. La position du servomoteur doit s'adapter à la position de la voiture sur la piste. La variable `servoPosition` contient un entier qui indique la position du servomoteur. N'ayant aucun élément d'information sur la plage du servomoteur, nous avons dû trouver **expérimentalement** la valeur qui donnait l'angle maximal à droite et l'angle maximal à gauche. Pour ce faire, nous avons simplement exclu le bloc position et nous avons codé en dur la valeur écrite dans `TPM1_C1V`. Après de nombreux essais, nous avons trouvé la plage égale à **[6000 ; 8000]** où 7000 représente le milieu. La variable `diff` dans le bout de code ci-dessous provient du bloc image, où `diff` représente la différence en « pixels » (la plage de la caméra étant de 0 à 127) entre la position de la voiture et le milieu de la piste. Cette variable peut donc théoriquement varier entre **-64 et 63**.

Selon la position de la voiture, nous enlevons donc de 7000 la valeur provenant de la fonction `setServoPosition`. Si par exemple la voiture est au milieu de la route, `diff` vaut nul et `TPM1_C1V` reçoit donc la valeur 7000

```
/* Bloc "Position" */
// Direction Control Loop: PD Controller
servoPositionOld = servoPosition;
servoPosition = setServoPosition(servoPositionOld, diffOld, diff);
// Set channel 0 PWM_Servo position
TPM1_C1V = 7000 - servoPosition; // set channel 1 PWM_Servo
```

Nous avons implémenté un correcteur **Proportionnel Dérivé** (PD) pour gérer la position du servomoteur. Les ingénieurs chez NXP nous ont dit que la fait d'introduire un facteur intégral dans ce correcteur n'apporterait rien, nous n'avons donc pas exploré cette piste. Une partie du métier d'ingénieur est aussi de se servir des informations issues du travail déjà effectué par d'autres ingénieurs, et nous ne sommes alors focalisés sur le choix des paramètres du correcteur PD. Les contraintes de temps jouaient aussi un rôle dans ce choix.

Le détail de la fonction `setServoPosition` appelée est disponible ci-dessous :

```
int setServoPosition(int servoPositionOld, int diffOld, int diff) {
    servoPosition = KP * diff + KDP * (diff - diffOld);

    if (servoPosition < -1000)
        servoPosition = - 1000;
    else if (servoPosition > 1000)
        servoPosition = 1000;

    // prevent saturation of variation of servo
    if ((servoPosition - servoPositionOld) > 250)
        servoPosition = servoPositionOld + 250;
    else if ((servoPosition - servoPositionOld) < - 250)
        servoPosition = servoPositionOld - 250;

    return servoPosition;
}
```

Pour régler les coefficients **KP** et **KDP** du correcteur PD, nous avons tenté plusieurs options. Nous avons par exemple essayé de mettre en œuvre la **méthode de Ziegler-Nichols** pour la réglage d'un PD. Cette méthode consiste à coller le KDP à zéro, puis à augmenter le facteur proportionnel, ici KP, jusqu'à avoir un système **auto-oscillant**. À partir de cette valeur de KP et de la période des oscillations, nous pouvons calculer les facteurs proportionnels et dérivés. Pour réussir à faire la méthode de Ziegler-Nichols sur la voiture réelle, nous avons créé une longue piste avec tous les bouts droits dont nous disposions, et nous avons augmenté progressivement la valeur de KP. Une fois auto-oscillante, nous avons chronométré la voiture et compté le nombre de périodes pour avoir la période d'oscillations.

Étonnamment pour nous, les résultats pour KP et KDP obtenus par Ziegler-Nichols n'ont pas marché sur la voiture. Elle était très oscillante et même instable. Nous avons ensuite découvert que la méthode de Ziegler-Nichols a de nombreuses limitations. Cette méthode est pertinente pour de systèmes **linéaires et lents**, dont la réponse est dominée par un simple pôle. Des pôles de degrés supérieurs introduisent un décalage de phase qui influence beaucoup sur la stabilité en boucle fermée.

