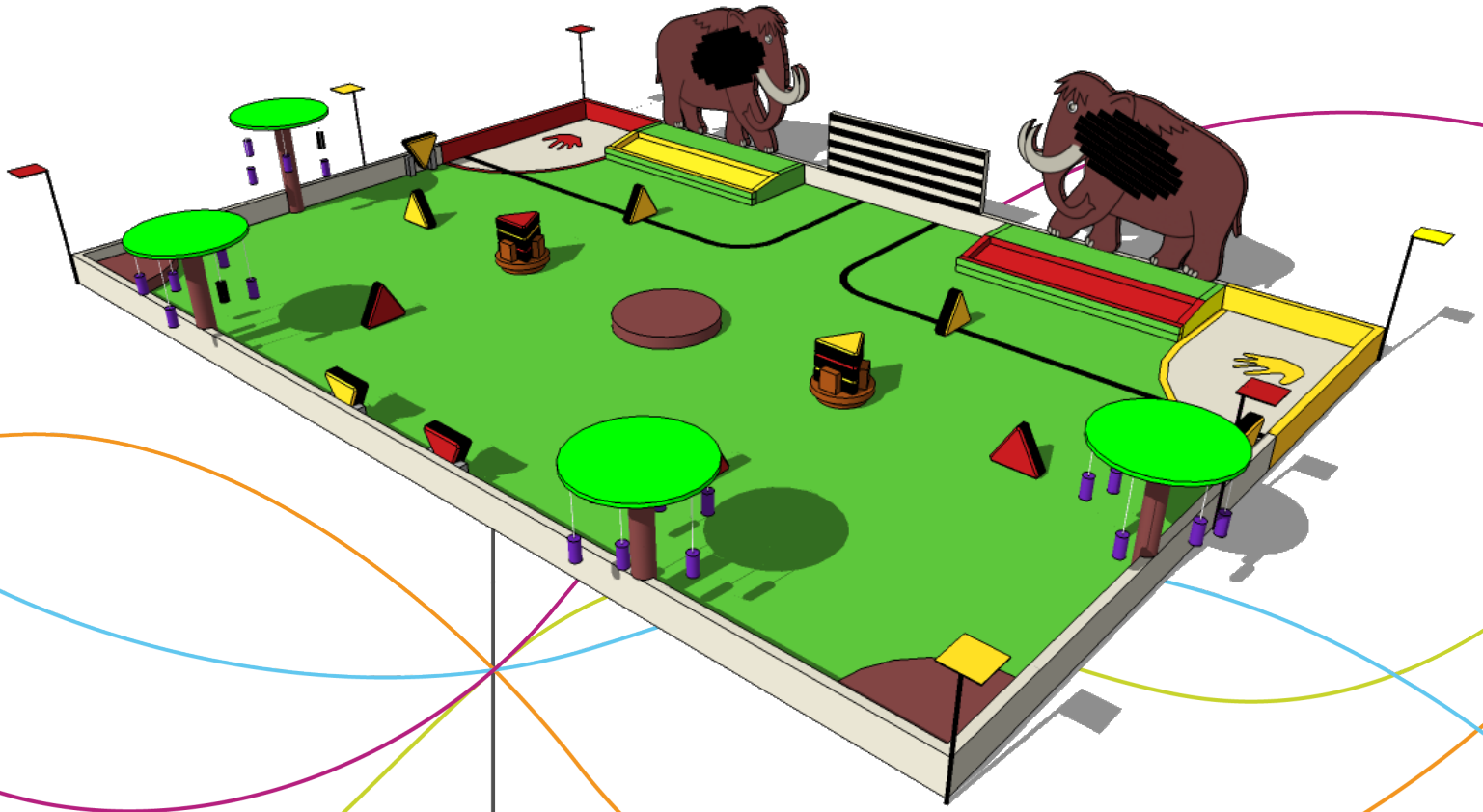
INSA ROUEN
À taille humaine,
à l'échelle du monde

Projet de Physique P6
STPI/P6/2013-2014

Robot pour la Coupe de France de Robotique
Pôles Electronique et Informatique



Enseignant responsable
Fabrice DELAMARE

Étudiants :
Manon BERTIN
Nathan LAPEL
Paul SELLE
Laura GELINEAU
Pierre-Jean AUBRIL

Date de remise du rapport : Pour le 16/06/2014

Référence du projet : STPI [1] /P6/2013–2014

Intitulé du projet : Robot pour la Coupe de France de Robotique

Type de projet : Expérimental

Objectifs du projet : (10 *lignes maximum)*

Il s'agit de créer un robot qui, théoriquement, pourra participer à la Coupe de France de Robotique 2014 dont le thème est la préhistoire. Le robot doit pouvoir faire un minimum d'actions parmi celles disponibles sur le plateau de jeu. (Le liste des actions et le détail de la Coupe de France sont expliqués dans le chapitre 1.) Pour la réalisation du robot, le groupe de 10 a du se scinder en deux : un pôle Mécanique et un pôle Électronique/Informatique. Ce rapport est donc celui du pôle Électronique et Informatique. Les détails du travail de chaque pôle sont expliqués ci-après.

1. INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE ROUEN
DÉPARTEMENT SCIENCES ET TECHNIQUES POUR L'INGÉNIEUR
685 AVENUE DE L'UNIVERSITÉ BP 08- 76801 SAINT-ETIENNE-DU-ROUVRAY
TÉL : 33 2 32 95 66 21 - FAX : 33 2 32 95 66 31

Table des matières

Introduction

Le but de ce projet de P6 est de réaliser un robot qui devrait pouvoir participer à la Coupe de France de Robotique et effectuer un certain nombre d'actions par rapport à celles possibles. Pour ce faire, nous avons divisé notre groupe de dix en deux groupes de cinq personnes. Chacun de ces petits groupes s'est attelé à une partie du robot : la Partie Mécanique, et la partie Electronique/Informatique. Ce rapport est celui des pôles Electronique et Informatique.

Laura et Pierre-Jean se sont donc attelés à la mise en place des deux cartes électroniques , pendant que Manon, Nathan et Paul se sont attaqués à la programmation de ce qu'allait faire le robot, que l'on a choisi totalement automate (par opposition à "intelligent") par manque de temps. C'est donc notre travail respectif que nous avons développé dans ce rapport.

Chapitre 1

Présentation du projet

1.1 La Coupe de France de Robotique

Créé par la ville de la Ferté-Bernard ainsi que par l'association Planètes-Sciences, elle regroupe chaque année depuis 1994 jusqu'à 185 équipes venant de toute la France. Ces équipes viennent de toutes sortes d'organisations : clubs indépendants, écoles d'ingénieurs, IUT, universités et même lycées.

Une fois inscrites, elles ont toutes une mission : réaliser un ou deux robots autonomes capables de réaliser une liste d'actions définies par l'organisateur. Le thème du concours change chaque année, obligeant les équipes à faire des nouveaux robots.

Quelque soit le thème, le principe du concours reste le même : il s'agit d'un tournoi de match de 90 secondes. Le tournoi se découpe en 3 phases :

L'homologation , qui consiste en la vérification par les organisateurs de la conformité des robots par rapport aux dimensions et caractéristiques mentionnées dans le règlement du concours.

Les matchs de qualifications , qui permettent de déterminer les finalistes du concours en classant toutes les équipes présentes.

Les matchs de phase finale , qui départagent les 16 meilleures équipes dont la gagnante.

1.2 L'édition 2014 : Préhistobot

Cette année, le thème du concours était Préhistobot. Il s'agissait de faire évoluer le(s) robot(s) dans un environnement à l'air de préhistoire. Le plateau de jeu est le suivant :

1.2.1 Les actions

Sur ce plateau, le(s) robot(s) a (ont) de multiples actions à faire :

La fresque : les robots doivent marquer l'Histoire de leur empreinte.

La conquête du feu : les robots doivent s'approprier le plus de feux.

La cueillette : les robots doivent cueillir le plus de "fruitmouths" possibles.

Les mammouths : les robots doivent envoyer le plus de lances possibles sur les mammouths.

Capture des mammouths : les robots peuvent capturer un mammouth à la fin du match.

Chacune de ces actions rapporte des points et c'est l'équipe qui en a le plus au bout des 90 secondes de match qui gagne.

1.2.2 Palmarès

Les gagnants de cette édition ont été : les Space Crackers de Castanet-Tolosan.

Les équipes de l'INSA ont fini :

PRIGER : 29ème

PRIGER 2 : 55ème

Chapitre 2

Pôle électronique

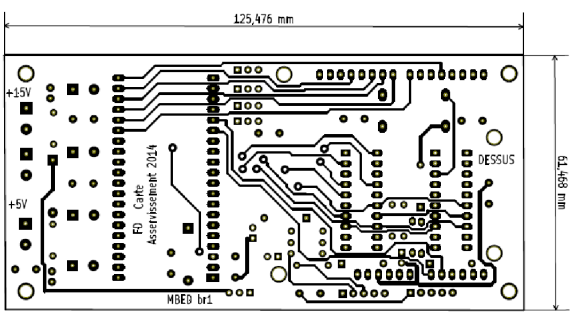
2.1 Méthodologie

Concernant le pôle électronique, l'objectif final était de produire une carte électronique complète pour le fonctionnement autonome du robot.

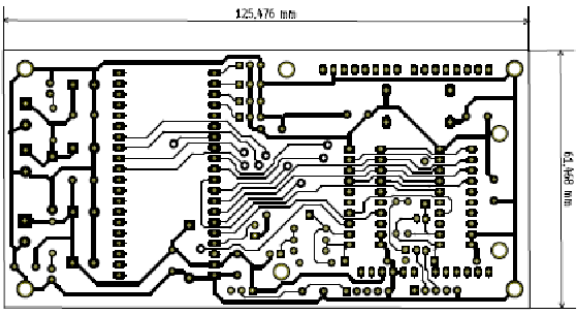
La partie électronique du robot s'est articulée autour de deux constituants principaux :
– La carte électronique d'asservissement
– Les cartes électroniques d'entrées, sorties et d'alimentation

2.1.1 Carte électronique d'asservissement

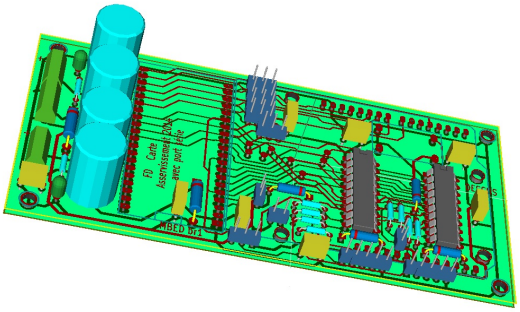
La conception par ordinateur de la cartes a été fournie par M.DELAMARE, notre professeur de projet. Cette dernière dessert un module MBED composant principal pour les déplacements du robot, elle envoie aux moteurs la puissance nécessaire ainsi que le sens de rotation pour faire le déplacement voulu. Grâce aux données fournies par les encodeurs du robot, elle agit sur ses moteurs de telle sorte que les déplacements soient précis et ne dévient pas de leur trajectoire. Les encodeurs sont placés à l'extérieur des roues motrices sur le même axe, ils ont pour but de mesurer la distance faite par le robot en temps réel, le programme corrigera l'erreur pour que le robot garde sa trajectoire, cela permet au robot de connaître sa position en temps réel ce qui est très utile pour les tache qu'il a besoin de réaliser.



