Jospeh-Marie Valleix

Projet Pluridisciplinaire Encadré 2010

Louis Le Grand TS5

**Robot suiveur de ligne à caméra embarquée**

*Cam & Léon*

Ce PPE est le premier grand projet effectué en Science de l’Ingénieur après celui des TPE de première. Mon TPE portait sur un savant de la Renaissance dont nous avions étudié les travaux : le joint de Cardan et sa méthode de résolution d’équations. Ces études très théoriques étaient suivies d’une série de programmes informatiques résolvant les équations en x³.

La conception d’un robot était donc une expérience totalement nouvelle pour moi lorsque nous commencions notre PPE en janvier. Les problèmes auxquels nous avons été confrontés étaient le plus souvent d’ordre pratique, la théorie n’exerçant son joug que lors de la phase de conception.

Étant donné mon expérience en programmation acquise lors du TPE, je fus immédiatement préposé à la programmation de notre robot. Elle fut extrêmement intéressante car je n’avais jamais eut l’occasion de travailler sur un système embarqué.

La réalisation de notre robot, baptisé Cam & Léon, s’est bien passée, même si nous avons eu à faire face à quelques évènements imprévus au début du projet…

La « CmuCam Original »

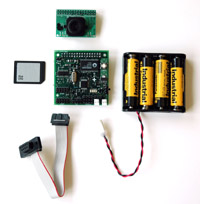
Nous avons commencé notre PPE en utilisant une « CmuCam Original » proposée par notre professeur de génie électrique. Ce système est composé d’un microcontrôleur SX28, d’une caméra numérique enregistrant 17 images par secondes. La conception de cette caméra remonte à 10 ans et son utilisation dans un environnement de programmation récent est pour le moins délicate : nous avons du compiler en ligne de commande le programme permettant de communiquer par le port série avec la carte…

Cette carte ne dispose que d’un unique port pour servomoteur, or pour diriger notre robot, nous en avions besoin de deux, il nous fallait donc utiliser une carte externe. Nous voulions utiliser le microprocesseur SX28 de la CmuCam pour analyser l’image, Anna a donc envoyé un e-mail à l’université de Carnegie Mellon qui a conçu la carte, leur demandant le code source du programme d’origine pour que nous puissions l’adapter à nos besoins. Le code source en questions est long de 1691 lignes de code, non commentées : autant dire qu’il me fut totalement incompréhensible… De plus, pour charger un nouveau programme sur la carte, nous devions acheter une clef spéciale (50$) se connectant sur le SX28 pour écraser sa ROM…

Nous avons donc abandonné l’idée de rechercher nous-mêmes la ligne et pensions alors utiliser le programme d’origine de la carte qui en est déjà capable. Notre travail se serait donc limité à gérer la communication entre la CmuCam et une seconde carte.

Nous avons alors utilisé le Basic Stamp, une carte de développement programmable en Basic. Son interpréteur est hélas très lent : 2000 instructions basic/s maximum. Or la vitesse minimum de connexion par le port série de la CmuCam est 9600 bauds : nous n’avons jamais réussi à faire communiquer les deux cartes. Nous avons donc choisi d’utiliser un PIC 16F877A qui a lui la capacité de communiquer avec la CmuCam. La gestion de l’UART fut assez difficile, mais la veille des vacances de février nous avons enfin réussi à faire passer quelques milliers de bits entre les deux cartes…

La CmuCam3

**Première approche**

Au retour des vacances, nous avons eut la belle surprise de trouver une CmuCam troisième génération, nettement plus récente : conçue en 2007. Elle dispose de 4 ports pour servomoteur, est entièrement programmable en C et l’enregistrement du programme sur la ROM se fait directement par le port série. Nous n’avions plus besoin d’une seconde carte.

J’ai d’abord essayé de faire simplement tourner un servomoteur. Il existe pour cela un programme de test que nous devions compiler pour l’enregistrer sur la carte. La compilation s’effectue grâce à Cygwin, un compilateur hexadécimal issu de Linux, qui dispose d’un module optimisé pour les processeurs d’architecture ARM comme le LPC2106 de la CmuCam3.

Cygwin se présente comme un terminal, la commande de compilation est « make -f Makefile ». Il m’a fallut 3 heures pour trouver dans l’obscure documentation l’option « –f » sans laquelle il ne compile pas le code…

L’envoi du fichier hexadécimal dans la ROM se fait par l’intermédiaire d’un programme développé par Philips qui se charge de la communication ordinateur/série/microcontrôleur.

**L’Application Programming Interface (API) de la Carnegie Mellon**

L’université américaine qui a créé la carte a également codé en C un ensemble de fonctions spéciales qui aident la programmation. Leur code source n’est pas accessible mais un rapide aperçu de chaque fonction est disponible dans le fichier contenant leurs « headers ». Ces fonctions permettent par exemple d’initialiser la caméra, de faire tourner un servomoteur, de lire un pixel *etc…*

Pour m’habituer au style de programmation du microcontrôleur, j’ai lu et analysé quelques programmes de test fournis avec la caméra. Le premier programme que j’ai codé nous a permis de mesurer la vitesse des roues en fonctions des paramètres que l’on envoie à la fonction de l’API « cc3\_gpio\_set\_servo\_position() ; ». Ce programme est assez complet : il utilise le bouton de la carte et le port série pour communiquer avec l’ordinateur. Il est en effet possible de communiquer avec la caméra pendant l’exécution d’un programme grâce au terminal CmuCam Frame Grabber.