Nous avons donc fait des réglages expérimentaux après avoir consulté les ingénieurs chez NXP. Après chaque essai nous avons changé les coefficients en s'appuyant sur les caractéristiques que nous avons souhaité pour le système. Pour un correcteur PID, chacun des facteurs dans le correcteur représente un gain ou une perte de rapidité, de précision et de stabilité, comme le montre la **figure 9**.

	Précision	Stabilité	Rapidité
P	↗	↘	↗
I	↗	↘	↘
D	↘	↗	↗

Figure 9 : Effets d'un correcteur PID sur un système

[source : https://fr.wikipedia.org/wiki/Régulateur_PID#/media/File:TableauPID.jpg]

Le but étant d'avoir un système rapide et stable, nous avons fini par avoir un KPD assez élevé, même plus élevé que le facteur proportionnel KP. Ces résultats ont été trouvés par, comme indiqué précédemment, de nombreux **essais** avec des constructions de piste différents. Comme nous étions limités en matériel, nous avons fait ce que nous avons pu avec les bouts que nous avions, et nous avons par exemple testé des pistes classiques en rond, en forme de S, avec des lignes longues et ensuite un tournant, etc.

Chaque fois que nous avons pu augmenter la vitesse maximale de la voiture, nous avons dû re-régler les coefficients de ce correcteur PD pour arriver à prendre les tournants à une vitesse plus élevée.

Nous avons aussi mis en place un bloc *if* pour nous assurer que la commande soit comprise entre 1000 et -1000, car nous nous sommes aperçus que si la voiture reçoit une valeur qui n'est pas comprise entre 6000 et 8000 sur la broche du servomoteur, `TPM1_C1V`, le servomoteur se comporte de manière **indéterminée**.

Ensuite, nous avons rencontré des problèmes de « blocage » du servomoteur, c'est-à-dire qu'au bout d'un certain temps, les roues avant ne tournaient plus, peu importe ce qu'observait la caméra. Cela a été difficile de trouver ce qui pouvait causer ce problème. En mettant en place le bloc *if* qui vérifie si la commande a changé de plus que 250 depuis l'interruption dernière, on fixe le changement à 250. Cela a réglé le problème, donc visiblement, le servomoteur ne peut pas tourner plus qu'une certaine **limite** en un seul coup. La limite a été trouvée en diminuant progressivement ce chiffre jusqu'à ce que l'on observe le blocage.

4.4.3. Commande de la vitesse - moteurs

```
/* Bloc "Vitesse" */

// Manage car speed
if (servoPosition < 350 && servoPosition > -350 && stopMotors != 1)
    cntHighSpeed++;

if (servoPosition < 300 && servoPosition > -300) {
    speedCmd = 0;
    if (stopMotors != 1) cntHighSpeed++;
} else {
    speedCmd = KS * servoPosition + KSD * (servoPosition -
servoPositionOld);
    cntHighSpeed = 0;
}

if (MOTOR_SPEED - abs(speedCmd) < MOTOR_SPEED_MIN) {
    speedCmd = MOTOR_SPEED - MOTOR_SPEED_MIN;
}

ajustedSpeed = MOTOR_SPEED - abs(speedCmd);
```

Cette partie du bloc vitesse est indissociable de la stratégie voulue durant la réalisation d'un tour de piste : gagner du temps partout où l'on peut en gagner. Et c'est le cas pour les lignes droites où la voiture est en mesure **d'accélérer** et ainsi de gagner du temps sur nos concurrents directs.

Le premier « if » dans le bloc vitesse incrémente un compteur si l'on considère que l'on roule tout droit, donc à haute vitesse. Nous avons décidé de mettre cette plage considérée comme « tout droit » à [-350 ; 350] après de nombreux essais. On reviendra à l'utilité de ce **compteur**.

Ensuite, on déroule tout un algorithme pour prendre en compte le plus de scénarios envisageables possibles. Les différentes parties ont été rajoutées au fur et à mesure de nos découvertes de nouveaux scénarios.

Le deuxième instruction **if** regarde si la voiture roule tout droit avec une plage encore plus petite que celle d'avant. Si c'est vrai, nous n'allons pas diminuer la vitesse et nous affectons donc la valeur zéro à la variable `speedCmd`. La voiture roulera donc à la vitesse maximale. Puis, dans le même cas, on regarde si une variable locale `stopMotors` est à zéro, et si c'est bien le cas, le compteur de haute vitesse est incrémenté. Cette fonctionnalité sera détaillée dans la suite du rapport.

