



Livrable 1 - Robot Kinocto

Machines Électriques

présenté à

M. Dominique Grenier, M. Luc Lamontagne et M. Abdelhakim Bendada

<i>matricule</i>	<i>nom</i>
XXX XXX XXX	Émile Arsenault
908 190 985	Philippe Bourdages
910 098 468	Pierre-Luc Buhler
998 107 355	Diane Fournier
908 159 170	Imane Mouhtij
908 318 388	Olivier Sylvain
XXX XXX XXX	Daniel Thibodeau
XXX XXX XXX	Francis Valois

Université Laval
11 février 2012

Chapitre 1

Introduction

Chapitre 2

Diagramme de contexte

Chapitre 3

Description des propriétés fonctionnelles

Chapitre 4

Description des cas d'utilisation

Cette section contient un résumé de chaque cas d'utilisation. De plus, elle est divisée en trois sous-sections : les cas d'utilisation en lien à l'utilisateur, à la station de base et au robot. Voir le diagramme des cas d'utilisation en annexe A.1.

4.1 Cas d'utilisation en lien avec l'utilisateur

4.1.1 Lancer le signal de départ

À l'aide d'un GUI installé sur la station de base, l'utilisateur clique sur un bouton qui permet au robot de lancer son exécution (signal de départ transmis au robot pour commencer à effectuer une tâche).

4.1.2 Entrer les coordonnées initiales du robot

Avant d'envoyer le signal de départ au robot, l'utilisateur doit pouvoir entrer dans le GUI de la station de base les coordonnées de la position de départ du robot par rapport au terrain de jeu.

4.2 Cas d'utilisation en lien avec la station de base

4.2.1 Localiser le robot

À l'aide de la Kinect, la station de base doit être capable de trouver la position du robot sur le terrain de jeu.

4.2.2 Détecter les obstacles

La station de base, à l'aide de la Kinect, doit être en mesure de détecter la position des deux obstacles disposés entre les deux zones principales du terrain de jeu, ainsi que les limites (murs) du terrain.

4.2.3 Afficher la solution du sudocube

La station de base doit être capable d'afficher une image du sudocube résolu et indiquer à l'écran le chiffre qui se trouve dans la case rouge du sudocube.

4.2.4 Afficher le message de confirmation du lancement de la tâche

La station doit être capable d'afficher un message de confirmation du lancement de la tâche pour l'utilisateur lorsque le message de confirmation du robot est reçu.

4.2.5 Afficher la trajectoire prévue et réelle du robot

La trajectoire calculée et transmise par le robot doit être affichée à l'écran de la station de base. De plus, à l'aide de données fréquemment transmises par la Kinect contenant la position du robot, la station doit afficher, en comparaison à la trajectoire prévue, la trajectoire réelle du robot.

4.2.6 Transmettre des données au robot

La station doit transmettre au robot, par connexion sans fil, des messages indiquant de lancer une nouvelle tâche, la position des obstacles et la position de ce dernier.

4.2.7 Recevoir des données provenant du robot

La station de base doit être capable de recevoir par connexion sans fil des données provenant du robot comme la trajectoire que le robot prévoit emprunter, le sudocube résolu, le message de confirmation du lancement d'une tâche et le message indiquant la fin d'une tâche.

4.2.8 Afficher le message confirmant que la tâche a été complétée avec succès ou a échoué

La station de base doit afficher un message à l'écran confirmant que la tâche a été complétée avec succès ou un message d'échec si cette dernière a pris de plus de 10 minutes pour s'exécuter.

4.2.9 Afficher le temps d'exécution de la tâche

La station de base doit être en mesure d'afficher à l'écran le temps d'exécution de la tâche.

4.3 Cas d'utilisation en lien avec le robot

4.3.1 Déterminer le chemin optimal entre un point A et un point B tout en évitant les obstacles

En connaissant la configuration et la position des obstacles, le robot doit calculer la trajectoire optimale qu'il doit emprunter pour se rendre d'un point à un autre.