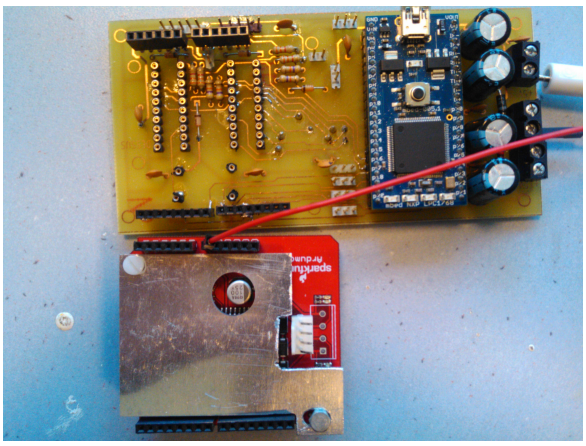
(a) Vue de dessus de la carte A



(b) Vue de dessous de la carte A



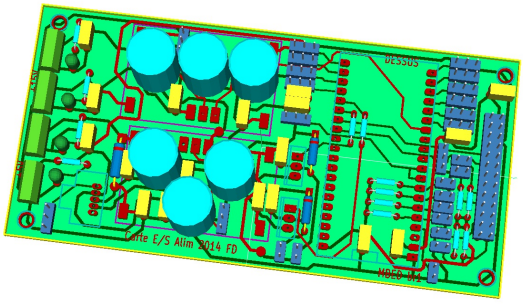
(a) Vue 3D de la carte A



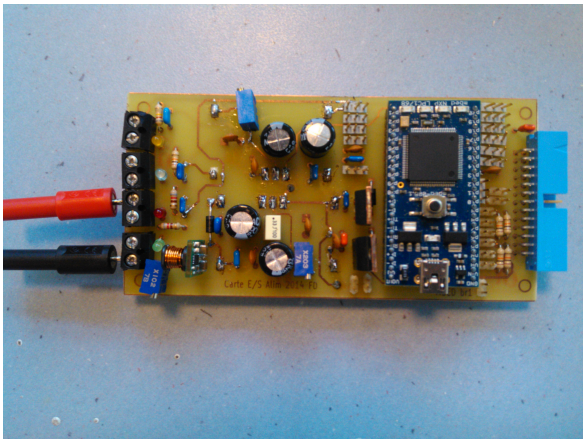
(b) Photo de la carte A

2.1.2 Carte électronique d'entrées, sorties et d'alimentation

Au début du projet, le but était de produire une unique carte mais pour des raisons de place et de simplicité, nous avons finalement réalisé deux cartes. La deuxième carte sert à faire fonctionner les fonctionnalités du robot tel que le bras, la pompe, le lance balle, les capteurs. Elle est aussi équipée d'un module Mbed pour gérer les information du programme et envoyer les signaux voulu aux servomoteurs pour qu'ils exécutent la tache voulue.



(a) Vue 3D de la carte B



(b) Photo de la carte B

Ces deux cartes comportent divers composants électroniques, certains utilisés fréquemment et d'autres éléments que nous avons découvert dans ce projet. Parmi les composants classiques, nous avons utilisé :

Des résistances et des potentiomètres (résistances variables) : permettent de baisser l'intensité dans un circuit.

Des condensateurs : permettent de stocker de l'énergie pour la restituer au besoin (en cas de micro coupures par exemple).

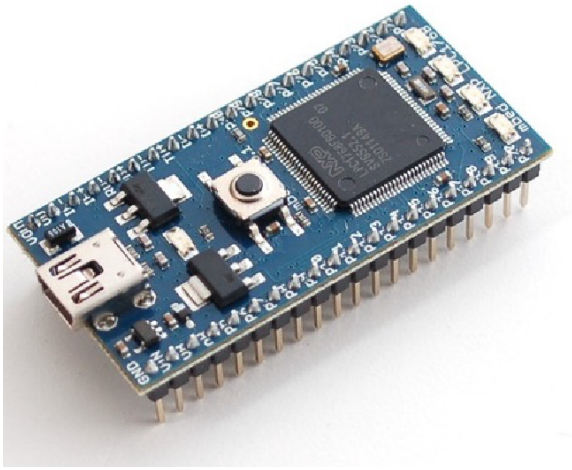
Des diodes : permettent de bloquer le passage du courant dans un sens sur une des pistes du circuit.

Des LED : ce sont des diodes qui font de la lumière comme l'indique leur nom (Light Emitted Diode), elles s'allument lorsque le courant les traverses (dans notre circuit les LED nous permettent de savoir si l'alimentation fonctionne).

