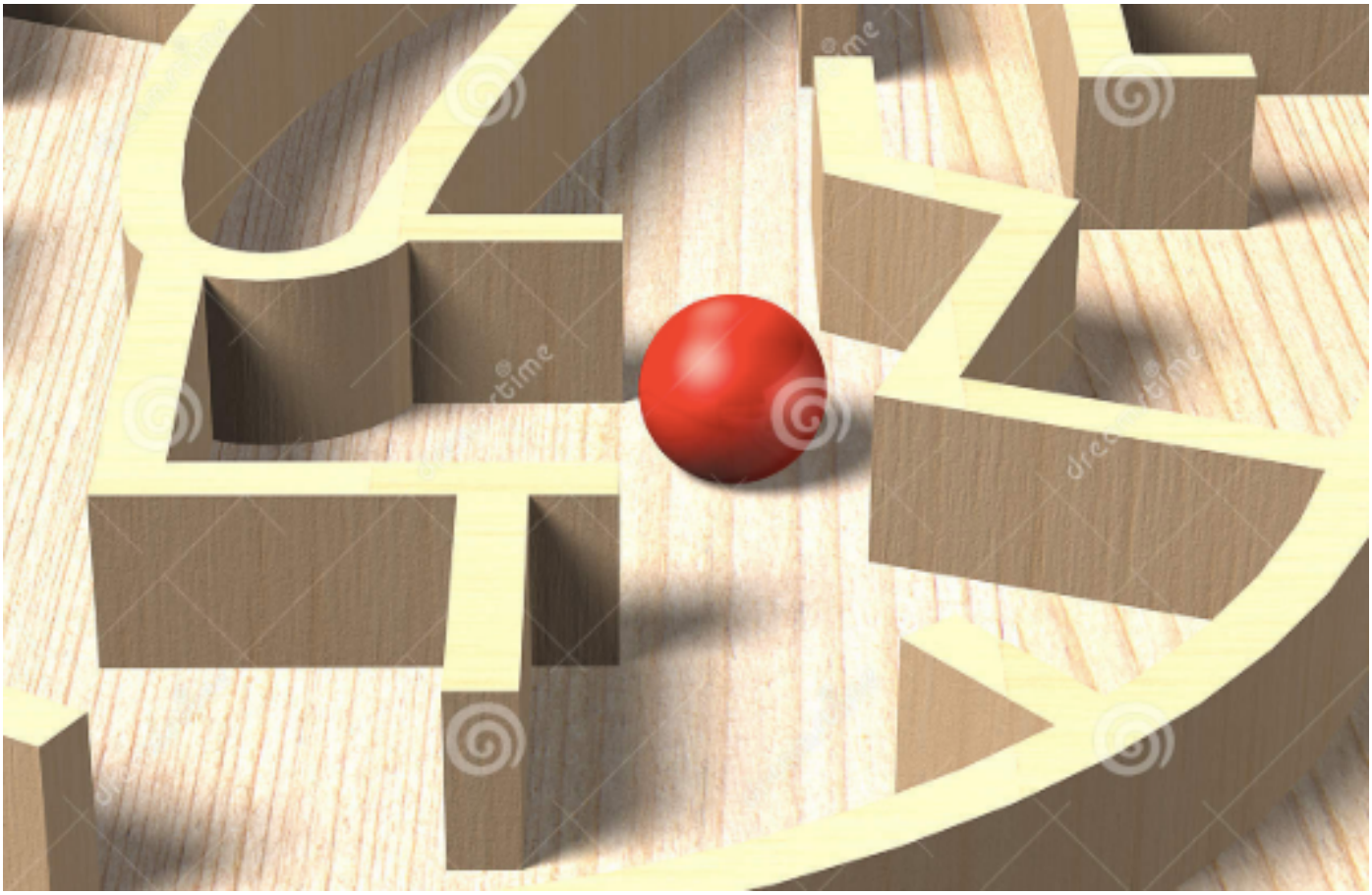
Le rapport final de projet fin d'étude

Labyrinthe à bille autonome



Encadrant : M.Blaise CONRARD

Étudiante : YAN Xuelu

Année : 2019-2020

Sommaire

Introduction

L'objectif du projet est de créer un labyrinthe dans lequel une bille doit suivre une trajectoire de la surface. L'un des systèmes de contrôle assure le mouvement de la bille grâce au contrôle de la pente de la surface.

Pour cette réalisation, il faut d'abord créer de l'équipement à la découpe laser. Et ensuite on utilise d'un Arduino comme système de traitements pour commander le système. Pour contrôler la trajectoire de la bille, il faut détecter ses coordonnées tout le temps. Donc on utilise d'une caméra comme capteur de position Enfin on utilise de servomoteurs pour le contrôle de l'inclinaison du plan et du mouvement de la bille.

