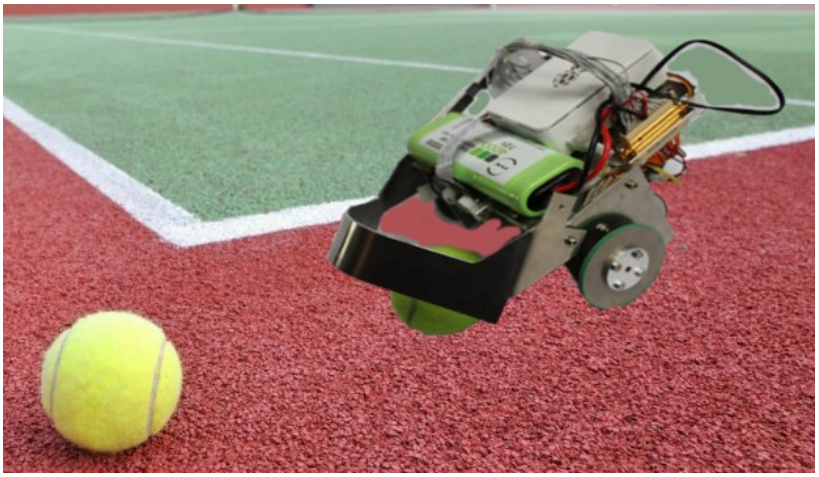
PRÉVOST Thomas TS3

Projet de spécialité I.S.N. terminale : Compte-rendu :

Définition du projet :

Introduction :

Le projet consiste en un **robot qui va chercher des balles de tennis** et les ramène aux joueurs.
(photo non contractuelle)



Lorsqu'on joue au tennis, le moment « ramassage de balles » est toujours le moins amusant, d'autant plus que ces dernières vont se placer dans des endroits improbables (sous un banc, une chaise…). Le projet consiste donc en un robot qui irait chercher les balles en parfaite autonomie. Lorsque je joue au tennis, c'est sur les courts les plus proches de chez moi, qui ont un revêtement béton. C'était donc un environnement idéal pour qu'une machine puisse s'y déplacer.
C'est un **projet en solo**, je ne suis pas en équipe.

Recherche d'idées :

Lorsque je suis entré dans la spécialité I.S.N., en Septembre 2017, j'ai immédiatement commencé à chercher des idées de projet dans le but de démarrer le plus tôt possible. Il fallait quelque chose de réalisable et qui, si possible, fasse appel à des connaissances dont je disposais déjà. Il fallait en outre une certaine difficulté (pour ne pas le terminer en 2 jours) et que je puisse m'identifier au projet (ici via le tennis). Finalement, c'est en discutant avec ma famille que l'idée nous est venue, et la réalisation a pu démarrer en Octobre 2017.

PRÉVOST Thomas TS3

Cahier des charges :

I – Définition du projet :

Le projet se définit comme un « *robot qui va chercher et rapporte des balles de tennis* ». C'est une machine autonome capable de repérer des balles de tennis, de rejoindre une de ces balles et ensuite de la ramener jusqu'à un point donné. La machine devra ensuite recommencer jusqu'à ce qu'il n'y ait plus de balles à aller chercher. Elle devra fonctionner sur terrain plat et lisse (type « court de tennis béton »), voire synthétique et terre battue si possible. Pour plus de facilités, l'environnement (sol, murs...) pourra être d'une teinte unie et opaque. On suppose que ce dernier respecte les conditions d'extérieur en plein jour par beau temps (au moins 10 000 lux), sans perturbation électromagnétique, acoustique...

II – Exigences / sécurité :

La machine devra respecter certaines dimensions : 50 cm X 50 cm X 50 cm maximum ; sa masse ne devra pas excéder 5 Kg, de façon à être transportable facilement à la main et sur une distance de plus de 1 Km. Elle ne devra pas être bruyante (privilégier le moteur électrique au moteur thermique).

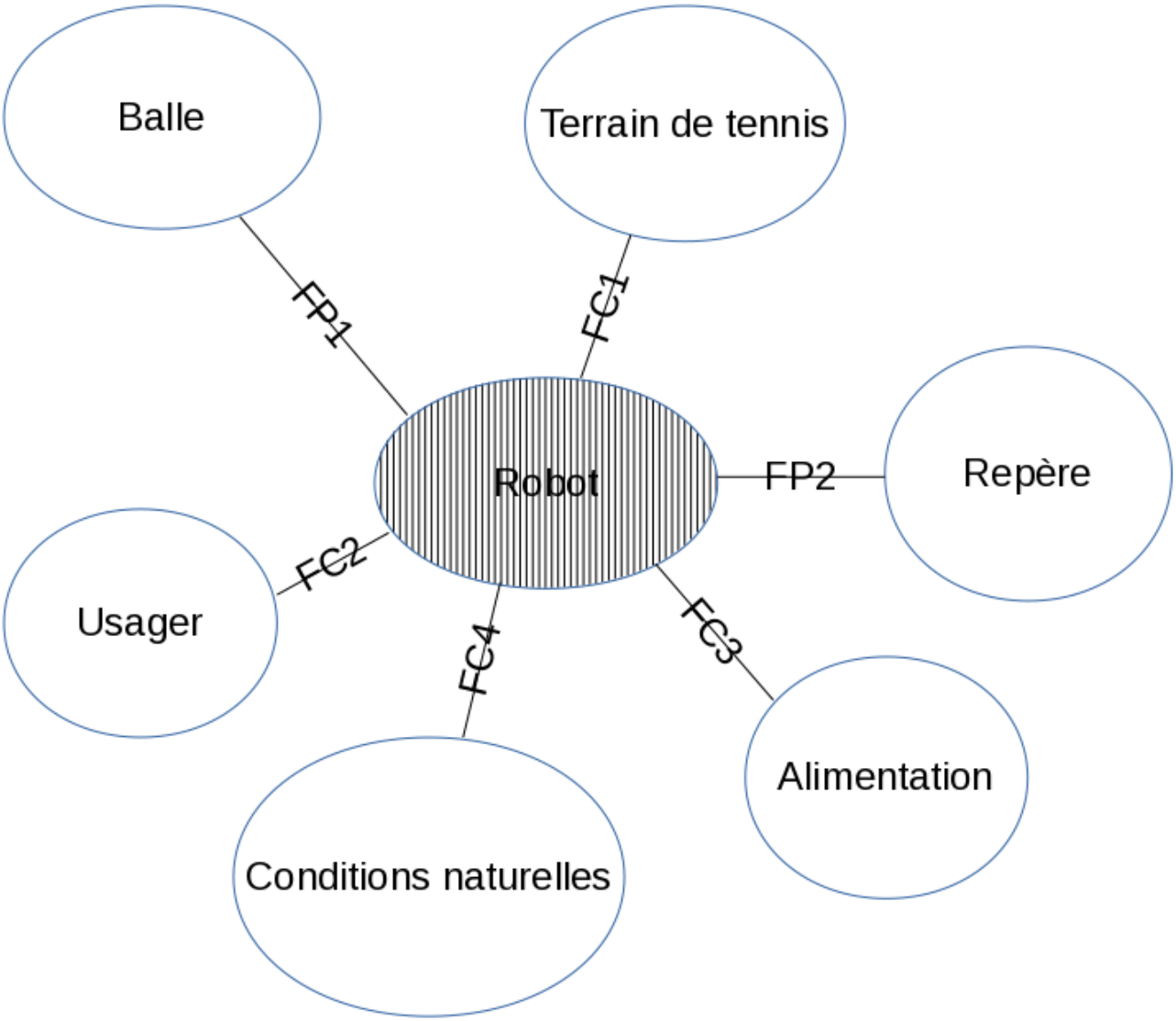
La machine ne doit évidemment pas être dangereuse : elle se déplacera à une vitesse maximale de 20 Km/h, aucune partie coupante/tranchante ne devra apparaître et être actionnée par la machine ; les parties susceptibles de chauffer à des températures supérieures à 50 °C devront être protégées.