4.3.2 Se déplacer d'un point A au point B

Le robot doit être capable de se déplacer, en contrôlant ces roues, pour partir d'un A et atteindre un point B sur le terrain de jeu.

4.3.3 Indiquer son adresse IP locale pour que la station de base puisse communiquer avec le robot

Le robot doit être capable d'indiquer son adresse IP locale à la station de base pour que celle-ci établisse une connexion sans fil avec le robot.

4.3.4 Extraire les informations d'un sudocube à partir d'une photo

À l'aide d'une webcam, le robot doit être capable de prendre une photo et modéliser le sudocube à partir de cette photo.

4.3.5 Résoudre le sudocube et identifier le chiffre dans le carré rouge

Le robot doit effectuer des traitements sur l'image du sudocube pour le modéliser et ensuite le résoudre. Il identifie également l'emplacement de la case rouge à l'intérieur du sudocube et donc trouve le chiffre qui s'y trouve.

4.3.6 Recevoir des données provenant de la station de base

Le robot doit être capable de recevoir, par connexion sans fil avec la station de base, des messages indiquant de lancer une nouvelle tâche, la position des obstacles et sa position.

4.3.7 Transmettre des données à la station de base

Le robot doit transmettre par connexion sans-fil des données à la station de base comme la trajectoire que le robot prévoit emprunter, le sudocube résolu, le message de confirmation du lancement d'une tâche et le message indiquant la fin d'une tâche.

4.3.8 Contrôler une LED

Une fois le message de fin de tâche transmis et le robot à l'extérieur de l'air de dessin, ce dernier doit allumer une LED qui est installée sur le robot.

4.3.9 Lancer de nouvelles tâches si la compétition n'est pas terminée

Si le temps alloué pour la compétition (10 minutes) n'est pas écoulé, le robot doit commencer une nouvelle tâche après en avoir fini une.

4.3.10 Se localiser

Le robot doit en tout temps être capable de déterminer sa position et son orientation, à l'aide de la webcam et des données reçues, ce qu'il lui permet de connaître son emplacement par rapport à l'antenne, aux murs, aux obstacles et aux sudocubes. De plus, il doit transmettre ces données à la station de base de façon régulière.

4.3.11 Décoder la transmission de l'antenne

Le robot doit se déplacer au-dessus de l'antenne et capter le signal transmis par celle-ci et le décoder pour trouver les paramètres indiquant le sudocube ciblé et l'orientation et la taille du chiffre à dessiner.

4.3.12 Dessiner, en suivant les consignes, le chiffre prélevé dans le carré rouge

Le robot doit descendre le préhenseur du crayon une fois dans l'air de dessin et se déplacer de façon à dessiner le chiffre trouvé dans la case rouge du sudocube tout en suivant les dimensions exigées et les paramètres décodés.

4.3.13 Afficher des informations sur l'écran LCD

Les paramètres trouvés lors du décodage du signal de l'antenne doivent être affichés sur un écran LCD installé sur le robot.

Chapitre 5

Diagrammes de séquences

Chapitre 6

Diagrammes de classes

Chapitre 7

Description des prototypes

7.1 Préhenseur

Comme dispositif pour le traçage des dessins par le robot, nous opterons pour la solution la plus simple et robuste qui soit. Afin d'éviter d'augmenter inutilement la charge de travail du microcontrôleur, nous avons décidé d'éviter d'utiliser des solutions complexes qui pourraient exiger la production d'un signal particulier par le microcontrôleur. Le signal de sortie que devra produire le celui-ci sera simplement un signal binaire : « 1 » lorsque le crayon devra être en mode traçage et « 0 » lorsque le crayon sera soulevé et donc en mode « attente ». Pour maintenir le crayon en position « attente » durant la durée du trajet, nous utiliserons un système de maintien magnétique qui gardera le crayon en position relevé lorsqu'il recevra le signal « 1 » logique. Pour abaisser le crayon et ainsi entrer en mode traçage, nous utiliserons un ressort à faible constante de rappel qui entrera en jeu lorsque le système de maintien magnétique relâchera le crayon « 0 » logique. Il est bien important que la constante de rappel du ressort soit faible pour ne pas briser la mine lors du contact de celle-ci avec la surface de traçage. La constante de rappelle du ressort devra également être inférieur à la force exercée par le système de maintien magnétique qui ramènera le crayon en mode attente une fois le dessin terminé.