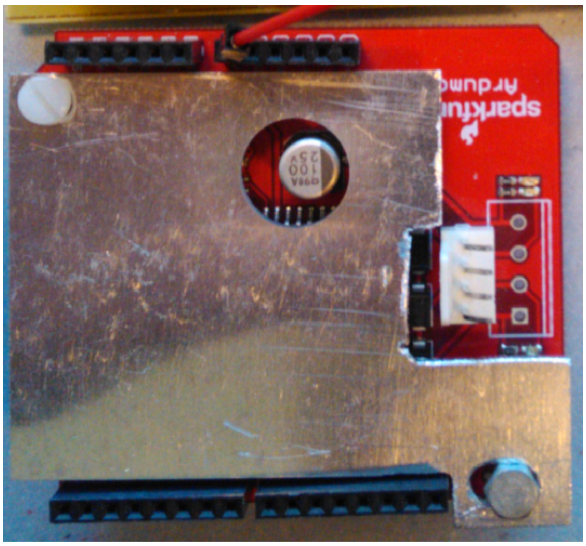
Quant aux composants découverts grâce à ce projet, nous avons utilisé :

Un module Mbed : micro contrôleur qui permet l'asservissement du robot par l'intermédiaire de deux moteurs, les déplacements sont contrôlé par deux encodeurs un pour chaque moteur pour qu'il soit précis, le robot doit savoir où il se trouve avec la plus grande précision possible pour être efficace.

Un module Arduino : reçoit des information du module Mbed et les transforme en tension pour les moteurs. Il reçoit pour chaque moteur un signal en temps réel pour savoir le sens de rotation de celui-ci, et un autre pour savoir la puissance à laquelle il doit tourner.



(a) Photo d'un module Mbed



(b) Photo d'un module Arduino

2.2 Organisation du travail

Tout d'abord, les deux cartes électroniques n'étant pas prêtes pour les deux premières séances, nous nous sommes chargés de commander les composants spécifiques à l'alimentation sur le site de T.I Instruments. Nous nous sommes aussi occupés de trouver la majeure partie des éléments à souder sur les cartes pour faciliter les séances de soudure et gagner du temps. C'est-à-dire :

Pour la première carte : 10 résistances variant de 270 à $47\,\mathrm{k\Omega}$, 13 condensateurs variant de $100\,\mu\mathrm{F}$ à $470\,\mu\mathrm{F}$, 6 diodes et 2 LED.

Pour la deuxième carte : 14 résistances, 24 condensateurs variant de $100\,\mathrm{nF}$ à $1000\,\mu\mathrm{F}$, 3 diodes, 3 LED, trois potentiomètres et 2 transistors.

Une fois la première carte récupérée, nous nous sommes mis très rapidement au travail. Nous avons commencé par tester chacune des pistes attentivement à l'aide d'un ohmmètre, pour vérifier qu'aucune des pistes n'était coupée et au besoin, les souder avec de l'étain et une patte de résistance. Nous avons ensuite commencer à créer les vias, qui sont des passages à souder entre les faces de la cartes pour éviter que des pistes se chevauchent, on en fait passer une sur l'autre face. Puis, nous avons soudé en premier les petits éléments tels que les résistances, les condensateurs, les LED et les diodes en faisant attention à la polarisation des diodes, afin de ne pas déranger la soudure des plus gros composants. Durant cette première étape, nous avons chacun soudé plusieurs composants alternativement puisque nous n'avions que la première carte en notre possession à cet instant.

Dès que nous avons récupéré la seconde carte, nous avons pu nous partager les tâches de manière plus efficace : Pierre-Jean s'est occupé des soudures les plus difficiles de la première carte, c'est-à-dire les plus rapprochées comme les supports, et Laura a testé les pistes et entamé la soudure des petits éléments de la seconde carte.

2.3 Travail réalisé

Comme nous l'avons décrit dans la partie méthodologie et organisation du travail, nous avons soudé tous les composants sur les deux cartes en commençant par les plus petits. Une fois les deux cartes finies, nous les avons testé au ohmmètre les soudures pour vérifier qu'elles étaient faites correctement et de chaque côté. Cependant, plusieurs d'entre-elles avaient été oubliées, nous avons donc tous les deux dessoudés certains composants gênant et refait les mauvaises soudures, cela nous a pris plusieurs séances pour trouver les problèmes, et les résoudre car les soudures oubliés n'étaient pas évidentes à faire.

Ensuite, après avoir résolu le problème des soudures, nous avons réglé l'alimentation à l'aide des potentiomètres soudés sur la deuxième carte de telle sorte que la carte, notamment grâce aux composants commandés sur le site de T.I Instruments, puisse générer des tensions de $5\,\mathrm{V}$, $12\,\mathrm{V}$ et $15\,\mathrm{V}$. Nous avons pu voir que les alimentations marchaient par l'intermédiaire des LED, nous avons eu un problème avec la LED de $12\,\mathrm{V}$ qui ne s'allumait pas correctement, cela était du à une résistance trop élevé que nous avons par la suite changée.

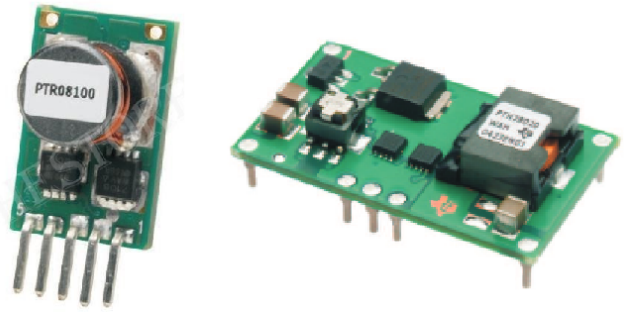


FIGURE 2.5 – Composants servant à diminuer la tension.