III – Performances requises :

La machine devra être capable de détecter et d'aller chercher une balle située à une distance de 19 mètres. L'autonomie devra être de 60 minutes minimum. Pour aller chercher une balle située à 19 mètres, la machine prendra 1 minute maximum. La phase de récupération de la balle est très importante : en cas d'échec la balle de doit pas être projetée trop loin. Si le terrain est supposé plat, la machine devra être un minimum stable et ne pas se renverser sur une pente allant jusqu'à 10°.

La machine devra obligatoirement fonctionner d'elle même ou utiliser uniquement les ressources présentes à proximité (sur le court) : elle n'utilisera pas les réseaux satellitaires (GPS…), GSM...

L'aspect esthétique de la machine n'a que très peu d'importance.



FP1 : Repérer et aller chercher la balle, une seule à la fois.
FP2 : Rapporter la balle sur un repère choisi, et repartir pour FP1

FC1 : Se déplacer sur un terrain de tennis plat « béton » assez vite pour rejoindre la balle avec un temps raisonnable
FC2 : Ne pas mettre en danger l'usager, ne pas être encombrant (max 50cm X 50cm X 50 cm)
FC3 : Autonomie de 15 minutes minimum

PRÉVOST Thomas TS3

Solutions techniques envisagées :

I – Objectifs :

Le robot sera capable de détecter une balle de tennis jaune dans un environnement si possible uniforme et dont la couleur contraste bien avec celle de la balle. Le sol sera si possible plat et sans obstacle pour permettre un meilleur fonctionnement du capteur ultrasons. Le robot se déplacera jusqu'à la balle, l'attrapera, fera marche arrière et ira chercher une autre balle.

II – Matériel :

L'ordinateur du robot pourrait être un Raspberry Pi, ou tout autre mini ordinateur. La détection de la balle devrait se faire avec une caméra Raspberry. Lorsque le robot se situera en face de la balle à une distance inférieure à 1,50 mètres environ, la détection se fera via le capteur ultrasons. Il se déplacera grâce à deux moteurs pas-à-pas. Ces moteurs pourraient se contrôler avec les pins GPIO du Raspberry Pi. Comme ces derniers ne pourront délivrer seulement une tension de 3V, ils devraient être amplifiés pour alimenter les moteurs en 7V environ (à tester). La balle devrait être attrapée par une structure pivotante mue grâce à un servomoteur (le plus simple). Une seule balle sera donc stockée à chaque fois : le robot fera marche arrière et déposera la balle.

Il serait facile d'alimenter le robot par une batterie NIMH 7,2V. Le châssis sera en aluminium, solide, léger et facile à usiner.

III – Développement :

Le système d'exploitation équipant le Raspberry Pi sera Raspbian Lite (distribution basée sur Debian, pour une architecture ARM) de préférence. Le contrôle du robot se fera au moyen d'un seul et même programme, développé en C++ (que je connais déjà, et qui est plus polyvalent que le Python). Il tournera à terme sous forme d'un service (démon), se lançant donc dès le boot / mise sous tension.

Après quelques recherches, je pense que la détection de balles se fera grâce à la bibliothèque OpenCV, le contrôle du bus I2C et du GPIO (pins pouvant prendre une valeur binaire) grâce à la bibliothèque WiringPi.

Sources :

L293D : http://meayne.free.fr/isn/L293D.pdf
Moteurs : https://fr.wikipedia.org/wiki/Moteur_pas_%C3%A0_pas

PRÉVOST Thomas TS3

SRF08 : http://meayne.free.fr/isn/SRF08.pdf / http://meayne.free.fr/isn/SRF08-2.pdf / https://fr.wikipedia.org/wiki/I2C
GPIO : http://meayne.free.fr/isn/GPIO-physique.png / http://meayne.free.fr/isn/GPIO-WiringPi.png / http://wiringpi.com/
OpenCV : https://opencv.org/ / https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/ / https://www.elphel.com/wiki/OpenCV_Tennis_balls_recognizing_tutorial / https://docs.opencv.org/3.4.0/d7/d5d/houghcircles_8cpp-example.html / https://stackoverflow.com/questions/9179189/detect-yellow-color-in-opencv etc.
C++ : https://openclassrooms.com/courses/programmez-avec-le-langage-c / http://www.cplusplus.com/ / ISBN 0-672-30923-8
Servomoteur : http://www.instructables.com/id/Servo-Motor-Control-With-Raspberry-Pi/ / http://espace-raspberry-francais.fr/Composants/Controler-Servo-Moteur-Raspberry-Francais/
Linux : https://openclassrooms.com/courses/reprenez-le-controle-a-l-aide-de-linux / https://doc.ubuntu-fr.org/ / https://openclassrooms.com/courses/faire-un-demon-sous-linux / https://doc.frapp.fr/doku.php?id=tutoriel:linux:demon / https://doc.ubuntu-fr.org/tutoriel/comment_transformer_un_programme_en_service