7.2 Récepteur et décodeur du signal manchester

Pour le récepteur du signal magnétique, nous utiliserons un « tone decoder » qui est utilisé dans de nombreuses applications qui nécessitent la reconnaissance de certaines tonalités bien précises. Comme nous avons besoins de détecter une série tonalités identiques qui se suivent et forme un code, ce genre de dispositif nous convient parfaitement. Le « tone decoder » recevra le signal par une boucle de fil installées à un endroit sur le robot qui captera la variation du flux magnétique produite par le fil sous la table et détectera si la tonalité inaudible est émise ou non. Lorsque la fréquence (ou tonalité) sera reçue par le récepteur, celui-ci présentera zéro volt en sortie et à l'opposé, lorsque le récepteur ne détectera pas la fréquence ciblée, il présentera 5V à sa sortie. Il exécutera ces opérations d'une façon suffisamment rapide pour que le segment binaire reçu soit exactement le même que celui transmis, mais complètement opposé étant donné la configuration logique du « tone decoder ». Ce message sera ensuite inversé par l'unité de traitement suivante et traité pour que le robot exécute la tâche adéquate. L'avantage de cette approche est que le « tone decoder » est un circuit intégré très robuste aux perturbations magnétiques de son environnement, il est peu énergivore et très compact.

7.3 Microcontrôleur

Le lien entre le pont en H contrôlant les moteurs, les encodeurs sur les moteurs, le signal décodé par le récepteur du signal d'antenne, le préhenseur, la LED, l'écran LCD et le Mac mini sera un microcontrôleur modèle Texas Instrument Stellaris LM3S9B92. Ce microcontrôleur possède un grand nombre de broches d'entrée/sortie, 4 PWM, 2 interfaces d'encodeur à quadrature qui permettront le contrôle et l'asservissement des moteurs. Il possède également deux ports de communication UART ainsi qu'une interface USB pour la communication avec le Mac mini. Le microcontrôleur peut être programmé en C avec un compilateur, un IDE et des bibliothèques de pilotes de périphériques fournies.

7.4 Affichage sur le robot

Pour l'affichage sur le robot, nous utiliserons un écran à cristaux liquides 16 x 2 caractères. Cet écran est commandé par 8 lignes de données qui permettent de transmettre des caractères ASCII et 3 lignes de contrôle, RS, R/W et E. Le huitième bit de données peut également servir de "busy flag", ce qui permet de savoir si le contrôleur de l'écran a terminé d'exécuter les dernières instructions. Il est possible d'ajouter des caractères "maison" dans des espaces libres de la mémoire de caractères. Le contraste peut être contrôlé par un potentiomètre qui fournit la référence sur la broche 3. Il peut être alimenté à même le microcontrôleur, en excluant le rétroéclairage, qui n'est pas nécessaire pour distinguer les caractères dans un environnement éclairé. Si le rétroéclairage devait être utilisé, il devra être alimenté sur une source indépendante du microcontrôleur, car il nécessite 150 mA. Les fonctions codées dans les fichiers `ecran.h` et `ecran.c` de l'annexe A.2 permettent d'envoyer des caractères et des chaînes de caractères sur le LCD et de contrôler le déplacement du curseur.