Dans la première moitié de mes études, j'ai créé l'équipement labyrinthe et déterminer le schéma mécanique. Puis j'ai testé la caméra et les moteurs avec l'Arduino. Dans ce cas, j'ai dessiné le schéma électronique dans le logiciel Fritzing.
Pour la seconde moitié du travail, je vais finir la partie électronique et après je vais travailler sur la partie programme. Il faut utiliser le contrôle PI pour contrôler servos. Enfin, je vais faire la fixation des moteurs et faire l'assemblage de tous équipement

Présentation du projet

Cahier des charges

Contexte

C'est le projet de IMA5. Ce projet vise à réaliser une démonstration où une bille parcourt un labyrinthe. Il doit se déplacer en douceur vers la position désignée en fonction de l'itinéraire spécifié. Une fois le projet terminé, il sera exposé lors de la journée portes ouvertes.

Objectif

L'objectif du projet est de créer un labyrinthe dans lequel une bille doit suivre une trajectoire de la surface. L'un des systèmes de contrôle assure le mouvement de la bille grâce au contrôle de la pente de la surface.

Fonctionnement

1. La bille se déplace en douceur

La bille doit éviter les obstacles et se déplacer en douceur dans le labyrinthe.

2. Déterminer les coordonnées de la bille

Pendant le mouvement de la bille, le système doit surveiller ses informations de position en temps réel et obtenir les coordonnées de la bille.

3. La bille se déplace selon la trajectoire spécifiée

Nous utilisons le programme pour définir la trajectoire de la bille. La bille peut répéter le mouvement en fonction de la trajectoire spécifié.

Ressources

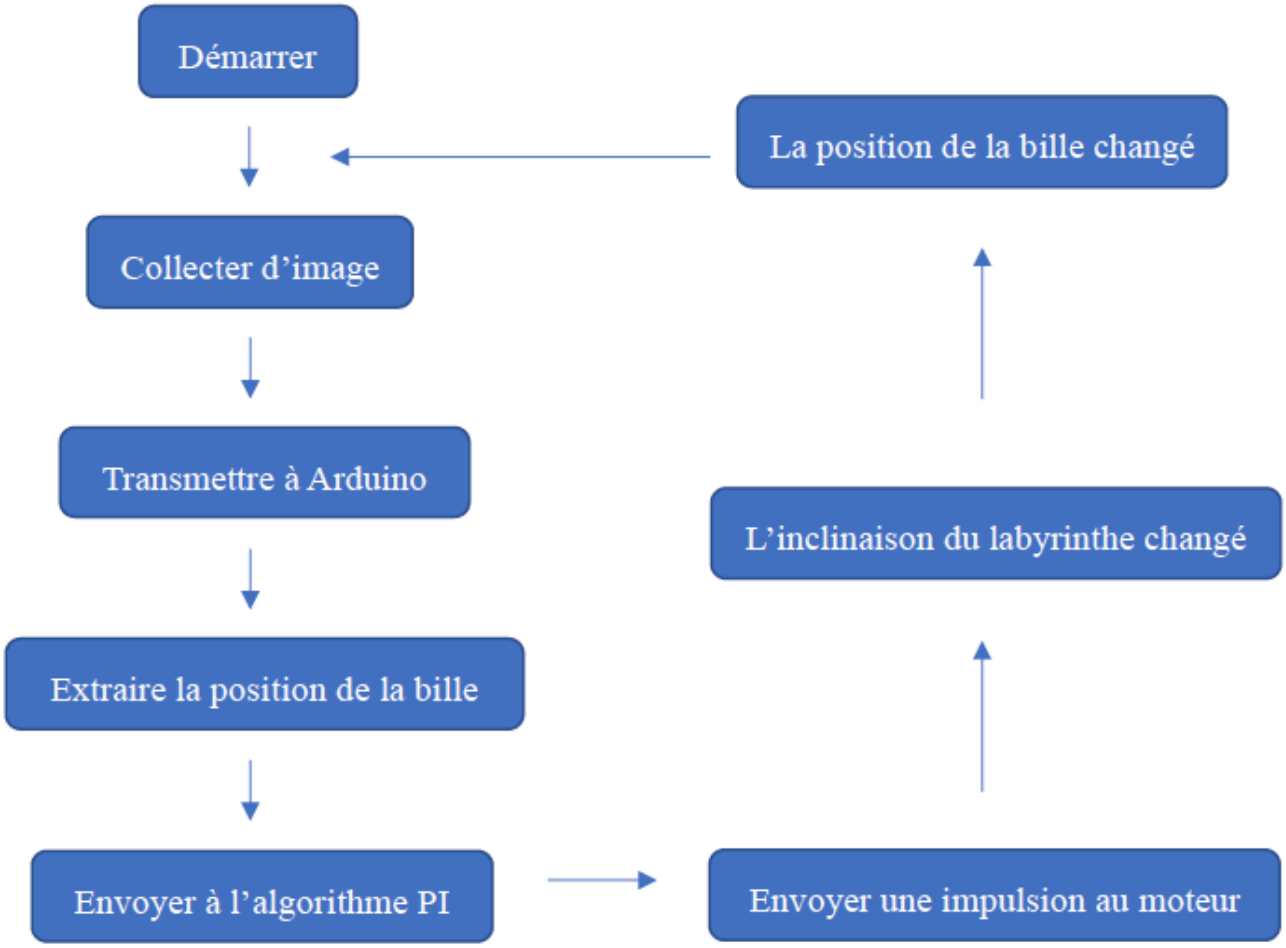
Sous la direction de mon tuteur Blaise Conrard, ce système est réalisé par moi personnellement. Le matériel informatique, les composants et le support technique requis dans ce système sont fournis par l'école.