Réalisation :

Matériel :

J’ai commencé à construire le robot autour du Raspberry Pi (modèle 3). La structure est faite en aluminium usiné. L’alimentation se fait grâce à une batterie accumulateur nickel-hydrure métallique (NIMH, type « voiture télécommandée ») 7,2 V, 4 000 mAh. La tension est ensuite convertie en 5V pour le Raspberry (norme USB) et les capteurs, et est laissée telle quelle pour les moteurs. On utilise pour cela un convertisseur à courant continu réglable. La détection de balle se fait tout d’abord grâce à la « caméra Module V2 Raspberry Pi » HD (3280 x 2464), reliée au port « camera » du Raspberry. Elle se fait ensuite plus précisément grâce à un capteur ultrasons SRF08 (I2C). Les 40 pins du Raspberry Pi sont reliées à une nappe 40 points jusqu’à un connecteur 25 contacts.

Le robot se déplace grâce à 2 moteurs pas-à-pas indépendants tournant à pas complets, utilisés en bipolaires. Il y a une troisième roue libre (récupérée sur un jeu de Meccano) à l’arrière qui pivote grâce à un roulement à billes de disque dur. Toutes les instructions du Raspberry Pi transitant par les ports GPIO 3V, il est nécessaire d’amplifier la puissance électrique reçue par les moteurs. Cela se fait au moyen de 2 circuits L293D, dont l’alimentation peut être coupée lorsque les roues ne tournent pas par la commande GPIO « unable » (pour ne pas trop consommer). La préhension de la balle se fait grâce à une pièce pivotante, actionnée par un servomoteur HITEC HS-303 (récupéré sur un avion télécommandé). Le servomoteur est commandé en largeur d’impulsion.

Au final, le Raspberry, la batterie et le servomoteur seront placés sur le robot. Le capteur ultrasons et les puces électroniques en dessous, à l'arrière. La balle passera sous le robot par l'avant et s'arrêtera au niveau du capteur ultrasons. La pièce pivotante se placera devant la balle pour la bloquer.

Développement :

Le programme est écrit en C++. La gestion de la caméra se fait grâce à la bibliothèque OpenCV ; la gestion du GPIO (moteurs compris) et de l'I2C grâce à la bibliothèque WiringPi. Voici le fonctionnement des différentes parties du programme :

Caméra : Le programme récupère tout d'abord une image de la caméra. Il détecte ensuite une certaine plage de couleur dans l'image correspondant au jaune des balles de tennis (grâce à HSV, Hue Saturation Value et non pas RGB, Red Green Blue). On obtient donc une seconde image de mêmes dimensions « masque » binaire, avec les pixels dans la bonne plage blancs (255) et les autre noirs (0). Le masque n'étant évidemment pas parfait, le programme l'affine avec une fonction gaussienne, de façon à perfectionner les courbes. Il détecte ensuite les ronds dans le masque (la forme des balles). La caméra se trouvant au centre du robot et couvrant un angle connu, on connaît alors la direction et la distance (trigonométrie) des balles. Le robot s'occupera alors de la balle la plus proche. A noter que la reconnaissance d'images n'est jamais parfaite et peut toujours être améliorée.

Ultrasons : Le capteur ultrasons se contrôle via un bus I2C. Pour effectuer une mesure, il faut simplement « écrire » l'unité (ici centimètres, mais aussi pouces ou microsecondes), attendre un peu pour que le capteur fasse la mesure (0,5 secondes par exemple) puis « lire » le résultat, sous forme de 2 octets. Pour plus de précision, le programme effectue plusieurs mesures (3 par exemple) et prend la valeur médiane.

Moteurs pas-à-pas : Le contrôle des moteurs pas-à-pas se fait via l'interface GPIO. Pour les démarrer, il faut tout d'abord régler la pin « unable » sur 1. Contrairement aux câblages unipolaires, les câblages bipolaires permettent d'inverser le sens du courant. Le moteur se contrôle via une succession d'impulsion séparées par un intervalle t (≥ 5 ms). Cela signifie donc que plus t est petit, plus la vitesse du moteur est élevée. Cependant sur tous les moteurs le couple (force) diminue avec la vitesse. J'ai choisi ici un intervalle t=20 ms. Le moteur est déjà construit avec un rapport de réduction dû à des engrenages. Il faut 200 pas (ou impulsions) pour que le moteur fasse un tour complet.

Fonctionnement du moteur à pas complets :

	Aimant 1	**Aimant 2**	**Aimant 3**	**Aimant 4**
Impulsion 1	+	0	0	0
Impulsion 2	0	0	+	0
Impulsion 3	0	+	0	0
Impulsion 4	0	0	0	+

La page Wikipédia consacrée aux moteurs pas-à-pas est très explicite :
https://fr.wikipedia.org/wiki/Moteur_pas_%C3%A0_pas

Servomoteur : Le servomoteur permettra de faire pivoter la pièce chargée de bloquer la balle. C'est un moteur capable de prendre une position et de se bloquer. Il se contrôle grâce à un signal carré (créneau), impulsé depuis un port GPIO (0 / 3V). Il faut envoyer une onde de période 20 millisecondes. L'angle du moteur (de 0° à 180°) est donné par la durée de l'impulsion : entre 500 et 2 500 µs. Lorsqu'on arrête le signal, le moteur s'arrête, peu importe sa position. Il faut donc envoyer le signal en continu jusqu'à ce que le moteur ait atteint la position souhaitée. Chaque période durant 20 ms, le programme va envoyer 35 périodes, car le moteur peut faire les 180° au complet en moins de 0,7 secondes.