7.5 Communication entre le microcontrôleur et le Mac mini

Comme le microcontrôleur possède une interface USB qui permet la communication avec l'ordinateur via un port série UART, nous utiliserons le port UART pour la communication. Le port UART lève une interruption lorsqu'il reçoit un caractère dans son registre de données. Les données reçues peuvent être enregistrées dans un tampon circulaire durant l'interruption pour être traitées dans le processus principal. Du côté du Mac mini, un terminal série permettra d'envoyer des données via l'interface USB.

7.6 Alimentation du Mac mini

En ce qui a trait au système d'alimentation du Mac mini, le dispositif à concevoir doit réussir à élever la tension de la batterie jusqu'à 24V. Cette tension correspond à un point

de fonctionnement qui est jugé comme idéal selon les spécifications techniques de l'appareil. On constate par la suite que la puissance demandée par le Mac mini sera la plus grande dépense énergétique du système. Il faut donc envisager une alimentation avec un très bon rendement, de manière à limiter le dimensionnement de la batterie. L'usage d'un régulateur de tension conventionnel, qui utilise une tension supérieure à l'alimentation, n'est pas envisageable pour le projet, dans la mesure où le rendement dépasse rarement les 50%. Ce qui s'offre à la conception de l'alimentation est un système de type «Boost» qui effectue une amplification de la tension continue. Ce type de circuit est le pendant en tension continue du transformateur. En soi, il est possible de réaliser la fonction d'amplification au moyen de composants discrets. Cependant, l'implantation pratique demande beaucoup de temps et de ressources et est généralement moins robuste qu'une alimentation utilisant des composants intégrés. Il est préférable, vu les coûts de l'électronique actuelle, d'opter pour des composants intégrés qui effectuent le même travail avec des rendements très élevés (de l'ordre du 90% et plus). Aussi, vu le coût de l'ordinateur et sa sensibilité aux oscillations de tension, il est très important de considérer la stabilité de la tension de sortie de l'alimentation. Les régulateurs envisageables pour l'application et les spécifications en courant présentent une ondulation de tension inférieure aux niveaux maximaux du Mac mini ($\pm 200mV$). Il est donc tout indiqué d'opter pour un circuit intégré de type «Boost», vu son bon rendement, sa robustesse, sa facilité d'implantation ainsi que sa stabilité de tension de sortie.

7.7 Alimentation de l'électronique embarquée

En ce qui a trait au système d'alimentation de l'électronique embarquée, l'alimentation de l'électronique autre que le Mac mini doit se faire optimalement en 5V, puisqu'il s'agit d'une tension utilisable pour le servomoteur de la tourelle de caméra et permet aussi de donner un point de référence au pont en H. Par ailleurs, la plupart des microcontrôleurs ont des entrées 5V, il est donc de mise d'utiliser cette tension pour l'électronique auxiliaire. Le dispositif devra abaisser la tension de la batterie à 5V puisque celle-ci sera sélectionnée avec une tension supérieure. On peut tout de suite songer à l'usage d'un régulateur, cependant, pour les mêmes raisons qu'énoncées à la section 7.6, un faible rendement de l'alimentation pourrait dégrader la durée de vie de la batterie. Vu qu'il est nécessaire d'abaisser la tension, l'utilisation d'une technologie de type «Boost» est impossible. Il existe cependant la technologie de type «Buck», qui produit exactement l'effet contraire. L'utilisation de composés discrets requiert davantage de temps et est moins robuste. Une solution impliquant des composés intégrés doit être envisagée. Par ailleurs, les rendements de ce type d'alimentation peuvent s'élever à plus de 90%. Les composantes comme le microcontrôleur et le pont en H requièrent une tension d'alimentation stable et l'emploi d'une alimentation faite à partir de composants intégrés permet de convenir à ce besoin. Il est donc tout indiqué d'opter pour un circuit intégré de type «Buck», vu son bon rendement, sa robustesse, sa facilité d'implantation ainsi que sa stabilité de tension de sortie.