Si, dans le cas contraire, la voiture n'est pas considérée comme roulant en ligne droite, nous calculons la valeur affectée à `speedCmd` par un **correcteur PD** avec la même analogie que pour la commande du servomoteur.

La dernière condition permet de savoir si la différence entre la vitesse actuelle et la commande que l'on souhaite appliquer est inférieure à un seuil que nous avons fixé appelé `MOTOR_SPEED_MIN`. Si c'est le cas, nous affectons plutôt la différence entre `MOTOR_SPEED` et `MOTOR_SPEED_MIN` à `speedCmd`. Le but de cette instruction est d'assurer que la vitesse de la voiture ne soit **pas trop basse**. De cette manière, elle ne changera jamais de plus que `MOTOR_SPEED_MIN`. L'objectif est d'avoir une voiture **réactive**, mais avec une réactivité néanmoins limitée.

```
// Bloc Differential
diffCmd = KV * servoPosition;
if (ajustedSpeed - abs(diffCmd) < MOTOR_SPEED_MIN) {
    diffCmd = ajustedSpeed - MOTOR_SPEED_MIN;
}
if (stopMotors != 1) {
    if (diff < 0) {
        TPM0_C2V = ajustedSpeed - abs(diffCmd);
        TPM0_C5V = ajustedSpeed + abs(diffCmd);
    } else {
        TPM0_C2V = ajustedSpeed + abs(diffCmd);
        TPM0_C5V = ajustedSpeed - abs(diffCmd);
    }
}
}
```

Nous nous sommes aperçus que les roues arrière de la voiture dérapaient dans les tournants si elle roulait vite. Nous avons donc consulté des personnes avec plus de connaissances en mécanique et en fonctionnement des voitures que nous-mêmes, qui nous ont conseillé d'implémenter un **différentiel** entre les deux roues.

Dans les vraies voitures, les roues arrière ne roulent pas à la même vitesse dans un tournant. Dans un virage à gauche, la roue gauche roule moins vite et la roue droite roule plus vite. C'est donc ce que nous avons implémenté dans le code représenté ci-dessus.

La valeur affectée à `diffCmd` est proportionnelle à « l'angle » du servomoteur. Le facteur proportionnel `KV` a été fixé à 2/1000, et sachant que 1000 est la valeur maximale de `servoPosition`, `diffCmd` est compris entre -2 et 2. Nous avons observé que pour un différentiel trop important, la voiture tourne trop et il y a un risque qu'elle fasse un tour de 180 degrés.

Il y a également une condition avec une fonctionnalité analogue à celle vue dans la partie précédente pour assurer que la vitesse ne soit jamais inférieure à `MOTOR_SPEED_MIN`.

```
// Check if speed is to high before a curve, and cut the motors if true
if (cntHighSpeed >= 110 && (servoPosition > 250 || servoPosition < -250)) {
    cntHighSpeed = 0;
    stopMotors = 1;
}
if (stopMotors == 1) {
    LED_BLUE_OFF;
    LED_RED_ON;
    TPM0_C2V = 0; // Cut motor power
    TPM0_C5V = 0;
    if (cntMotorsOff >= 100) { // until
        cntMotorsOff = 0;
        stopMotors = 0;
        LED_BLUE_ON;
        LED_RED_OFF;
    } else {
        cntMotorsOff++;
    }
}
```

Enfin, notre dernière partie du bloc vitesse est intimement liée à notre **stratégie d'accélération** en ligne droite. En effet, si en sortie de telle sections rectilignes vient à se présenter un virage, même avec le correcteur implémenté, la voiture n'était pas capable de tenir la route. C'est pourquoi nous avons implémenté un système tout « bête » permettant de régler ce problème. Le compteur `cntHighSpeed` qui était incrémenté plus haut nous sert à repérer le temps depuis lequel la voiture est en phase de vitesse maximum. Si un virage va être abordé, c'est-à-dire si la valeur de `servoPosition` atteint des valeurs supérieures à 250 ou inférieure à -250, on remet le compteur à 0, et on attribue à la variable `stopMotors` la valeur `true` (1). Ainsi, on passe en **mode roue libre** durant un certain temps déterminé par l'incrémentation de `cntMotorsOff`. Les moteurs sont coupés, et la LED d'état passe à rouge pour que ce comportement soit bien visible en temps réel sur la piste.