Mise en relation des capteurs et des effecteurs : Le programme final se présente sous la forme d'une boucle infinie, lancée dès la mise sous tension du Raspberry. Le programme cherche d'abord une balle avec la caméra. S'il n'en trouve pas, le robot pivotera de 30° jusqu'à en trouver une. Une fois la balle la plus proche trouvée, le robot pivotera afin d'être bien en face de la balle. Il s'approchera de la balle en évaluant la distance aux ultrasons. Lorsqu'il aura atteint la balle, le servomoteur s'actionnera pour attraper la balle. Le robot reculera ensuite de la distance parcourue, puis reculera encore de 20 cm, pivotera de 30° et recommencera.

Planification :

On remarque dans ce présent rapport de projet que la partie « Solutions techniques choisies » entrent dans la partie « Réalisation ». En effet, ce projet s'est construit autour du prototypage, et je n'ai pas hésité à modifier certaines parties après réalisation.

Je n'ai pas réellement été tout seul pour se projet puisque mon père m'a aidé pour la partie hardware et la structure physique en général. Il m'a notamment aidé à usiner les pièces en aluminium (à la fraiseuse) et m'a montré comment souder les parties électronique.

On ne peut pas réellement parler de planification de mon projet, car on effectuait les tâches au jour le jour, en fonction des problèmes rencontrés et de notre temps libre. Ceci dit, voici grosso modo comment nous avons réalisé ce projet :

La réalisation a débuté en Octobre 2017. Sachant qu'il faudrait indéniablement passer par la reconnaissance d'images, je me suis immédiatement mis au développement d'un programme capable de détecter les balles de tennis. J'avais déjà trouvé des exemples sur Internet mais développés en C, avec la bibliothèque OpenCV. J'ai tout de suite commencé par convertir le code en C++. Malheureusement, l'algorithme de départ détectait n'importe quoi, puis rien du tout… J'ai donc dû modifier pas mal de chose pour obtenir un résultat satisfaisant. Dans le même temps, on commençait la CAO et à usiner les pièces du robot. Sans en avoir l'air, la conception et l'usinage de pièces métalliques avec une précision et une solidité satisfaisantes prend énormément de temps et requiert un savoir-faire dont je ne disposais pas au début du projet. La partie usinage de la structure s'est étendue du mois d'Octobre au mois de Février (il a parfois fallu reprendre certaines pièces). À partir du mois de Décembre, j'ai commencé à étudier l'interface GPIO du Raspberry PI, et I2C (pour les ultrasons) en Janvier. On a réussi à ce moment à faire tourner des moteurs pas-à-pas et à détecter des distances aux ultrasons. Cependant,

les moteurs pas-à-pas étant trop petits (ils avaient été récupérés sur une imprimante de Minitel), on a dû les changer pour des moteurs plus gros. C'est là qu'il a fallu mettre au point la carte à puces des L293D, afin de donner plus de puissance aux moteurs. Les soudures (qu j'ai appris à faire ici) et le placement des fils électriques à pris énormément de temps. De plus il a aussi fallu usiner la carte à puces. Les moteurs pas-à-pas étaient fonctionnels en Mars. On s'est ensuite attelés au Servomoteur, et il m'a fallu une semaine avant de pouvoir le faire pivoter à ma guise. On avait entre temps acheté une batterie NIMH haut de gamme (4 000 mAh) et il fallait maintenant séparer le courant 7,2 V de la batterie en un circuit 5 V (Raspberry Pi, SRF08, servomoteur) et un circuit 7,2 V (moteurs pas-à-pas). C'est là qu'on a acheté le convertisseur à courant continu. La partie physique du robot était prête en Avril. Comme je n'avais que des morceaux de programmes correspondant à chaque « organe » du robot, j'ai mis au point un algorithme général, prêt au mois de Mai.

Difficultés rencontrées :

Les difficultés rencontrées lors de la réalisation de ce projet ont été nombreuses. Il s'agit à la fois de problèmes d'ordre matériel et logiciel.

Lorsque j'ai voulu faire ma première reconnaissance d'images, OpenCV ne trouvait pas la caméra du Raspberry Pi. Il fallait en fait charger le pilote de la caméra avec la commande *modprobe*. Pour que le pilote soit chargé à chaque démarrage, il faut éditer le fichier */etc/modules*.

Au départ, les moteurs pas-à-pas avaient été récupérés sur une imprimante de Minitel. Ils n'étaient pas du tout assez puissants pour déplacer un robot de presque 1,7 kg (la batterie y est pour beaucoup). J'ai donc utilisé à la place des moteurs Portescap plus gros et plus puissants.

Lorsque que le robot démarrait, les moteurs patinaient. J'ai d'abord réduit la vitesse (intervalle de 20 ms) pour augmenter le couple. Comme cela n'a pas suffit, j'ai ajouté autour des roues des élastiques pour ajouter de l'adhérence sur mon carrelage. Comme les élastiques ne tenaient pas autour des roues, j'ai ajouté du scotch double face.

Avant de le tester sur batterie, j'alimentais le robot avec un petit générateur portatif (7,2 V, intensité max 1 A). Cependant, lorsque le programme actionnait le servomoteur, la consommation de courant augmentait et le générateur ne fournissait plus assez de tension. La connexion SSH se coupait effaçant tous les fichiers en cours d'édition.

De manière générale, le robot consomme beaucoup trop et ce problème n'a toujours pas été résolu. Le Raspberry Pi à lui tout seul est déjà très gourmand en énergie. Il s'alimente via une prise USB comme les chargeurs de téléphone. Ces derniers, suivant la norme USB, délivrent du 5V jusqu'à 500mA (2,5W) voire 1A (5W) pour les « Quick charge ». Par comparaison, la prise d'alimentation USB du Raspberry Pi délivre du 5,1V jusqu'à 2,5A, soit 12,75W ! Je me suis rendu compte de cette forte consommation à ma première utilisation d'un Raspberry Pi. Je l'avais alimenté avec un chargeur de téléphone et il plantait (redémarrage) à chaque fois qu'il devenait gourmand en énergie, comme à l'écriture sur la carte SD. Cela corrompait au passage cette carte

SD. De manière générale, le robot est alimenté en 7,2 V. Il va consommer en général entre 0,6A (4,32W) et 1A (7,2W), avec des pics de consommation dépassant cette valeur. La batterie fournit du 4 000 mAh à 7,2V, soit de l'ordre de 100 KJ. Le robot devrait donc pouvoir tourner 5h en continu. Malheureusement, il va planter quand la tension passe sous un certain seuil, diminuant drastiquement l'autonomie.