7.8 Communication sans fil

Pour initier la communication sans fil entre le robot et la station de base, il faut que cette dernière connaisse l'adresse IP du robot. Il nous est impossible de brancher un clavier et un écran sur le robot afin de récupérer celle-ci et le réseau sans fil de l'université empêche l'utilisation de service tel que `www.NOIP.com` pour obtenir un DNS. Nous avons donc créé un script bash pour récupérer l'adresse IP du robot et l'envoyer à une application en ligne appelée `pastebin` qui permet d'enregistrer du texte. Le script est placé dans le fichier `/etc/rc.d/rc.local` afin qu'à chaque démarrage l'adresse IP soit connue. Nous avons validé l'adresse IP obtenue en l'utilisant pour se connecter par ssh sur le robot. Confirmant ainsi la possibilité d'utiliser le réseau sans fil pour communiquer avec le robot.

Ultérieurement nous utiliserons une api en c++ ou en python pour communiquer.

7.9 Calcul et affichage de la trajectoire

Afin de trouver une trajectoire correcte, rapide et qui évite tous les obstacles, il faut trouver le chemin le plus court entre deux nœuds et évaluer si un chemin existe entre ces deux derniers. Pour atteindre ces objectifs, on va utiliser le path-planning avec l'algorithme A* qui est simple et répond à tous nos besoins.

Pour l'affichage de la trajectoire en temps réel, le robot transmet sa position(x,y) sous forme de message à la station de base (au minimum toutes les 15 secondes entre chaque position) en utilisant la connexion sans fil.

7.10 Lecture des informations d'un sudocube avec la caméra

Pour réaliser ce prototype, OpenCV fut utilisé ainsi que quelques photos des sudocubes de l'environnement de travail du robot. Le langage de programmation C++ fut employé afin de garantir un temps de traitement relativement faible. Les opérations sur les images exigeant beaucoup de cycles processeur. De plus, l'accès à un plus grand nombre d'exemples de code OpenCV en C++(versus en python) a aussi motivé le choix de ce langage.

L'algorithme testé dans ce prototype commence par effectuer une segmentation par couleur verte sur la photo afin d'isoler le cadre du sudocube. Puis, un algorithme de détection des contours est appliqué. OpenCV retourne après cette opération deux polygones rectangulaires que l'on peut utiliser pour vérifier l'existence du cadre vert dans l'image. On s'assure que le cadre remplit une bonne partie de l'image en calculant l'aire de celui-ci. Cela permet de garantir qu'il y a assez de détails pour lire correctement les cases du sudocube. Puis, l'image originale est convertie en tons de gris pour appliquer encore une fois un algorithme de détection des contours. Cela permet de récupérer les polygones de toutes les cases du sudocube. Il y a quelques polygones qui ne sont pas des cases. On les élimine en calculant leur aire et en ne gardant que ceux qui sont suffisamment grands.

Afin de garantir la lecture de toutes les cases du sudocube, on pourrait ajouter une détection automatique du “threshold” de l’algorithme de détection des contours en vérifiant le nombre de polygones trouvés. Il faut en trouver 47 très exactement.

Pour assurer une bonne lecture des caractères, on pourrait effectuer un réaligement du cadre vert.

Ce qui reste à faire : effectuer une segmentation sur la couleur rouge en mode HSV afin d’identifier la case rouge et créer un algorithme d’OCR pour lire les chiffres dans les cases.