La partie suivante sera donc consacrée aux divers essais et tests réalisés sur piste avant de participer à la compétition à Grenoble.

5. Tests et validations

5.1. Conditions de tests

Comme nous avons pu le détailler dans les parties précédentes, nous avons eu accès pour deux sessions d'entraînements à une piste officielle chez NXP. Cependant, le site de Toulouse à du la renvoyer début février pour la compétition, nous n'avons donc pas pu en profiter au moment où de vrais tests aurait été très utiles.

En effet, lors de ces deux séances notre voiture n'était pas prête à suivre la piste de manière autonome nous n'avons donc pas profité de cette expérience au maximum. Après ces deux après-midis durant lesquelles nous avons profité d'une piste officielle nous avons fabriqué **notre propre piste** afin de pouvoir nous entraîner sur l'INSA. Pour cela nous avons utilisé des plaques fines en polystyrène blanc avec des scotchs noirs sur les bords.

Nous avons donc fabriqué des lignes droites, des tournants ainsi qu'une piste avec des petites bosses. Nous nous sommes donc entraînés sur cette piste « faite maison » qui était beaucoup plus lisse que la piste réelle, ce qui faisait que la voiture sortait souvent de la piste pour des raisons autres que notre code. D'autant plus que pour pouvoir tester sur de plus grandes distances nous avons utilisé des paper-board peint en noir sur les bords comme piste.

Nos conditions de tests n'étaient donc **pas optimales** pour tester notre voiture et nous nous en sommes rendus compte le jour de la compétition. En effet la réflexion de la lumière sur les pistes n'est pas la même en fonction de la luminosité de la salle et du matériel.

Ces conditions de test nous ont permis de nous entraîner de manière efficace sur des situations facile et des parcours courts, cependant cela ne nous a pas permis de nous entraîner dans des conditions réelles et cela s'est ressenti le jour de la compétition.

5.2. Mise en place d'outils habiles

Afin de faciliter les réglages de la caméra nous avons imaginé deux techniques, une pour conserver l'**angle** parfait de la caméra, et l'autre pour conserver le **focus** parfait.

Dans un premier temps nous avons réglé la caméra à l'oscilloscope, lorsque qu'elle est bien réglée, nous pouvons observer le signal page suivante **figure 10** avec un morceau de piste en visuel.

Nous pouvons voir nettement deux pics qui représentent les bords de la piste. En bougeant les réglages de la caméra, on distinguait nettement les deux pics se déplacer, mais dans des positions qui signifiaient alors clairement que le contraste que voyait la caméra entre le sombre et le clair n'était pas vu de manière optimale.

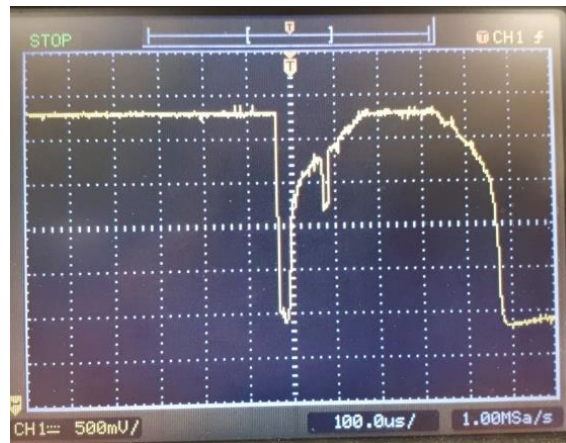


Figure 10 : Observation du signal reçu par la caméra

Afin de conserver ces réglages nous avons imaginé deux techniques. Pour conserver le focus nous avons simplement **rayé** légèrement la caméra au niveau de la vis de réglage. Il nous suffisait donc simplement pour régler le focus d'aligner les deux traits.