Lorsque la balle entre en contact avec le capteur ultrasons, celui-ci ne reçoit plus les ondes ultrasonores émises. Il interprète donc des distances aberrantes. Bloquer la balle avant le capteur pourrait être une solution mais ce dernier va alors détecter le bloqueur. Le capteur ultrasons est cependant capable de repérer le 2ème écho (en fait jusqu'à 17 écho), mais ce dernier s'avère beaucoup moins précis que le premier.

De manière générales, la reconnaissance d'images est un des domaines les plus complexes de l'informatique, et l'algorithme ne peut jamais être parfait et universel.

Bilan et perspectives :

Améliorations possibles :

Il existe de nombreuses possibilités d'amélioration pour ce projet. Il peut s'agir d'améliorations logicielles ou matérielles.

Tout d'abord la reconnaissance d'images n'est jamais parfaite et peut toujours être améliorée. En effet l'algorithme a parfois du mal à détecter les balles. La modification pourrait se faire dans la gestion de la balance des blancs de la caméra, ou dans la détection automatique de la luminosité. Le robot pourrait aussi se calibrer automatiquement en fonction des balles.

Comme évoqué plus haut, la détection ultrasons laisse à désirer. Le robot pourrait considérer avoir atteint la balle au déclenchement d'un bouton.

On aurait également pu utiliser un détecteur laser, mais on a voulu partir d'éléments qu'on possédait déjà.

Lorsque le robot a déposé la balle, il recule pour pouvoir pivoter sans être gêné. Comme il n'y a aucun capteur à l'arrière, il est complètement aveugle lorsqu'il recule.

Apports personnels :

La réalisation de ce projet m'a apporté un enrichissement personnel et une plus grande connaissance dans certains domaines. En réalisant ce projet, je me suis notamment amélioré dans le domaine de la reconnaissance d'images que je ne maîtrisais presque pas.

J'y ai aussi découvert les principes du GPIO et de L'i2C, que je ne connaissais pas. En règle générale, j'ai appris beaucoup de choses dans le domaine de l'électronique.

J'ai pu tester le fonctionnement des ultrasons grâce au détecteur. Je sais maintenant comment détecter un objet et dans quelles conditions.

PRÉVOST Thomas TS3

J’ai appris de nouvelles choses à propos de Linux et en particulier les distributions Debian. Je sais maintenant créer un démon (service) ou encore charger des pilotes.

Je sais maintenant souder des pièces électroniques, sertir des fils électriques. J’ai en outre plus de maîtrise quant à l’utilisation d’une fraiseuse.

Remerciements :

Mon père, qui m’a beaucoup aidé pour la partie physique du robot, qui m’a appris à souder et à sertir des câbles, et qui a conçu une bonne partie des cartes électroniques.
Ma mère et ma sœur, parce que le salon a été encombré pendant la réalisation.
Mr Martin

Merci pour votre lecture
Thomas Prévost

Si le robot ne fonctionne pas pendant la démonstration : https://www.youtube.com/playlist?list=PLh3qs8oBmtgHSlz46Wd8SXoNHdjiqC29y

Annexes :

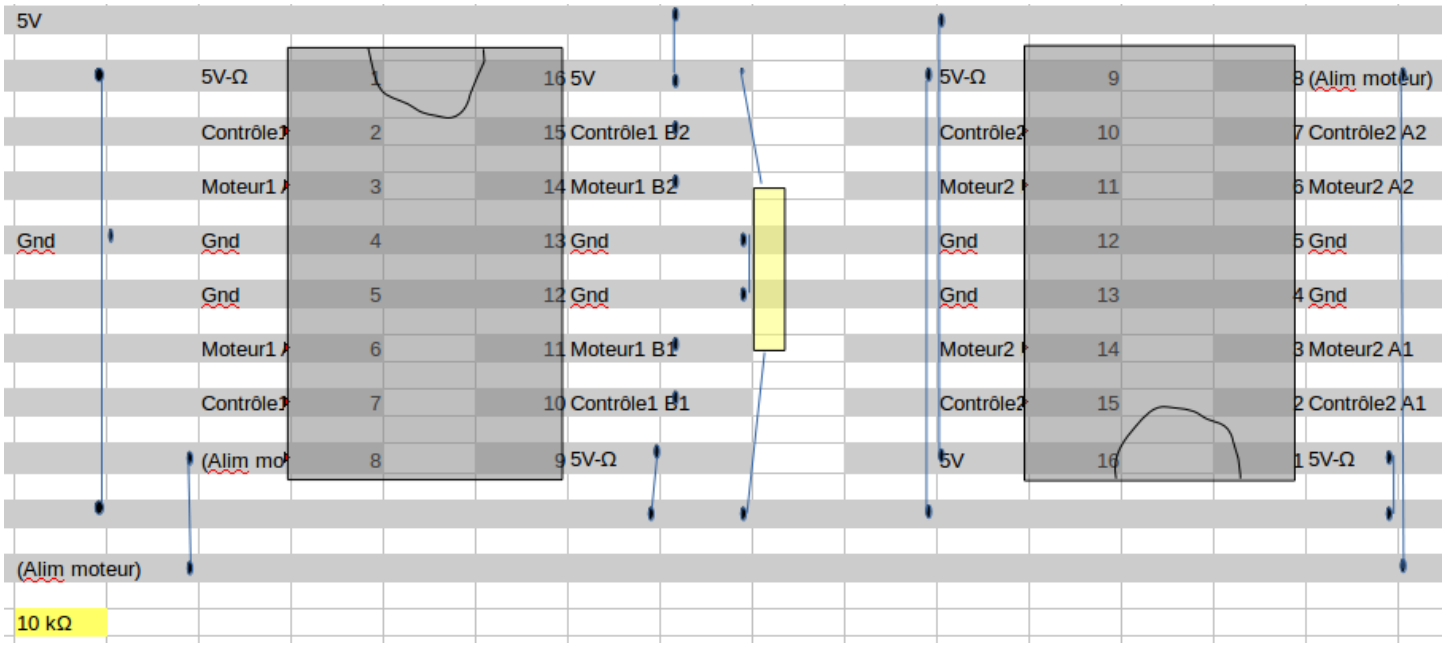
Tableau des différentes affectations des contacts électriques :