Delai

Il est prévu de mettre en place le fonctionnement et l'utilisation du système au début du mois de février 2020. La date de livraison est fin février 2020.

Diagramme de processus

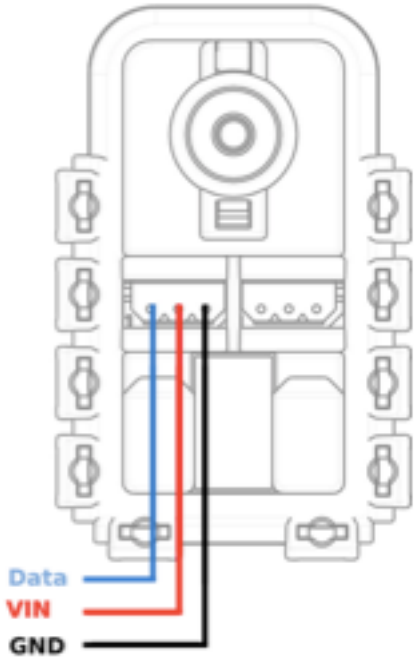
Avant de commencer ce projet j'ai cherché les informations sur ce projet et enfin j'ai dessiné le diagramme de processus.



Après le démarrage, la caméra commence à acquérir des images en temps réel. Les informations de coordonnées collectées par la caméra seront transmises à Arduino pour traitement. Les informations de position traitées sont ensuite transmises à un algorithme PI, on l'utilise pour contrôler la rotation du servomoteur. Par conséquent, l'inclinaison de la plaque de labyrinthe et le changement de la position de la petite bille sont contrôlés, de sorte que la bille peut répéter le mouvement en fonction de la trajectoire spécifié.

Choix du matériel

Servomoteurs



Principe :

Le système servo est un système de contrôle automatique qui permet à une quantité de sortie contrôlée d'un objet (comme la position, l'orientation, le statut...) de suivre un changement arbitraire d'une cible d'entrée. Le servomoteur est principalement positionné par impulsions. Lorsque le servomoteur reçoit une impulsion, il fait pivoter l'angle correspondant à une impulsion pour obtenir un déplacement. Le servomoteur lui-même a pour fonction d'émettre une impulsion. Ainsi, chaque fois que le servomoteur tourne d'un angle, un nombre correspondant d'impulsions est émis. De cette manière, l'impulsion reçue par le servomoteur forme un écho ou une boucle fermée. En conséquence, le système sait combien d'impulsions ont été envoyées au servomoteur et combien d'impulsions ont été reçues. De cette manière, la rotation du moteur peut être contrôlée avec une très grande précision, ce qui permet d'obtenir un positionnement précis pouvant atteindre 0,001 mm.

Avantage:

1. Précision : réalise un contrôle en boucle fermée de la position, de la vitesse et du couple, résout le problème du moteur pas à pas en déséquilibre;

2, Vitesse : la performance à haute vitesse est bonne, la vitesse nominale générale peut atteindre 2000 ~ 3000 tr / min;

3. Adaptabilité : il possède une forte capacité anti-surcharge et peut résister à trois fois le couple nominal. Il est particulièrement adapté aux occasions avec des fluctuations de charge transitoires et un démarrage rapide.

4. Stabilité : Il fonctionne sans heurts à basse vitesse et ne produit pas d'opération pas à pas similaire à celle d'un moteur pas à pas lorsqu'il tourne à basse vitesse. Convient pour les occasions avec des exigences de réponse à grande vitesse;

5. Rapidité d'exécution : La dynamique d'accélération et de décélération motrices est courte, généralement en quelques dizaines de millisecondes.