Ensuite pour conserver l'angle optimal nous avons découpé un **morceau de papier** afin que ce dernier corresponde exactement à l'angle entre le mat et la caméra. Nous pouvons ci-dessous **figure 11** par la suite une image de ce second outil :

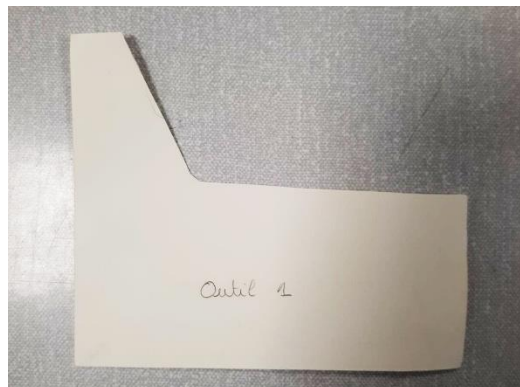
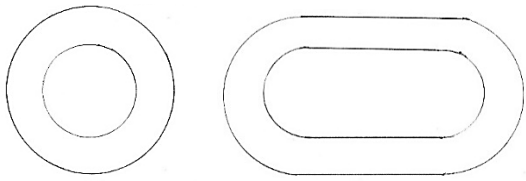
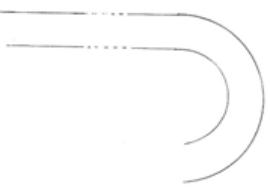

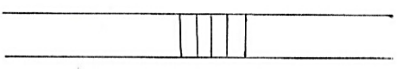
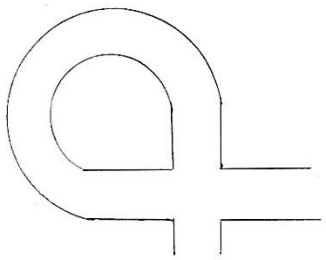


Figure 11 : Outil pour régler l'angle de la caméra

Ces deux outils nous ont rendus la vie beaucoup plus facile par la suite car nous n'avons plus besoin de régler la caméra à l'oscilloscope entre chaque essai ou après chaque **choc**.

5.3. Situations testées

Les situations testées ainsi que les résultats sont présentés dans le tableau suivant :

Situation	Observations
	Notre voiture gère parfaitement cette situation. Cela lui arrive au bout d'un certain nombre de tour de sortir de la piste mais cela est dû au fait que notre piste est beaucoup plus lisse que la piste officielle.
	Notre voiture gère parfaitement cette situation depuis que nous avons implémenté le ralentissement suite à la détection d'un virage.
	Notre voiture gère parfaitement cette situation.
	Notre voiture gère parfaitement cette situation.
	Malheureusement notre voiture ne gère pas toujours bien cette situation, cela dépend de l'angle d'arrivée de la voiture au niveau de l'entrée du croisement. Nous n'avons pas eu le temps d'améliorer notre programme mais nous avons commencé à réfléchir à des solutions possibles.

6. Zoom sur la compétition

Le 14 et 15 mars derniers, nous avons donc rejoins Grenoble avec les autres équipes afin de **concrétiser** ce projet. Nous savions qu'à notre arrivée là-bas, il y aurait bien entendu des réglages de dernières minutes çà faire, mais avons dû faire face à bien « pire » que cela...

6.1. *Organisation et tests préliminaires*

Le premier jour était consacré aux **essais** sur diverses portions de circuits mises à notre disposition. Nous avons vite remarqué que l'environnement était très différent de celui dans lequel nous nous étions entraînés à l'INSA. La luminosité globale de la salle était différente, et les bouts de piste de bien meilleure qualité. Nous avons donc réalisé quelques correctifs sur nos valeurs de seuils et correcteurs pour nous adapter.

De plus, nous avons dû faire face à gros un problème technique concernant le connecteur de caméra, avec la **rupture** d'un fil qui a nécessité plusieurs soudures et le remplacement global d'un des composants. Heureusement que le laboratoire de l'école accueillant la compétition était à notre disposition pour pouvoir faire toutes nos soudures correctrices. De même qu'à l'INSA, un intervenant ayant les compétences requises en électronique et en soudure comme M. Martin a pu nous encadrer.