Nom	Affectation sur 25p	GPIO (numéros physiques)
7,2 V régulateur	20	
5 V Raspberry Pi	21	
Alim 5 V depuis Raspberry Pi	1	2
// Ground	13	9
Alim moteurs (pas-à-pas)	7	
Ground moteurs régulateur	8	6
Contrôle moteur 1 A1	2	7
Contrôle moteur 1 A2	3	11
Contrôle moteur 1 B1	4	12
Contrôle moteur 1 B2	5	13
Contrôle moteur 2 A1	9	15
Contrôle moteur 2 A2	10	16
Contrôle moteur 2 B1	11	18
Contrôle moteur 2 B2	12	22
Unable	18	29

I2C SDA	25	3
I2C SCL	24	5
I2C Alim 5 V	23	4
I2C Ground	22	6
GPIO servomoteur	6	31

Schéma de la carte à puces L293D :

Pour ne pas avoir à créer un circuit imprimé, on a utilisé un circuit à bande pré-imprimée qu'il suffit de découper. La conception de ce type de circuit peut se faire simplement en utilisant un tableur type Excel. Les bandes grises correspondent aux pistes de cuivre.



http://meayne.free.fr/isn/Plan-carte-puce-l293d.xlsx

Algorithme (susceptible d'être modifié avant la présentation du projet) :

prog.cpp

```
#include <iostream>
#include "detecttennis.h"
#include "dcmotor.h"
#include "US-I2C_servo.h"

using namespace std;

int main()
{
   DetectTennis detection1; // Constructeur
   DcMotor moteurs; // Constructeur
   vector<pair<int, int>> listeBalles;
   unsigned short dist, distTotal(0);
```

```
    servoCommute(true); // On verifie que la partie mobile soit en haut
    while(1) // boucle infinie
    {
        debutDetection:
        if(detection1.detect()) // Si une balle est detectee
        {
            listeBalles=detection1.listBalles();
            sort(listeBalles.begin(), listeBalles.end(), DetectTennis::pairCompare); // On trie les balles par ordre
décroissant de distance
            moteurs.pivot(listeBalles[0].second); // On s'aligne avec la balle la plus proche
            if((listeBalles[0].first > 150) && (I2Cdetect(3)>100)) // Si la balle est loin
            {
                moteurs.marche(20); // On avance de 20 cm
                distTotal+=20;
                goto debutDetection; // On recommence la detection
            }
            do // tant que la balle est a plus de 10 cm
            {
                dist=I2Cdetect(3)-10;
                cout << dist << endl;
                if(dist > 100) // Si il y a une erreur dans la detection
                {
                    moteurs.marche(-10); // On recule de 10 cm pour recommencer la detection
                    distTotal-=10;
                }
                else
                {
                    moteurs.marche(dist); // On va sur la balle
                    distTotal+=dist;
                }
            }while(dist>10);
            servoCommute(false); // On attrape la balle
            moteurs.marche(-distTotal); // On recule de la distance totale parcourue
            servoCommute(true); // On lache la balle
            distTotal=0;
            moteurs.marche(-20); // On recule de 20 cm
            moteurs.pivot(30); // On pivote de 30° pour trouver une autre balle
        }
        else
        {
            moteurs.pivot(30); // On pivote de 30° pour trouver une autre balle
        }
    }
    return 0;
}
```

dcmotor.h

```
#ifndef DCMOTOR_H_INCLUDED
#define DCMOTOR_H_INCLUDED

#include <iostream>
#include <wiringPi.h>
```

PRÉVOST Thomas TS3

```
/** Compiler avec l'option -lwiringPi */

class DcMotor
{
public:
   DcMotor();
   ~DcMotor();
   void marche(short dist); // On avance / recule
   void pivot(short deg); // On pivote

private:
   void prochainStep(unsigned short numero, unsigned short pin[]); // Les impulsions donnees aus moteurs
   void reglage_pins(unsigned short p1, unsigned short p2, unsigned short p3, unsigned short p4, unsigned short
pin[]); // 1/4 d'impulsion

   unsigned short pin1[4]={7, 0, 1, 2}; // Les pins GPIO des 2 moteurs
   unsigned short pin2[4]={3, 4, 5, 6};
   unsigned short unable; // Pin GPIO pour arreter l'alimentation des moteurs
};

#endif // DCMOTOR_H_INCLUDED
```