6. Confort : la chaleur et le bruit sont considérablement réduits.

Caméra Pixy

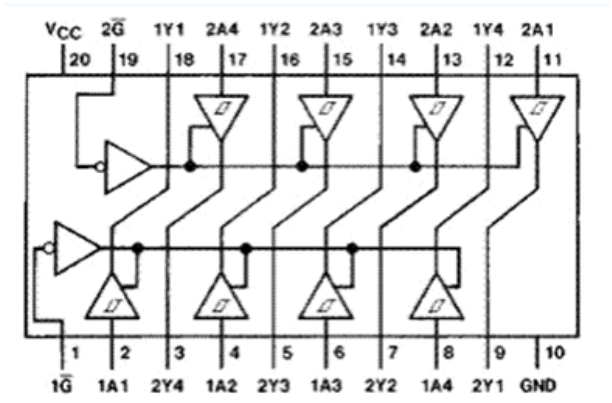
Fonctions :

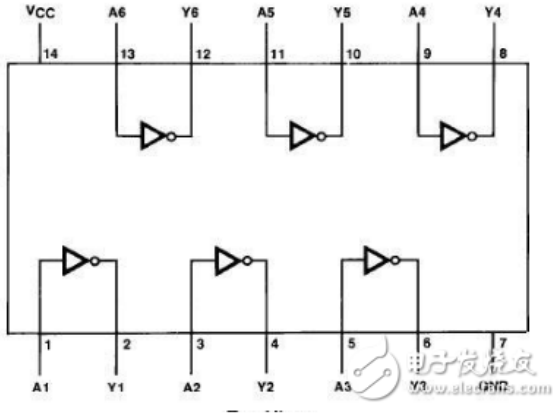
Pixy est un capteur de reconnaissance d'image open source qui prend en charge la reconnaissance de couleur multi-objet et multicolore et prend en charge jusqu'à 7 couleurs. Vous pouvez lui dire la couleur que vous voulez et lui apprendre à trouver quelque chose. Pixy prend en charge plusieurs méthodes de communication, telles que SPI, I2C, etc., qui peuvent être directement connectées à la carte de commande Arduino. Installez-le sur votre robot et ajoutez une paire d'yeux à votre petit robot. Il est équipé d'un capteur d'image doté d'un matériel puissant pouvant être utilisé avec un PC pour suivre et analyser des données multicolores. Pixy peut communiquer directement avec l'Arduino. Il envoie des informations de bloc à l'Arduino à 1 Mbit / s, ce qui signifie que le Pixy peut envoyer plus de 6 000 objets identifiés par seconde ou 135 objets identifiés par frame (Pixy peut traiter 50 images par seconde).

Communications :

Pas besoin de jouer avec de minuscules fils - Pixy est livré avec un câble spécial pour se connecter directement à un Arduino et un câble USB pour se connecter à un Raspberry Pi, afin que vous puissiez commencer rapidement. Pixy possède plusieurs interfaces (SPI, I2C, UART et USB) et des communications simples, de sorte que votre contrôleur choisi parle à Pixy en peu de temps.

74LS244N & 74HC04N





`

74LS244N :

Description :

74LS244N est un buffer à huit voies à trois états.
Ces tampons octaux et pilotes de ligne sont spécialement conçus pour améliorer à la fois les performances et la densité des pilotes d'adresse mémoire à trois états, des pilotes d'horloge et des récepteurs et transmetteurs orientés bus. Le concepteur dispose d'un choix de combinaisons sélectionnées de sorties inversées et non inversées, d'entrées symétriques de contrôle de sortie actif (G) et d'entrées de contrôle de sortie complémentaires (G et G). Ces appareils présentent un fort fan-out, un fan-in amélioré et une marge de bruit de 400 mV. Les appareils SN74LS et SN74S peuvent être utilisés pour conduire des lignes terminées vers 133 Ohm.

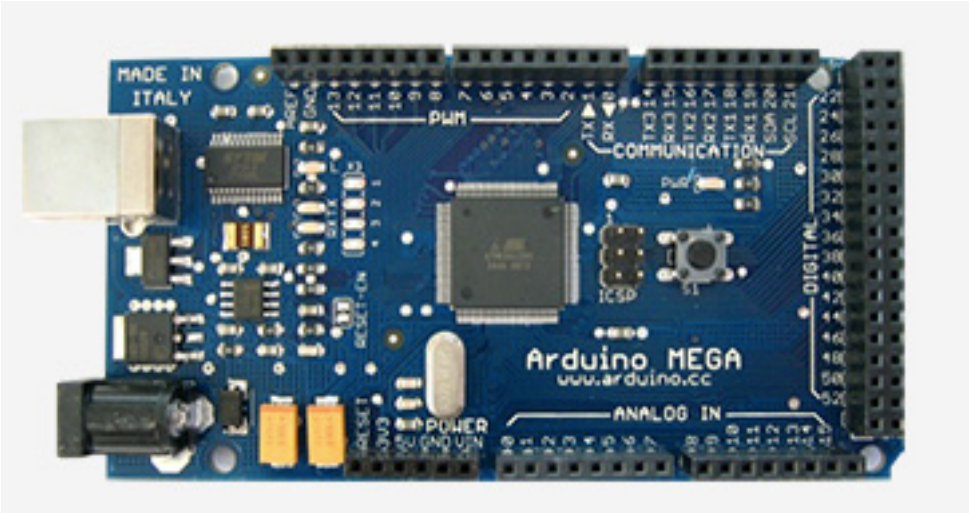
Fonctions :

Dans ce projet, le composant 74LS244N peut établir une communication entre le servomoteur et l'Arduino. Après la connexion par lui, on peut contrôler la rotation du moteur directement via Arduino.