Lors de la dernière séance après avoir vérifié et changé les mauvaises soudures puis étalonné les deux cartes pour avoir les tensions voulues. Pour les étalonner nous avons du régler les potentiomètres de la carte 2 pour obtenir des tensions de 15, 12 et $5\,\mathrm{V}$ pour les différents composants du circuit. À la fin du projet nous avons pu tester quelques fonctionnalités du robot :

- Le faire se déplacer, il y a un problème d'ordre mécanique sur notre robot, les roues ont une mauvaise adhérence. Bien que la partie électronique fonctionne correctement, le robot n'avance pas droit et ses déplacements ne sont pas précis.
- Lever et descendre le bras, nous avons pu connecter les servomoteurs à notre carte électronique, on a ainsi pu observer le bon fonctionnement de cette partie du robot.
- Activer la pompe, lorsque nous avons branché la pompe sur la carte, elle ne fonctionnait pas correctement. Nous avons ensuite testé la pompe seule, qui elle marchait parfaitement, notre professeur en a donc conclu qu'il s'agissait d'un problème de transistor sur notre carte électronique. Nous n'avons pas eu le temps de corriger ce problème, cette fonctionnalité est donc inachevée.

D'autres fonctionnalités pourraient marcher mais faute de temps nous n'avons pas pu les tester, c'est le cas du lance balle, des capteurs de contacts.

Chapitre 3

Pôle Informatique

3.1 Méthodologie et organisation du travail

Afin de concevoir le programme de notre robot, nous nous sommes inspirés du précédent code créé par M. Delamare.

Pour que le travail s'effectue plus rapidement et plus efficacement (nous envisagions initialement de le présenter à la coupe de France de robotique), nous nous sommes répartis les tâches au sein du pôle Informatique. Manon et Paul, ayant alors de faibles connaissances dans le langage C++, ont commencé par en apprendre les notions essentielles. Dans le même temps, Nathan, plus à l'aise avec ce langage à conçu l'architecture globale du programme. Dans ce but, il a réutilisé les classes gérant le moteur, la gestion PID, et les encodeurs quadratiques (QEI) du précédent code, et imaginé les classes à créer et leurs interactions entre elles.

La programmation dans le cadre de la robotique est un domaine très différent de ce que nous avons étudié jusqu'alors dans nos cours d'informatique. Comme aucun de nous n'avait jamais conçu ou même étudié la conception de tels robots, il nous a fallu nous documenter sur leur fonctionnement ; le rôle des encodeurs et leur fonctionnement, le fonctionnement des capteurs,... A ce titre, nous avons beaucoup consulté le site de MBED, mais également d'autres ressources traitant des micro-contrôleurs et de la robotique, de l'asservissement... De même, bien que nous ayons travaillé spécifiquement sur la programmation de ce robot, il a été essentiel de confronter nos idées à celles des membres des autres pôles (mécanique et électronique) et de réfléchir avec eux sur les tâches qui nous incombaient et la façon dont nous allions les traiter. Aussi, toutes les fonctionnalités du robot (que sera-t-il capable de faire, de quels organes disposera-t-il) ainsi que la stratégie qu'il emprunte ont été décidé avec tous les membres du groupe travaillant sur le robot. Nous avons donc dû prendre en compte le temps dont nous disposions avant la coupe de France, celui dont nous disposions avant la date de rendu, le matériel en notre possession ainsi que notre niveau novice dans le domaine.

Une fois les bases du langage acquises et la pré-conception effectuée, nous nous sommes chacun mis a coder le programme indépendamment. Nathan s'est occupé de la gestion des mouvements (déplacements du robot, calculs de la position et de l'orientation en continu, en créant la classe *MotionManager*) et des classes structurant le programme (*Application*, *NonCopyable*). Manon s'est attelée à la gestion des capteurs de distances et de contact (classe *SensorManager*) et à la gestion des ports série (classe *SerialIO*). Enfin, Paul s'est chargé de l'implémentation de la stratégie du robot (classe *StrategyManager*) et de créer certaines classes

annexes (*Vector2*).

3.2 Travail réalisé : Documentation du programme

3.2.1 Référence de la classe Application

Classe gérant le déroulement du programme (singleton contenant la boucle principale).

`#include <Application.h>`

Est dérivée de NonCopyable.

Fonctions membres publiques

– int execute ()
Boucle principale du programme.

Fonctions membres publiques statiques

– static Application & getInstance ()
Permet de retourner l'instance unique de cette classe (construit l'objet s'il s'agit du premier appel).

3.2.1.1 Description détaillée

Classe gérant le déroulement du programme (singleton contenant la boucle principale).

3.2.1.2 Documentation des fonctions membres

**3.2.1.2.1 int Application : :execute ( ) ** Boucle principale du programme.

Renvoie

Code de retour (exit code) du programme.

3.2.1.2.2 Application & Application : :getInstance ( ) `[static]` Permet de retourner l'instance unique de cette classe (construit l'objet s'il s'agit du premier appel).

Renvoie

Instance unique.

La documentation de cette classe a été générée à partir des fichiers suivants :
– Application.h
– Application.cpp

3.2.2 Référence de la classe MotionManager

Gestionnaire de mouvement (correction PID, position actuelle, calcul de coordonnées, d'orientation...)