Ce souci nous a quand même beaucoup **retardé**, car nous espérions profiter au maximum de la piste pour nous entraîner et faire les ajustements nécessaires, mais nous avons passés beaucoup de temps à réparer des bugs d'ordre matériel plutôt que logiciels.

6.2. *Choix final de la stratégie adoptée et résultat*

La voiture avait au final un comportement correct en fin de journée, et avait réussi à boucler les trois petits circuits tests disponibles, même si le comportement de notre voiture variait en fonction des essais pour le circuit comprenant un carrefour.

Nous avons alors décidé d'adopter une stratégie **un peu moins osée** que celle que nous avions prévue au vu des différents essais réalisés par les autres équipes participant à la compétition. Nous avons en effet décider de réduire légèrement la valeur de notre vitesse maximum pour assurer un peu plus nos chances de qualifications.

Le déroulement de la compétition pour le modèle Alamak (nouveau modèle) a alors été quelque peu surprenant. En effet, **aucune des équipes en lice n'est parvenue à compléter un tour entier**. Notre voiture a été la plus rapide concernant les portions réalisées, mais un des virages, placé en sortie de bosse et enchaînant sur un second très léger virage, a mis la moitié des équipes hors-piste. Nous étions assez déçus car notre voiture avait un vrai potentiel de qualification. La compétition « modèle Alamak » contrastait alors vraiment avec celle concernant les anciens modèles de voiture, ces équipes-là ayant alors des voitures très performantes. Cependant il était assez normal d'observer cette situation, ces équipes travaillant sur leur modèle de voiture depuis de nombreuses années.

Conclusion

Pour revenir sur cette expérience et ces résultats, nous trouvons qu'il est dommage qu'aucune des équipes n'ait pu se qualifier (notamment sur les trois équipes engagées par l'INSA), mais cela reste néanmoins une bonne expérience, sur un projet qui a su mettre en évidence nos compétences en **périphériques, automatique et codage informatique**.

De plus, des **optimisations** et améliorations auraient été possibles si nous avions passé moins de temps à régler tous les problèmes liés au nouveau modèle de voiture. En effet, les boutons disponibles sur la carte auraient pu servir à mémoriser des « modes » de course en fonction de leur sélection pour pouvoir ajuster la stratégie entre les essais, ce qui aurait permis par exemple de changer de vitesse maximum, ou de seuil de luminosité si la piste était un peu plus éclairée le jour de la compétition. De même, les bouts de piste pour s'entraîner à l'INSA étaient très **limités** et peu identiques à ceux officiellement utilisés le jour de la compétition, ce qui limitait nos tests à Toulouse.

Enfin, ce projet tutoré a aussi mis en évidence les compétences de **conduite de projet** acquises au fur et à mesure de notre cursus à l'INSA, et notamment dans l'UF que nous avons pu suivre ce semestre. Nous avons dû suivre un planning assez restreint en termes de marge entre date de début et de fin des différentes tâches effectuées, ainsi que de deadline pour le jour de la compétition.

En conclusion, ce projet tutoré reste une très bonne expérience, ayant demandée de nombreuses compétences et le besoin pour chacun d'entre nous d'avoir été très **polyvalent** dans les diverses phases d'avancement qu'il a suscité.