74HC04N :

Les appareils 74HC04N contiennent six inverseurs indépendants. Ils exécutent la fonction booléenne Y = $\bar{A}$ en logique positive.
Ce composant est pour inverser la commande.

Arduino Mega



Présentation :

L'Arduino Mega est une carte microcontrôleur basée sur l'ATmega1280 (fiche technique). Il dispose de 54 broches d'entrée / sortie numériques (dont 14 peuvent être utilisées comme sorties PWM), 16 entrées analogiques, 4 UART (ports série matériels), un oscillateur à cristal de 16 MHz, une connexion USB, une prise d'alimentation, un en-tête ICSP, et un bouton de réinitialisation. Il contient tout le nécessaire pour supporter le microcontrôleur ; connectez-le simplement à un ordinateur avec un câble USB ou alimentez-le avec un adaptateur AC-DC ou une batterie pour commencer.

Communication :

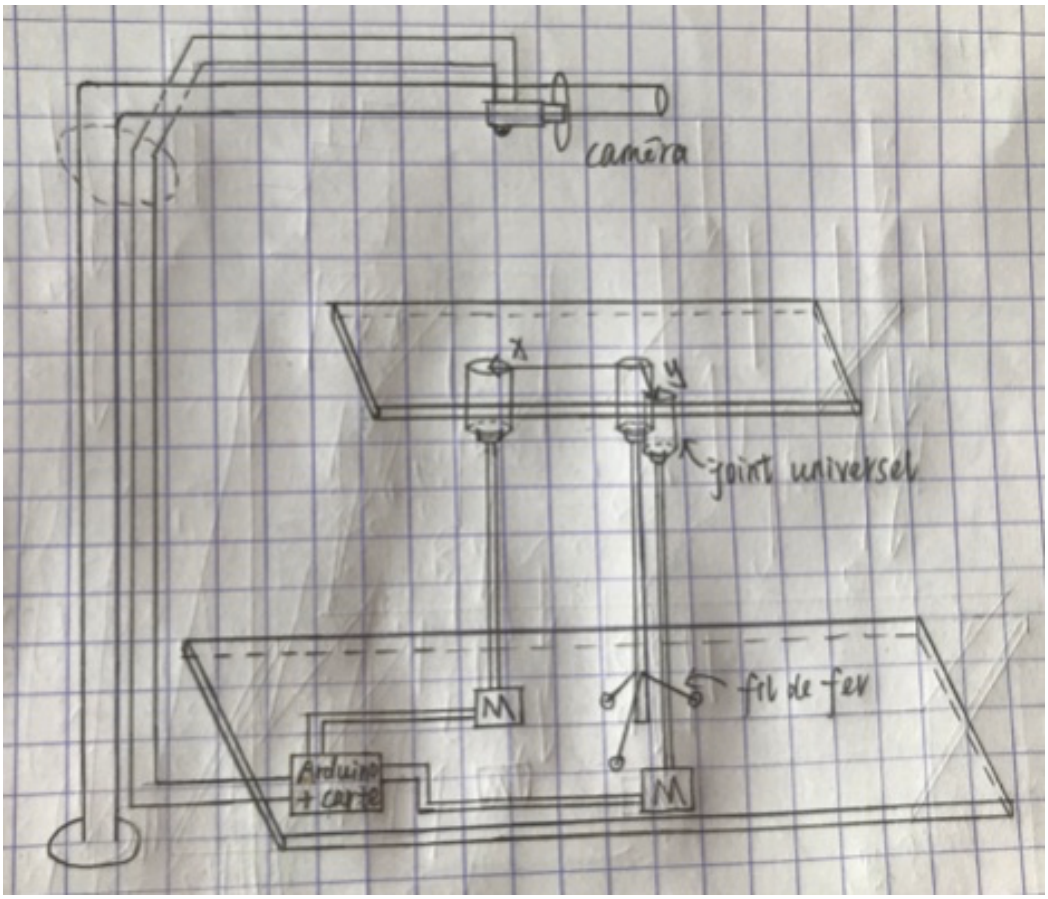
L'Arduino Mega dispose d'un certain nombre d'installations pour communiquer avec un ordinateur, un autre Arduino ou d'autres microcontrôleurs. L'ATmega1280 fournit quatre UART matériels pour la communication série TTL (5V). Un FTDI FT232RL sur la carte canalise l'un d'entre eux via USB et les pilotes FTDI (inclus avec le logiciel Arduino) fournissent un port de communication virtuel aux logiciels de l'ordinateur. Le logiciel Arduino comprend un moniteur série qui permet d'envoyer des données textuelles simples vers et depuis la carte Arduino. Les LED RX et TX de la carte clignotent lorsque des données sont transmises via la puce FTDI et la connexion USB à l'ordinateur (mais pas pour la communication série sur les broches 0 et 1).