dcmotor.cpp

```
#include "dcmotor.h"

using namespace std;

DcMotor::DcMotor()
{
   unable=21;
   wiringPiSetup(); // On initialise la bibliotheque WiringPi
   pinMode(pin1[0], OUTPUT); // On regle les pins en mode sortie
   pinMode(pin1[1], OUTPUT);
   pinMode(pin1[2], OUTPUT);
   pinMode(pin1[3], OUTPUT);
   pinMode(pin2[0], OUTPUT);
   pinMode(pin2[1], OUTPUT);
   pinMode(pin2[2], OUTPUT);
   pinMode(pin2[3], OUTPUT);
   pinMode(unable, OUTPUT);
}

DcMotor::~DcMotor()
{
}

void DcMotor::marche(short dist)
{
   // 1 tour = 200 pas = 18,85 cm
   // delai 5 ms -> 18,85 cm/s
   // 1 pas=0.0943 cm
   int nbPas;
   unsigned int delai=20; // Delai entre chaque pas : 20 ms (change la vitesse)
```

```
    digitalWrite(unable, 1); // On alimente les moteurs
    if(dist >= 0) // On avance
    {
        nbPas=dist/0.0943;
        for(unsigned int i=0; i<nbPas; i++)
        {
            prochainStep(3 - (i % 4), pin1); // les moteurs 1 et 2 sont symetriques, donc inverses
            prochainStep(i % 4, pin2);
            delay(delai);
        }
    }
    else // On recule
    {
        nbPas=(-dist)/0.0943;
        for(unsigned int i=0; i<nbPas; i++)
        {
            prochainStep(i % 4, pin1); // les moteurs 1 et 2 sont symetriques, donc inverses
            prochainStep(3 - (i % 4), pin2);
            delay(delai);
        }
    }
    digitalWrite(unable, 0); // On enleve l'alimentation des moteurs
}

void DcMotor::pivot(short deg)
{
    // dist 360°: 17 cm * pi = 53,407075111 cm
    int nbPas;
    unsigned int delai=20;
    digitalWrite(unable, 1); // On alimente les moteurs
    if(deg >= 0)
    {
        nbPas=deg*1.5732;
        for(unsigned int i=0; i<nbPas; i++)
        {
            prochainStep(i % 4, pin1);
            prochainStep(i % 4, pin2);
            delay(delai);
        }
    }
    else
    {
        nbPas=(-deg)*1.5732;
        for(unsigned int i=0; i<nbPas; i++)
        {
            prochainStep(3 - (i % 4), pin1);
            prochainStep(3 - (i % 4), pin2);
            delay(delai);
        }
    }
    digitalWrite(unable, 0); // On enleve l'alimentation des moteurs
}

void DcMotor::prochainStep(unsigned short numero, unsigned short pin[])
```

```
{
switch(numero)  // fonctionnement « pas complets »
   {
   case 0:
      reglage_pins(1, 0, 0, 0, pin);
      break;
   case 1:
      reglage_pins(0, 0, 1, 0, pin);
      break;
   case 2:
      reglage_pins(0, 1, 0, 0, pin);
      break;
   case 3:
      reglage_pins(0, 0, 0, 1, pin);
      break;
   }
}

void DcMotor::reglage_pins(unsigned short p1, unsigned short p2, unsigned short p3, unsigned short p4,
unsigned short pin[])
{
   digitalWrite(pin[0], p1);
   digitalWrite(pin[1], p2);
   digitalWrite(pin[2], p3);
   digitalWrite(pin[3], p4);
}
```

detecttennis.h

```
#ifndef DETECTTENNIS_H_INCLUDED
#define DETECTTENNIS_H_INCLUDED

#include <iostream>
#include <vector>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

/** Compiler avec les options -lopencv_core -lopencv_highgui -lopencv_imgproc */

class DetectTennis
{
public:
   DetectTennis();
   ~DetectTennis();
   bool detect();
   std::vector<std::pair<int, int>> listBalles();
   static bool pairCompare(const std::pair<int, int>&firstElem, const std::pair<int, int>&secondElem); // On trie
les balles par ordre décroissant de distance

private:
   void cvOpen(cv::Mat src, cv::Mat dst, cv::Mat element); // Adoucissement des traits
   void cvClose(cv::Mat src, cv::Mat dst, cv::Mat element);
   cv::Mat rotateArr(cv::Mat src, double angle); // On pivote l'image
```

```
    cv::VideoCapture cap; // Prendre une photo
    unsigned short erreur; // Une eventuelle erreur
    std::vector<cv::Vec3f> circles; // Liste des balles
    CvSize imgsize; // Taille de l'image
};

#endif // DETECTTENNIS_H_INCLUDED
```

detecttennis.cpp

```
/* Mat=tableau de pixels
   IplImage=image chargee en memoire */

#include "detecttennis.h"

using namespace cv; // Specifique a OpenCV
using namespace std;

DetectTennis::DetectTennis()
{
   cap=0;
   erreur=0;
   if(!cap.isOpened()) // Erreur d'ouverture de la camera (pas branchee, pilote non charge...)
   {
      cout << "Impossible d'ouvrir la camera !" << endl;
      erreur=1;
   }
}

DetectTennis::~DetectTennis()
{
}

bool DetectTennis::detect()
{
   Mat img;
   cap >> img; // On prend la photo
   img=rotateArr(img, 180); // La camera est a l'envers donc on pivote de 180°
   imgsize = img.size(); // On regarde la taille de l'image

   IplImage *hsv = cvCreateImage(imgsize, IPL_DEPTH_8U, 3);
   cvtColor(img, (Mat)hsv, CV_BGR2HSV);
   Mat mask = cvCreateMat(imgsize.height, imgsize.width, CV_8UC1); // Masque : seconde image avec en
blancs les pixels qui etaient dans une certaine fourchette et en noir les autres
   inRange((Mat)hsv, Scalar(23,41,133), Scalar(40,150,255), mask); // La fourchette en question

   Mat se21 = getStructuringElement(MORPH_RECT, Size(21, 21), Point(10, 10));
   Mat se11 = getStructuringElement(MORPH_RECT, Size(11, 11), Point(5, 5));
   cvClose(mask, mask, se21); // Traitements techniques necessaires a l'algorithme
   cvOpen(mask, mask, se11);

   IplImage *hough_in = cvCreateImage(imgsize, 8, 1);
   mask.copyTo((Mat)hough_in);
```

```
    GaussianBlur((Mat)hough_in, (Mat)hough_in, Size(15, 15), 0, 0); // On lisse le masque grace a une fonction gaussienne

    HoughCircles((Mat)hough_in, circles, CV_HOUGH_GRADIENT, 2, imgsize.height/10, 81, 29, imgsize.height/200, 0); // On detecte les cercles

    img.release(); // On detruit les variables pour faire de la place en memoire
    mask.release();
    cvReleaseImage(&hsv);
    cvReleaseImage(&hough_in);

    if(circles.size()==0) // Pas de balle trouvee
    {
        return false;
    }
    else // Une balle trouvee
    {
        return true;
    }
}

vector<pair<int, int>> DetectTennis::listBalles()
{
    unsigned int i;
    float distance;
    int deg;
    vector<pair<int, int>> vectorRetour(circles.size()); // Voir les conteneurs en C++
    for(i = 0; i < circles.size(); i++)
    {
        deg = ((circles[i][0] – ((imgsize.width)/2))/imgsize.width)*30; // angle
        distance = 3.2/tan(circles[i][2]*0.942477796/imgsize.width); // distance par trigonometrie
        vectorRetour[i]=make_pair(distance, deg);
    }
    return vectorRetour;
}

void DetectTennis::cvOpen(Mat src, Mat dst, Mat element)
{
    erode(src, dst, element);
    dilate(src, dst, element);
}

void DetectTennis::cvClose(Mat src, Mat dst, Mat element)
{
    dilate(src, dst, element);
    erode(src, dst, element);
}

Mat DetectTennis::rotateArr(Mat src, double angle)
{
    Mat dst;
    Point2f pt(src.cols/2., src.rows/2.);
    Mat r = getRotationMatrix2D(pt, angle, 1.0);
    warpAffine(src, dst, r, Size(src.cols, src.rows));
```

```
    return dst;
}

bool DetectTennis::pairCompare(const pair<int, int>&firstElem, const pair<int, int>&secondElem)
{
    return firstElem.first < secondElem.first;
}
```