7.11 Recherche de chemin

Afin de résoudre le problème de recherche d’un chemin à parcourir par kinecto afin d’éviter les obstacles, nous avons dû nous pencher sur certaines contraintes particulières à notre situation. Nous avons tout d’abord décidé de représenter la table à l’aide d’un nuage de points quadrillés, un peu à la manière d’un graphe ayant des coordonnées (x,y). Ce type de représentation facilite beaucoup la recherche de chemin puisque les algorithmes permettant la recherche de chemin les plus efficaces travaillent sur des graphes. Par la suite, il a fallu considérer que notre robot n’est pas un point ponctuel sur la table. Ce problème a été contourné en insérant une marge autour des obstacles. En ayant des obstacles plus gros, ceci nous permet de garder une représentation du robot comme un point et utiliser un algorithme de recherche de chemin. Quant à l’algorithme, nous avons choisi l’algorithme A*. Cet algorithme est très rapide et est facile à implémenter. Finalement, il y a une dernière contrainte à considérer, venant du fait que nous utilisons un nuage de point comme référence sur la table. Il est impossible de se déplacer en diagonale sans “zigzaguer” à travers les obstacles. Ceci engendre beaucoup de rotations nécessaires afin d’arriver au point final ce qui contribue à l’incertitude de la position réelle du robot. Nous ajouterons donc une couche logicielle permettant d’identifier les points critiques du chemin trouvé par l’algorithme A*. Ainsi, cette liste réduite de points permettra au robot de se déplacer en ligne droite lorsqu’il fera des diagonales en ignorant les points inutiles entre les points critiques qui créaient le mouvement zigzag, et réduira par le fait même le nombre de rotations nécessaires avant d’arriver au point final.

7.12 Kinect

La kinect est l’outil qui va nous aider à localiser les obstacles et localiser notre robot sous différents angles. Après l’installation d’OpenNi et OpenCv, on a testé la kinect avec le fichier test.cpp qui permet de faire une capture d’image.

Pour localiser le robot par rapport à la kinect, on va le modéliser avec une figure (exemple : deux cercles). Ensuite, grâce à la connaissance de la distance entre les deux cercles, on pourra déterminer à chaque rafraîchissement de position : la distance du robot par rapport à la kinect et par rapport aux obstacles et sa position angulaire. Pour les équations, on va utiliser des équations simples comme les règles de trigonométrie. Afin d’éviter le bruit dans nos images

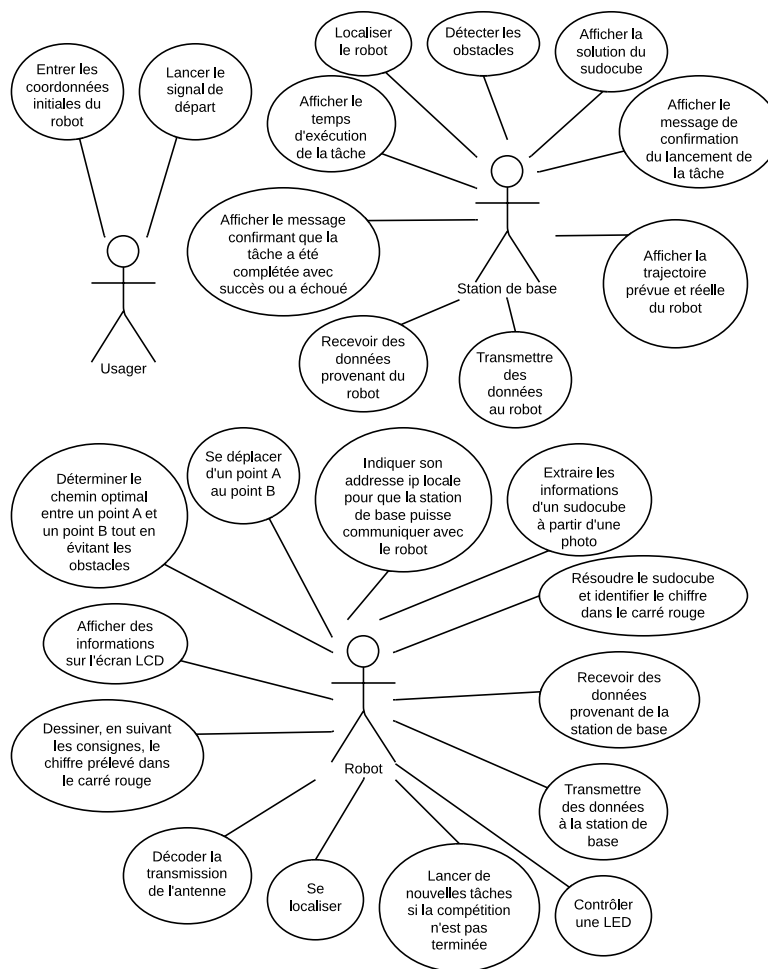
on va appliquer des filtres. Le filtre passe bas est le plus réputé pour enlever du bruit. Par contre, il rend les images "flou", alors il faut chercher les meilleurs paramètres pour avoir un filtre optimal et efficace. Pour la détection d'obstacles, on a deux choix soit faire une capture avec la caméra ensuite faire l'analyse 2D, soit utiliser l'image de profondeur. Pour le moment la solution qui semble efficace c'est l'utilisation de l'image de profondeur, elle consiste tout d'abord à isoler l'obstacle qui est déjà connu dans notre cas, ensuite conserver les points dont la profondeur est proche de celle des obstacles, on conserve une image noir et blanc, puisque c'est le contour de l'obstacle qui nous intéresse, finalement, on rentre l'image dans OpenCv pour définir un détournement de l'obstacle et l'affiner.