`#include <MotionManager.h>`

Est dérivée de NonCopyable.

Fonctions membres publiques

- MotionManager (Vector2 curpos=Vector2(), float orientation=0.f)
  Constructeur.
- ~MotionManager ()
  Destructeur.
- Motion : :Status update ()
  Met à jour l'état du mouvement (position, orientation, fin du mouvement).

3.2.2.1 Description détaillée

Gestionnaire de mouvement (correction PID, position actuelle, calcul de coordonnées, d'orientation...)

3.2.2.2 Documentation des constructeurs et destructeur

3.2.2.2.1 MotionManager : :MotionManager (Vector2 *curpos* = `Vector2()`, float *orientation* = `0.f`) Constructeur.

Paramètres

curpos	Position de départ du robot, celle à laquelle il a été posé.
orientation	Orientation initiale du robot (en radians).

3.2.2.3 Documentation des fonctions membres

3.2.2.3.1 Motion : :Status MotionManager : :update ( ) Met à jour l'état du mouvement (position, orientation, fin du mouvement).

Renvoie

Etat du mouvement actuel.

La documentation de cette classe a été générée à partir des fichiers suivants :
- MotionManager.h
- MotionManager.cpp

3.2.3 Référence de la classe SensorManager

Gestionnaire des capteurs.

`#include <SensorManager.h>`

Types publics

- enum ContactSensor

Désigne un capteur de contact.
- enum DistanceSensor
  Désigne un capteur de distance.
- enum Distance
  Désigne une distance obtenue via un capteur de pression.

Fonctions membres publiques

- SensorManager ()
  Constructeur par défaut.
- bool contact (ContactSensor sensor) const
  Indique si un capteur indique qu'un contact est en cours.
- Distance getDistance (DistanceSensor sensor) const
  Indique la distance indiquée par un capteur.

3.2.3.1 Description détaillée

Gestionnaire des capteurs.

3.2.3.2 Documentation des fonctions membres

3.2.3.2.1 bool SensorManager : :contact (ContactSensor *sensor*) const Indique si un capteur indique qu'un contact est en cours.

Paramètres

sensor	Identifiant du capteur à examiner.

Renvoie

Vrai s'il y a contact, faux sinon.

3.2.3.2.2 SensorManager : :Distance SensorManager : :getDistance (DistanceSensor *sensor*) const Indique la distance indiquée par un capteur.

Paramètres

sensor	Identifiant du capteur à examiner.

Renvoie

Distance indiquée par le capteur.

La documentation de cette classe a été générée à partir des fichiers suivants :
- SensorManager.h
- SensorManager.cpp

3.2.4 Référence de la structure SerialCommand

Représente une commande reçue sur le port série.

`#include <SerialIO.h>`

Types publics

– enum
Type de la commande.

Attributs publics

– enum SerialCommand : : { ... } type
Type de la commande.

3.2.4.1 Description détaillée

Représente une commande reçue sur le port série.

La documentation de cette structure a été générée à partir du fichier suivant :
– SerialIO.h

3.2.5 Référence de la classe SerialIO

Gère les interactions avec le port série.

`#include <SerialIO.h>`

Fonctions membres publiques

– SerialIO ()
Constructeur par défaut.
– bool getNewCommand (SerialCommand &command)
Permet d'obtenir une commande reçue sur le port série.
– Serial & getRawSerial ()
Permet d'obtenir l'objet associé au port série utilisé.
– const Serial & getRawSerial () const
Permet d'obtenir l'objet associé au port série utilisé.

3.2.5.1 Description détaillée

Gère les interactions avec le port série.

3.2.5.2 Documentation des fonctions membres

3.2.5.2.1 bool SerialIO : :getNewCommand (SerialCommand & *command*) Permet d'obtenir une commande reçue sur le port série.

Paramètres

command	Contiendra la commande reçue après l'appel à la fonction.

Renvoie

Vrai si une commande valide a été reçue, faux sinon.

3.2.5.2.2 Serial & SerialIO : :getRawSerial ( ) Permet d'obtenir l'objet associé au port série utilisé.

Renvoie

Objet de type Serial associé.

3.2.5.2.3 const Serial & SerialIO : :getRawSerial ( ) const Permet d'obtenir l'objet associé au port série utilisé.

Renvoie

Objet de type Serial associé.

La documentation de cette classe a été générée à partir des fichiers suivants :
– SerialIO.h
– SerialIO.cpp

3.2.6 Référence de la classe StrategyManager

Gère la stratégie, l'ordre des commandes, interprète les retours d'information (capteurs)

`#include <StrategyManager.h>`

Est dérivée de NonCopyable.

Fonctions membres publiques

– StrategyManager (Robot : :Color color)
Constructeur.
– MotionManager & getMotionManager ()
Permet d'obtenir le gestionnaire de mouvement utilisé.
– bool checkSensors ()
Vérifie l'état des capteurs.
– bool update (float etime)
Met à jour l'état du gestionnaire de stratégie (réfléchit à la situation actuelle et planifie les prochaines actions).

3.2.6.1 Description détaillée

Gère la stratégie, l'ordre des commandes, interprète les retours d'information (capteurs)

3.2.6.2 Documentation des constructeurs et destructeur

3.2.6.2.1 StrategyManager : :StrategyManager (Robot : :Color *color*) Constructeur.

Paramètres

color	Couleur du robot lorsque la partie commence.