US-I2C_servo.h

```
#ifndef USI2CSERVO_H_INCLUDED
#define USI2CSERVO_H_INCLUDED

#include <iostream>
#include <vector>
#include <algorithm>
#include <wiringPi.h>
#include <wiringPiI2C.h>

/** Compiler avec l'option -lwiringPi */

unsigned short I2Cdetect(unsigned short nbMesures); // Detection ultrasons
void servoCommute(bool haut); // Changement de position du servomoteur

#endif // USI2CSERVO_H_INCLUDED
```

US-I2C_servo.cpp

```
#include "US-I2C_servo.h"

using namespace std;

unsigned short I2Cdetect(unsigned short nbMesures)
{
    unsigned short adress=0x70, bus=1; // L'adresse constante du SRF08 et le numero du bus I2C
    int fd;
    vector<int> dist(nbMesures);
    fd=wiringPiI2CSetup(adress); // On initialise la bibliotheque WiringPi
    if(fd == -1) // Si erreur I2C (pas branche, mauvaise adresse...)
    {
        cout << "Erreur initialisation I2C" << endl;
        return -1;
    }
    unsigned short i;
    for(i=0; i<nbMesures; i++) // On peut faire plusieurs mesures pour plus de precision
    {
        wiringPiI2CWriteReg8(fd, 0, 0x51); // On donne l'unite avec laquelle encoder la reponse (cm), l'actualise
        delay(500); // On patiente pour que le SRF08 effectue la mesure
        dist[i]=((wiringPiI2CReadReg8(fd, 2)*256) + wiringPiI2CReadReg8(fd, 3)); // On lit la distance (2 octets)
    }
    sort(dist.begin(), dist.end()); // On trie les distances par ordre croissant
    return dist[dist.size()/2]; // On recupere la valeur mediane
}
```

PRÉVOST Thomas TS3

```
void servoCommute(bool haut)
{
   wiringPiSetup(); // On initialise la bibliotheque WiringPi
   unsigned short pinServo(22), temps; // Pin de contrôle du servomoteur
   if(haut) // Si on doit lever
   {
      temps=1000;
   }
   else // Si on doit baisser
   {
      temps=2500;
   }
   pinMode(pinServo, OUTPUT); // Pin en mode sortie
   for(unsigned short i(0); i<35; i++) // Signal carre « en creneaux », le servomoteur fait 180° en 20*35ms=0,7s
   {
      digitalWrite(pinServo, 1);
      delayMicroseconds(temps);
      digitalWrite(pinServo, 0);
      delayMicroseconds(20000-temps); // Periode : 20 ms c-a-d 20 000µs
   }
}
```

Compiler le programme avec :

$ g++ -o progRobot prog.cpp detecttennis.cpp dcmotor.cpp US-I2C_servo.cpp -lopencv_core -lopencv_highgui -lopencv_imgproc -lwiringPi

/etc/modules :

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.

I2c-dev                 # bus I2c
bcm2835-v4l2            # camera Raspberry Pi
```

/etc/init.d/progRobot

```
#!/bin/sh -e

### BEGIN INIT INFO
# Provides:          progRobot
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Lancement robot
# Description:       Demon de lancement automatique du robot
### END INIT INFO

DESC="Demon de lancement automatique du robot"
PIDFILE=/var/run/progRobot.pid
SCRIPTNAME=/etc/init.d/progRobot
DAEMON="/opt/progRobot" # On aura copie progRobot dans le repertoire /opt/ avec les droits execution
```

PRÉVOST Thomas TS3

```
DAEMONUSER="root" #utilisateur du programme
daemon_NAME="progRobot" #Nom du programme (doit être identique à l'exécutable).

PATH="/sbin:/bin:/usr/sbin:/usr/bin"

test -x $DAEMON || exit 0

. /lib/lsb/init-functions

d_start () {
	log_daemon_msg "Starting system $daemon_NAME Daemon"
	   start-stop-daemon --background --name $daemon_NAME --start --quiet --chuid $DAEMONUSER --
exec $DAEMON
	log_end_msg $?
}

d_stop () {
	log_daemon_msg "Stopping system $daemon_NAME Daemon"
	start-stop-daemon --name $daemon_NAME --stop --retry 5 --quiet --name $daemon_NAME
	   log_end_msg $?
}

case "$1" in

	start|stop)
		d_${1}
		;;

	restart|reload|force-reload)
			d_stop
			d_start
		;;

	force-stop)
		d_stop
		killall -q $daemon_NAME || true
		sleep 2
		killall -q -9 $daemon_NAME || true
		;;

	status)
		status_of_proc "$daemon_NAME" "$DAEMON" "system-wide $daemon_NAME" && exit 0 || exit
$?
		;;
	*)
		echo "Usage: /etc/init.d/$daemon_NAME {start|stop|force-stop|restart|reload|force-reload|status}"
		exit 1
		;;
esac
exit 0
```

Démoniser le programme :

 # chmod 0755 /etc/init.d/progRobot
 # systemctl daemon-reload

PRÉVOST Thomas TS3

 # update-rc.d progRobot defaults
Démarrer : *# service progRobot start*
Arrêter : *# service progRobot stop*

Plan du robot dessiné sous TurboCAD platinum 18.2