Annexe A

Annexes

A.1 Diagramme des cas d'utilisation

Figure A.1 – Diagramme des cas d'utilisation



A.2 Code test pour affichage sur LCD

A.2.1 ecran.h

```

1 #ifndef ECRAN_H_
2 #define ECRAN_H_
3
4 #include "inc/hw_ints.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "driverlib/debug.h"
8 #include "driverlib/gpio.h"
9 #include "driverlib/interrupt.h"
10 #include "driverlib/rom.h"
11 #include "driverlib/timer.h"
12 #include "inc/lm3s9b92.h"
13
14 //definition de trucs qu vont etre pratiques
15
16 // #define STCTRL (*((volatile unsigned long *)0xE000E010))
17 // #define STRELOAD (*((volatile unsigned long *)0xE000E014))
18 // #define attend(t) SysCtlDelay((SysCtlClock()/3)/(1000/t))
19
20 #define ECRAN_DATA GPIO_PORTA_DATA_R
21 #define ECRAN_CTRL GPIO_PORTA_DATA_R
22 //volatile unsigned long ulLoop;
23
24 //bit de data du LCD
25 #define ECRAN_D0 0x01 //PE0
26 #define ECRAN_D1 0x02 //PE1
27 #define ECRAN_D2 0x04 //PE2
28 #define ECRAN_D3 0x08 //PE3
29 #define ECRAN_D4 0x10 //PE4
30 #define ECRAN_D5 0x20 //PE5
31 #define ECRAN_D6 0x40 //PE6
32 #define ECRAN_D7 0x80 //PE7
33
34 //bit de control du LCD
35 #define ECRAN_RS 0x01 //PA0 reset
36 #define ECRAN_RW 0x02 //PA1 read/write
37 #define ECRAN_EN 0x04 //PA2 enable
38 #define ECRAN_BF 0x08 //PA3 "busy flag", indique que l'ecran est occupe
39
40 //volatile unsigned long ulLoop;
41 void ecranClear(void);
42 void ecranInit(void);
43 void ecranWriteChar(char caractere);
44 void ecranWriteLine(char * line, unsigned short size);
45 void ecranSetPosCursor(short pos);
46 void ecranAttend(void);
47 void ecranControl(unsigned long input);
48
49 #endif /*ECRAN_H_*/