Réalisation du projet

Partie mécanique

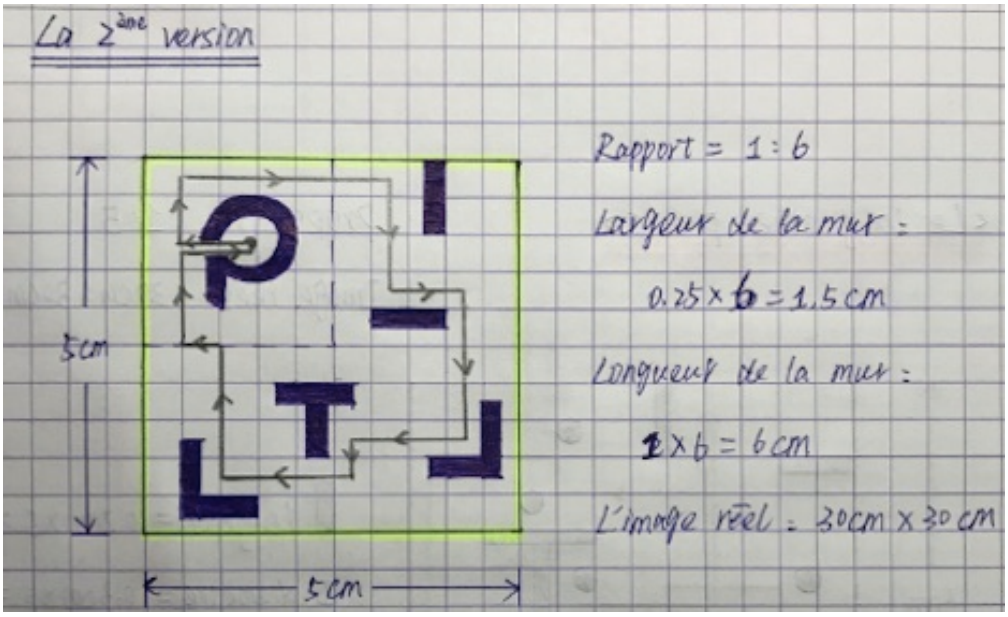
Design du schéma mécanique

Tout d'abord, il y a quatre moteurs en bas pour contrôler la direction. Fixez une tige au culbuteur de chaque moteur. La rotation du moteur fait bouger la tige d'haut en bas pour contrôler l'inclinaison de la planche. J'utilise 4 moteurs pour stabiliser la structure, et parce que la bille se déplace en ligne droite, chaque fois que l'inclinaison de la planche doit être contrôlée par deux moteurs du même côté. Enfin, il faut fixer la caméra sur une tige utilitaire directement au-dessus de la carte. La caméra et les pièces de connexion Arduino sont également fixées à la tige. Cela garantira la stabilité de la caméra.

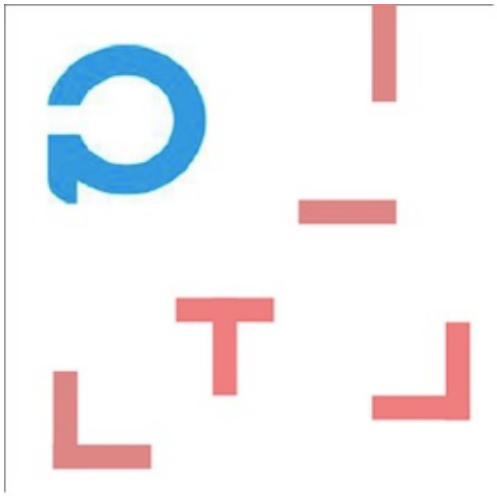


Le labyrinthe

Tout d'abord, j'ai dessiné le labyrinthe. Dans ma conception, je règle la taille du labyrinthe (plache du haut) à 30 cm x 30 cm. La taille de la planche du bas est de 40 cm x 40 cm. Je mets quelques obstacles, le largeur est 1.5cm et la longueur est 6cm. J'ai dessiné la trajectoire de la bille. La bille va partir du symbole. Et elle va répéter le mouvement en fonction de la trajectoire spécifié.



Après j'ai déterminé le dessin en papier, je l'ai crée dans le logiciel 'Inkscape'. Ensuite, j'ai réservé la machine 'Découpeuse Laser' pour faire ma planche de labyrinthe. Avant de couper, je dois ajuster divers paramètres. J'ai importé le fichier conçus à l'ordinateur. Avant de couper, je dois ajuster divers paramètres. Le rouge représente la coupe et le noir représente le gravure. Donc, j'ai défini la bordure en rouge , le chemin du milieu et la zone du logo en noir. Enfin, j'ai obtenu la planche du labyrinthe.