**Le traitement de l’image**

Parallèlement à l’évolution de la programmation, Anna et Tramy dessinaient le robot sous SolidWorks. Vivien (lorsqu’il ne codait pas) et Romain construisaient le robot. Cette répartition efficace des tâches nous a permis d’obtenir assez rapidement un robot fonctionnel, prêt pour des tests.

J’ai pensé à plusieurs solutions pour détecter la ligne : nous aurions pu soit calculer les variations de couleur d’une ligne de pixel, soit mesurer l’intensité lumineuse de chaque pixel de cette ligne. Nous avons choisi la seconde solution car elle est la plus rapide et simple. Une ligne noire sur un fond blanc possède de forts contrastes et minimise par conséquent les risques d’erreurs dans la mesure de l’intensité lumineuse des pixels.

Le programme compare donc l’intensité de chaque pixel d’une ligne de l’image et enregistre l’abscisse du pixel le plus sombre. Pour réduire encore les risques d’erreurs, nous analysons 10 lignes de l’image. Nous aurons pu analyser les 88 lignes de pixels mais cela ralentissait trop le programme. Pour obtenir la position de la ligne, le programme calcule la moyenne de l’abscisse des pixels sombres.

Pour obtenir la valeur de l’intensité d’un pixel, le programme lit le canal « Value » de ce pixel. L’image est encodée en « Hue Saturation Value » (HSV), cela signifie qu’à chaque pixel sont associé un canal H, S et V. Chacun des canaux est en fait un entier codé sur 8 bits. Avec trois canaux de 8 bits, on donc peut encoder 255³ soit 16 777 216 couleurs différentes. Plus la valeur de Value est basse, plus le pixel est sombre.

Notre programme dispose d’un ensemble de directives agissant sur le préprocesseur avant la compilation, et permettant de modifier les valeurs des constantes (de vitesse, d’exposition, information affichées sur le terminal *etc*…) sans modifier profondément le code.

**L’absence de ligne**

Pour détecter l’absence de ligne, j’ai d’abord pensé à calculer la moyenne de l’intensité de chaque pixel de l’image, que le robot comparerait à une valeur étalonnée lors du démarrage du programme. Bien que théoriquement intéressante, cette solution ne fonctionne pas.

J’ai alors pensé à calculer la cohérence des point sombres obtenus, c’est à dire calculer l’écart type de la série de pixels sombres obtenus et déterminé la présence de la ligne en fonction de sa valeur. Cette méthode n’est pas non plus très performante.

La manière la plus efficace et de définir empiriquement une valeur minimale de l’intensité nécessaire pour que le pixel soit considéré comme sombre. Si après l’analyse des 10 lignes, il n’y a pas assez de pixels sombres détectés, le programme interprète cela comme une absence de ligne.

**La gestion des moteurs**

Il nous est paru tout de suite évident que la vitesse des roues devait dépendre du désaxement du robot par rapport à la ligne. La difficulté de la gestion des moteurs réside dans le choix des constants utilisées pour définir la vitesse relative des roues. L’équilibre entre les deux roues fut long à trouver, et il n’est pas encore parfait. Il existe également un mode de recherche de ligne dans lequel le robot effectue une rotation sur lui même.

**Retour sur la programmation**

Actuellement, une exécution complète du programme demande 19 ms. La caméra enregistrant 50 images par secondes, nous avons atteins les limites technologiques de la carte. Pour y arriver, nous avons réduit drastiquement le nombre d’instructions du programme qui se limite à seulement 250 lignes de code, API non comprise. Une fois compilé, le programme contient 121 738 digits en hexadécimal…

Avant nos phases d’optimisation, le programmes durait 121 ms, soit plus de 600% du temps actuellement nécessaire !

Notre programme possède une « architecture modulaire » : le fichier « main.c » ne contient que la fonction « main() », il y a un fichier de headers (cc3\_func.h) et un autre fichier (cc3\_func.c) contenant nos fonctions InitAll(), ImAnalysis() et MotComm() qui se chargent d’initialiser la carte CamCaum3, d’analyser l’image et de commander les servomoteurs.

**Evolution du robot**

Il serait bien d’utiliser dans le future le lecteur de carte SD intégré à la carte CmuCam3, par exemple pour enregistrer sur dans image le parcours effectué. Pour cela il faudrait gérer des interruptions en C, ce qui est loin d’être évident… De plus, la puissance du LPC2106 n’est pas illimité est le suivi de ligne ne doit pas être ralentit par des fonctions facultatives.

Nous pourrions également utiliser les ports GPIO secondaire pour connecter un buzzer ou un capteur ultrason, ce qui ajouterait des fonctionnalités utiles au robot.

Le hardware

Ma contribution en ce qui concerne le « hardware » est limitée conformément à ce que l’on avait prévu lors de la distribution des tâches. J’ai néanmoins participé à la conception de l’architecture du robot, notamment en ce qui concerne l’optimisation de sa taille (par exemple en plaçant la plaque de PVC entre les servomoteurs et le support de la caméra).

J’ai également créé le « design » de la présentation Power Point, sachant qu’une importante partie du contenue à été reprise et adaptée d’une version précédente rédigée par Romain. Je le remercie d’ailleurs d’avoir prêté un espace de son serveur FTP au groupe, l’utilisation de cette plateforme web fut très pratique et facile. J’avais déjà l’habitude d’utiliser ce type de stockage « on the cloud » étant donné que je possède moi même un site web.

robot2.jpgCam & Léon pourrait être commercialisé, il est solide et fonctionne bien. Ce projet aura été mené jusqu’au bout assez régulièrement, il aura été très instructif, non seulement technologiquement mais également humainement. Il illustre bien j’espère, ce que nous réserve les sciences de l’ingénieurs dans le future.