```

A.2.2 ecran.c

```

1 #include "ecran.h"

3
4 /*
5  * Cette fonction permet de s'assurer que l'ecran n'est pas occupe avant d'ecrire
6  */
7 void ecranAttend(void)
8 {
9     ECRAN_CTRL &= ~(ECRAN_RS); // RS = 0
10    ECRAN_CTRL |= ECRAN_RW; // RW = 1
11    volatile unsigned long busyflag = ECRAN_CTRL & ECRAN_BF;
12    while (busyflag != 0) //on regarde le busyflag pour s'assurer que l'ecran est pas ←
13        occupe
14    {
15        busyflag = GPIO_PORTD_DATA_R & ECRAN_BF;
16    }
17    return;
18 }

19 /*
20  * Cette fonction execute la routine d'initialisation de l'ecran dans le mode qu'on veut
21  */
22 void ecranInit(void)
23 {
24     ecranControl(ECRAN_D5 | ECRAN_D4 | ECRAN_D3 | ECRAN_D2);
25     ecranControl(ECRAN_D3 | ECRAN_D2 | ECRAN_D1);
26 }

27
28 /*
29  * Cette fonction permet de faire des commandes tel que l'initialisation, changement de ←
30  * positions et
31  * trucs du genre, tout ce qui est pas l'écriture de caracteres sur l'ecran
32  */
33 void ecranControl(unsigned long input)
34 {
35     ecranAttend();
36     volatile unsigned long ulLoop;
37     ECRAN_DATA = input; //on met notre entree sur le port de donnees
38     ECRAN_CTRL &= ~(ECRAN_RS | ECRAN_RW); //reset et read/write a zero pour pas qu'il re-ecrive ←
39     par erreur
40     ECRAN_CTRL ^= ECRAN_EN;
41     ulLoop = SYSCTL_RCGC2_R;
42     for (ulLoop = 0; ulLoop < 200; ulLoop++)
43     {
44         // on met un delai pour que le LCD puisse voir le enable
45     }
46     ECRAN_CTRL ^= ECRAN_EN;
47     ulLoop = SYSCTL_RCGC2_R;
48     for (ulLoop = 0; ulLoop < 200; ulLoop++)
49     {
50         // on attend un peu pour que l'instruction s'execute avant de changer la pin 7
51     }
52     ECRAN_DATA &= ~(ECRAN_D7); //on reset la pin 7 parce elle output aussi le busy flag et ont ←
53     pas veux etre pogné dans ecranAttend()
54 }

55
56 /*
57  * Cette fonction efface le contenu de l'ecran
58  */
59 void ecranClear(void)
60 {

```

```

61     ecranControl(ECRAN_D0);
        volatile unsigned long ulLoop;
63
        for (ulLoop = 0; ulLoop < 2000; ulLoop++)
65     {
        //loop pour etre sur que ton est bien clearer
67     }
    }
69
    /*
71     * Cette fonction change la position du curseur d'ecriture de l'ecran
    */
73 void ecranSetPosCursor(short pos)
    {
75     ecranControl(GPIO_PIN_7 | pos);
    }
77
    /*
79     * Cette fonction permet d'ecrire un caractere sur l'ecran
    */
81 void ecranWriteChar(char caractere)
    {
83     volatile unsigned long ulLoop;
        ecranAttend();
85     ECRAN_DATA = caractere;
        ECRAN_CTRL &= ~(ECRAN_RW); //RW = 0
87     ECRAN_CTRL |= ECRAN_RS; // rs = 1
        ECRAN_CTRL ^= ECRAN_EN; //En = 1
89     ulLoop = SYSCTL_RCGC2_R;
        for (ulLoop = 0; ulLoop < 200; ulLoop++)
91     {
        //petit delai, juste pour etre sur
93     }
        ECRAN_CTRL ^= ECRAN_EN; //En = 0
95     }
97
    /*
99     * Cette fonction permet d'ecrire une chaine de caracteres sur l'ecran
    */
101 void ecranWriteLine(char * line, unsigned short size) //todo
    {
103     unsigned short i=0;
        for(i=0; i <size;i++)
105     {
        ecranWriteChar(line[i]);
107     }
    }

```