3.2.6.3 Documentation des fonctions membres

3.2.6.3.1 bool StrategyManager : :checkSensors ( ) Vérifie l'état des capteurs.

Renvoie

Faux si les capteurs indiquent que le mouvement en cours doit être arrêté, vrai sinon.

3.2.6.3.2 MotionManager & StrategyManager : :getMotionManager ( ) Permet d'obtenir le gestionnaire de mouvement utilisé.

Renvoie

Gestionnaire de mouvement.

3.2.6.3.3 bool StrategyManager : :update (float *etime*) Met à jour l'état du gestionnaire de stratégie (réfléchit à la situation actuelle et planifie les prochaines actions).

Paramètres

etime	Temps écoulé depios le dernière appel à cette fonction (en secondes).

Renvoie

Faux si la liste des actions à effectuer est vide, vrai sinon.

La documentation de cette classe a été générée à partir des fichiers suivants :
– StrategyManager.h
– StrategyManager.cpp

3.2.7 Référence du fichier Pins.h

```
#include "mbed.h"
```

3.2.7.1 Description détaillée

Contient les alias des pins à utiliser pour chaque fonction.

Conclusion et perspectives

Pour tous, ce projet nous a permis d'apprendre un aspect de la fabrication d'un robot, que ce soit le côté électronique pour Laura et Pierre-Jean ou le côté programmation pour Paul, Manon et Nathan. Nous avons avancé petit à petit, et le plus grand problème que nous avons rencontré est le fait d'avoir testé tard, voire pas du tout, nos réalisations.

Pour la suite, il faudrait tester, une fois les parties mécaniques réajustées, le programme conçu. Selon les résultats, il faudra réajuster les éléments de la stratégie et les possibles problèmes pratiques liés.

Bibliographie

[1] http://www.planete-sciences.org/robot/data/file/coupe/2014/Rules2014%20-%20Version%20finale%20-%20Eurobot.pdf (Valide à la date du 15/06/14)

[2] http://fr.wikipedia.org/wiki/Coupe_de_France_de_robotique (Valide à la date du 15/06/14)

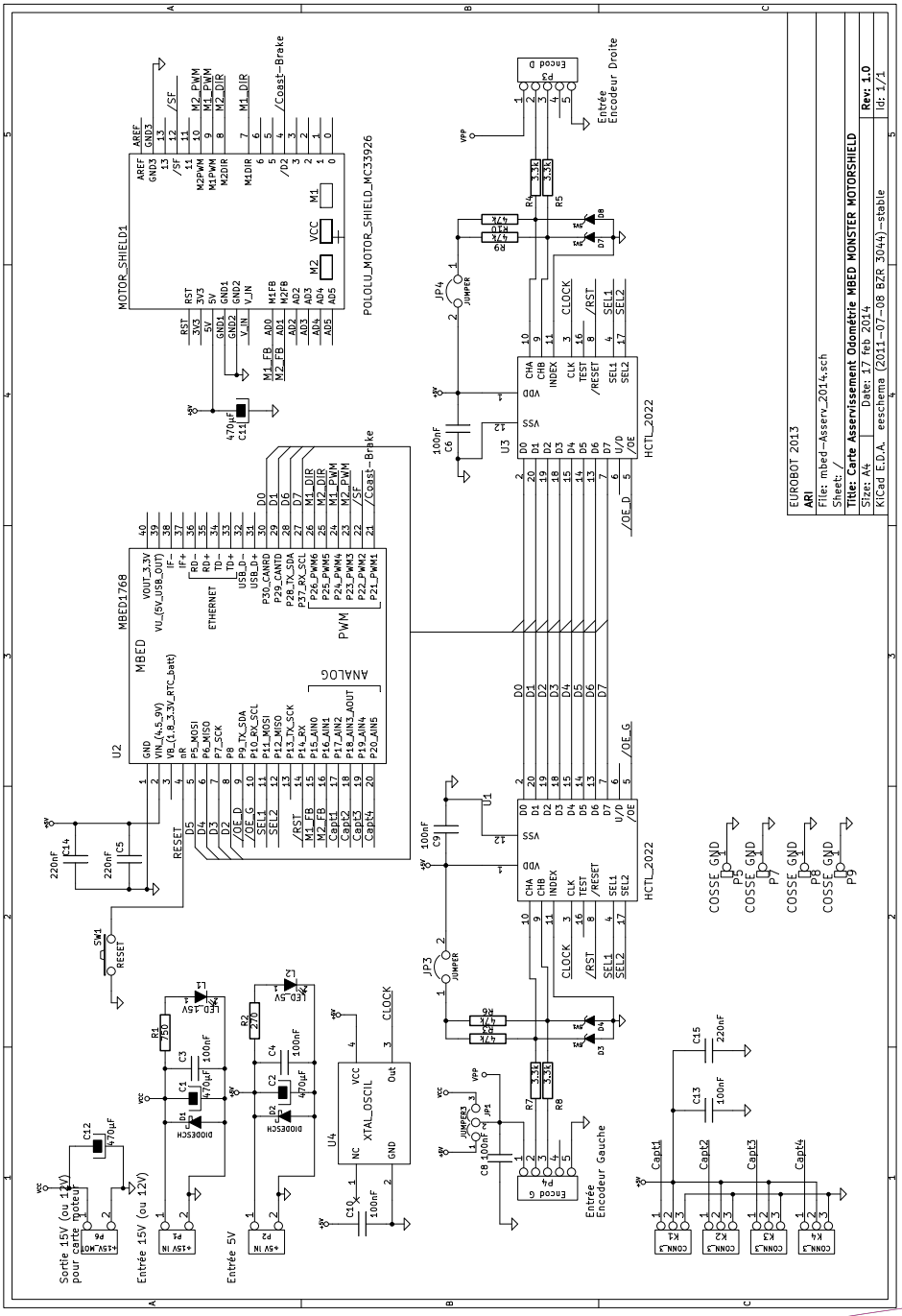
[3] http://www.planete-sciences.org/robot/index.php?section=pages&pageid=79 (Valide à la date du 15/06/14)