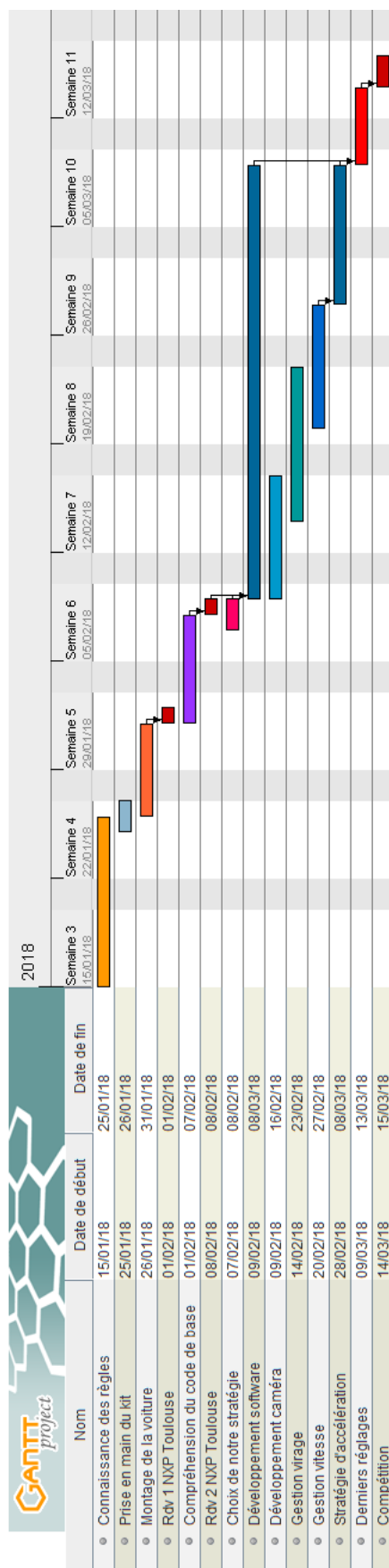
Annexes

Annexe A1 : *Diagramme de Gantt* i

Annexe A2 : *Glossaire* ii

Annexe A3 : *NXP Cup 2018* iii

Annexe A1 : Diagramme de Gantt



Annexe A2 : Glossaire

Hardware : désigne le matériel physique constituant la voiture.

Software : désigne le logiciel, le code embarqué sur le microcontrôleur.

Capteur : Dispositif permettant de capter un phénomène physique et de le restituer sous forme de signal.

Débugger : éliminer les anomalies de fonctionnement d'un programme.

Connecteurs : dispositif qui connecte un capteur avec la carte d'entrée/sortie.

PWM : Pulse Width Modulation ou Modulation de Largeur d'Impulsion (MLI) en français est une technique qui permet d'obtenir en sortie n'importe quelle valeur de tension entre 0 et la valeur d'alimentation de la carte.

Servomoteur : système motorisé capable d'atteindre des positions déterminées puis de les maintenir.

Librairies : programme informatique qui contient un ensemble de fonctions utilisables par d'autres programmes.

Périphérique : terme générique donné aux composants de matériel informatique assurant les communications entre le microcontrôleur et les capteurs.

Interruption : arrêt temporaire de l'exécution normale d'un programme informatique par le microcontrôleur afin d'exécuter un autre programme.

Annexe A3 :

NXP Cup 2018

Compétition de voiture autonome sur piste

RESUME :

Ce rapport est le résultat d'un projet tutoré qui fait partie de la formation d'ingénieur en **4^{ème} année à l'INSA de Toulouse**. Les auteurs du rapport ont choisi de participer à la compétition internationale NXP Cup, organisé par **NXP Semiconductors**. Des groupes de trois étudiants reçoivent une petite voiture en kit à assembler. Le but est de la rendre autonome et capable de compléter un tour sur une piste blanche tout en restant entre deux lignes noires, identifiées par une caméra de type line scan. La forme du circuit reste inconnue aux participant jusqu'à la course. Le projet demande aux étudiants de faire de la programmation périphérique, de l'algorithmique et de l'automatique. Il est aussi indispensable d'être créatif et de réaliser un projet sur un temps limité. Ce rapport détaille les choix techniques et organisationnels du groupe, ainsi que des réflexions et des pistes d'améliorations.

MOTS-CLES : *NXP Cup, voiture autonome, automatique, projet tutoré, programmation, informatique industrielle, ingénierie, système embarqués*

ABSTRACT :

This report is the result of a mentored project that is a part of the **4th year curriculum at INSA Toulouse**. The authors of this paper chose to participate in an international competition called the NXP Cup, organized by **NXP Semiconductors**. Groups of three students receive a kit to assemble into a small autonomous car. The aim is for the car to complete a tour around a white track and stay between two black lines, identified by a line scan camera. The shape and the particularities of the track is unknown to the participants until the race begins. The project demands peripheral programming, algorithmic and automation skills, as well as creativity and cooperation, over a short amount of time. This paper details the group's technical choices and organization.

KEYWORDS : *NXP Cup, autonomous car, automation, mentored project, programming, informatics, engineering, embedded systems*