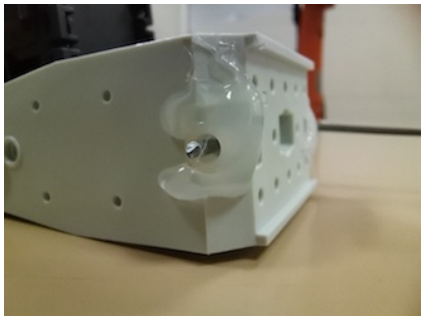


L’assemblage

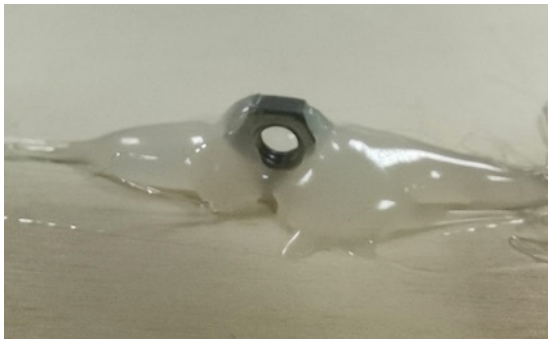
1. Pour réaliser la partie mécanique, j’ai coupé les deux planches (labyrinthe et la planches du bas). J’ai acheté un tube, son diamètre intérieur est 5mm, son diamètre extérieur est 16mm et son longueur est 30cm. J'ai mis l’universel joint dessus le tube. J'ai placé l‘universel joint et la tube au centre du labyrinthe. Percez un trou au centre de la plaque supérieure et fixez le joint universel avec un écrou. Ceci complète la structure de base.



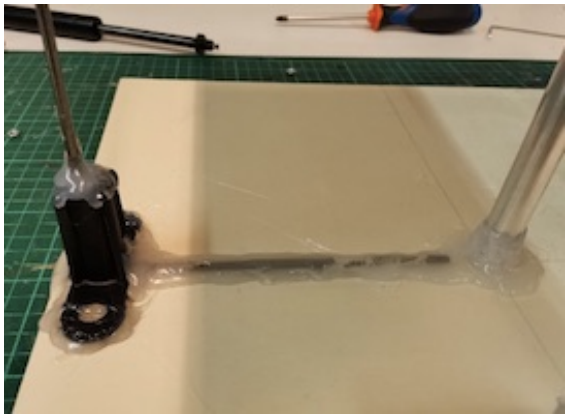
2. J’utilise les composants du servomoteur Bioloid comme la bascule. Et ajouter du fil de fer au milieu. Pour pouvoir contrôler précisément l'inclinaison du labyrinthe, il faut limiter la zone de mouvement du fil, j'ai donc mis un anneau en plastique.