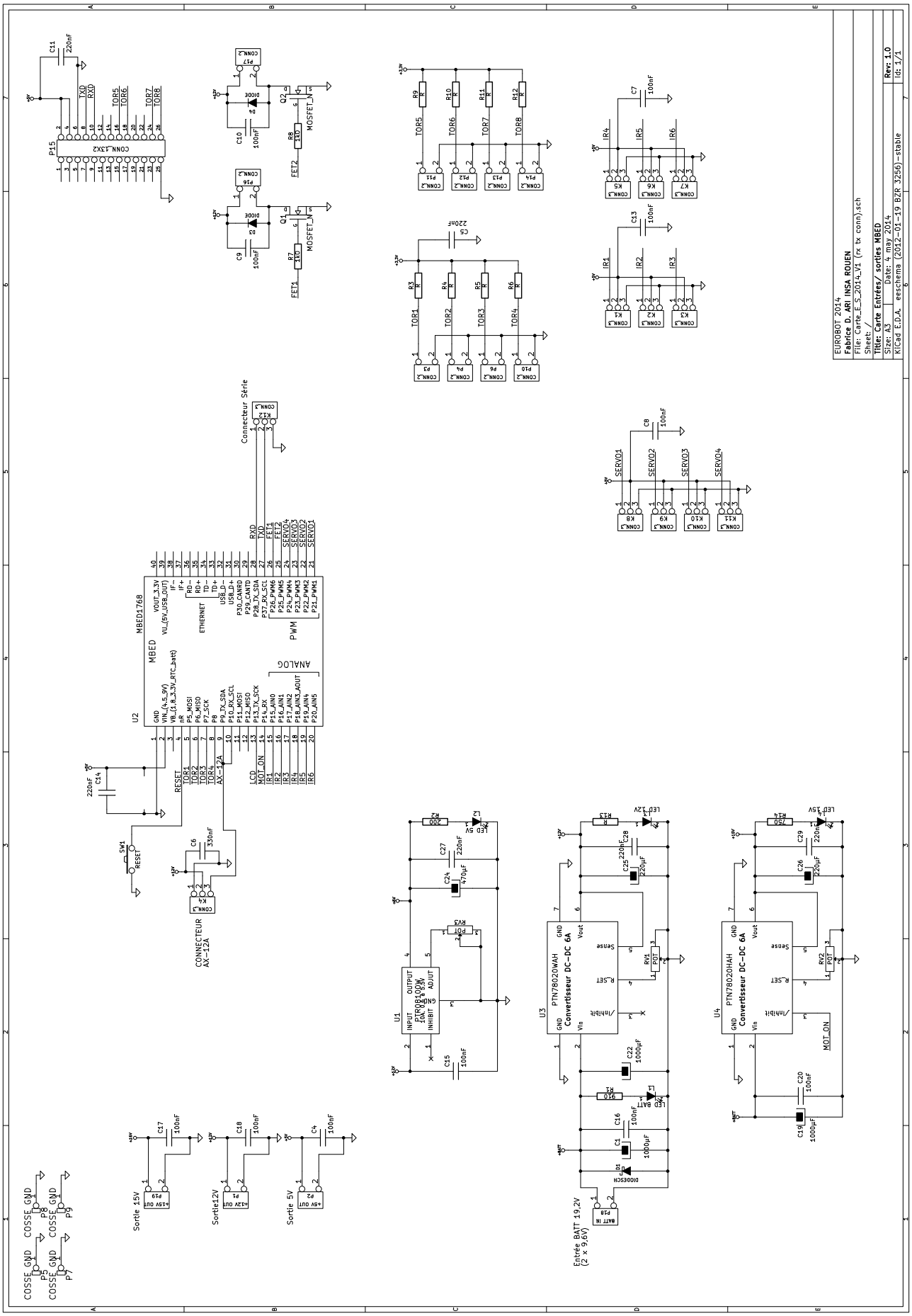
[4] https://mbed.org (Valide à la date du 15/06/14)

[5] http://www.ti.com/ (Valide à la date du 15/06/14)

Annexe A

Schémas des deux cartes




EUROBOT 2014
Fabrice D. ARI INSA ROUEN
File: Carte_E_S_2014_V1 (rx tx conn).sch
Sheet: /
Title: Carte Entrées/ sorties MBED
Size: A3
Date: 4 may 2014
KiCad E.D.A. eeschema (2012-01-19 BZR 3256)-stable
Rev: 1.0
Id: 1/1
MBED1768
MBED
ETHERNET
PWM
ANALOG
Connecteur Série
CONNECTEUR AX-12A
Sortie 15V
Sortie12V
Sortie 5V
Entrée BATT 19,2V (2 x 9,6V)
PTN78020WAH
Convertisseur DC-DC 6A
LED 5V
LED_12V
LED_15V
LED BATT
MOT_ON
RESET
COSSE GND