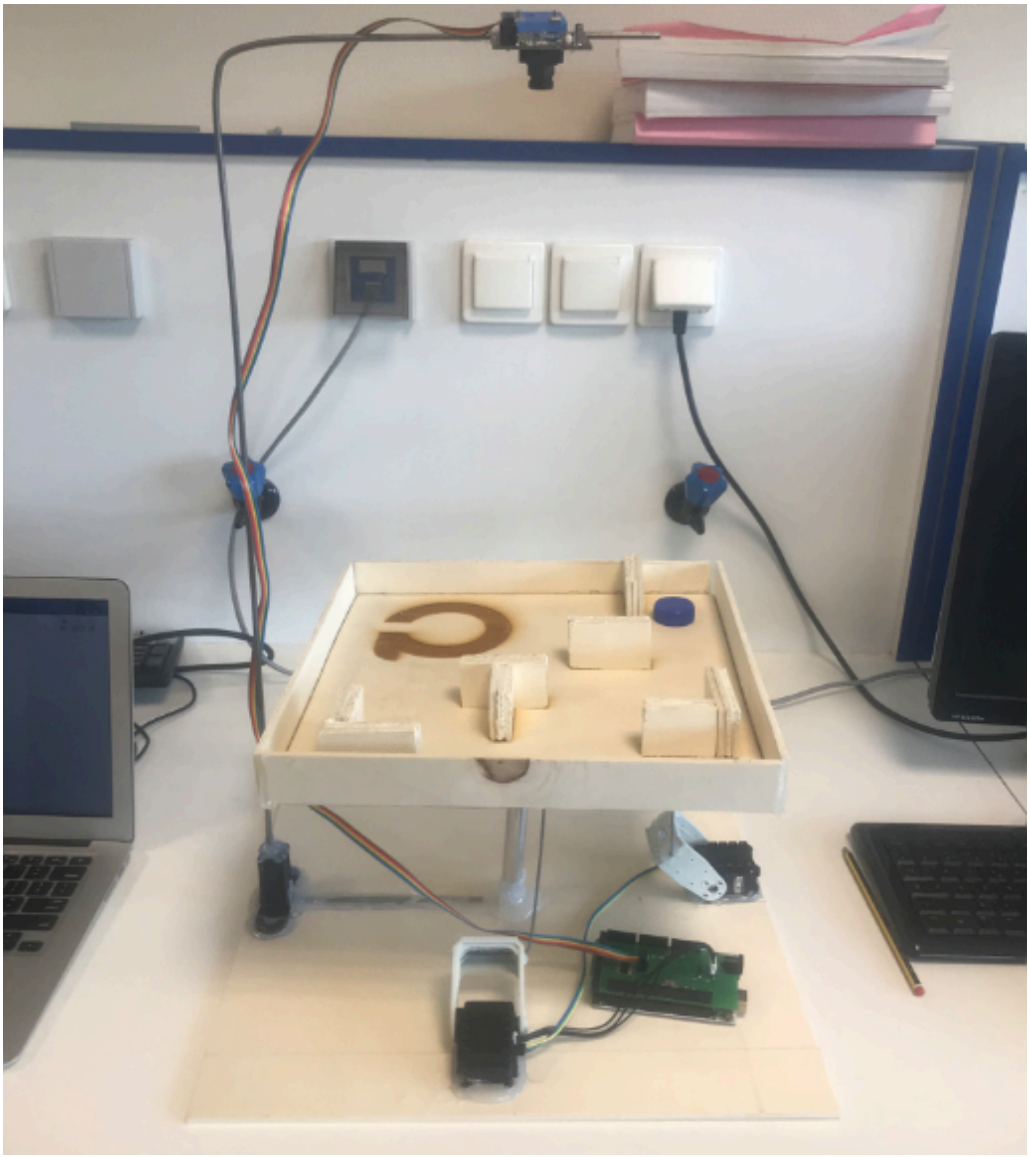
3. Pour fixer le fil de fer sur le labyrinthe, j'ai collé un anneau sur la face dessous.




4. Après construire la structure de base, j'ai fixé les servomoteurs sur la planche du bas. En même temps, j'ai ajouté un fil de fer épais pour mettre la caméra.

5. Enfin, j'ai ajouté les mûrs pour limiter le mouvement de la balle.
   La structure mécanique est terminée, comme la suivante :



Partie de caméra

Avant de tester la caméra, il faut installer le logiciel ‘PixyMon’.

Détection d’objets

Je l’ai connecté sur l’ordinateur pour le tester. Lorsque je l'ai connecté, l'image capturée par la caméra est très floue. J'ai essayé d'ajuster la caméra pour la faire la mise au point. Ensuite, j'ai commencé à apprendre à Pixy à reconnaître un objet. Dans le logiciel ‘Pixymon’, j'ai choisi ‘Signature 1’ pour enregistrer la cible du suivi de la caméra. J'ai utilisé l‘orange à titre d'exemple pour tester la caméra. Après l'objet est sélectionné, la carrée sur l'écran suit la position de l‘orange. Cette carrée est pour détecter les coordonnées de l'objet.

Tester avec l'Arduino

Après tester la caméra, je l'ai connecté avec Arduino. Et dans le logiciel 'Pixymon', il faut ajuster le mode à 'Arduino SPI'. La caméra communique et partage les données avec Arduino sous ce mode.

J'ai téléchargé la bibliothèque Arduino sur le site et puis j'ai ouvert le logiciel Arduino et importé le fichier de bibliothèque Pixy correspondant. Pour tester le programme, il faut le compiler et l'importer.

```
#include <PixyI2C.h>
#include <Pixy.h>
#include <TPixy.h>

#include <SPI.h>
// Author: Scott Robinson
// charmedlabs.com
//
// Continuously prints blob data
// using the Pixy library.
#include <SPI.h>
#include <Pixy.h>

Pixy pixy;

void setup()
{

  Serial.begin(9600);
  Serial.print("Starting...\n");
}

void loop()
{
```

```
  static int i = 0;
  int j;
  uint16_t blocks;
  char buf[16];

  blocks = pixy.getBlocks();

  if (blocks)
  {
    i++;

    if (i%50==0)
    {
      sprintf(buf, "Detected %d:\n", blocks);
      Serial.print(buf);
      for (j=0; j<blocks; j++)
      {
        sprintf(buf, "  block %d: ", j);
        Serial.print(buf);
        pixy.blocks[j].print();
      }
    }
  }
}
```

Après avoir réussi à identifier l'objet et à exécuter le firmware, nous avons connecté la caméra et l'Arduino. Exécutez ensuite un programme de base, le port série imprimera toutes les informations de l'objet. Comme indiqué ci-dessous :

```
Starting...
22:42:53.984 -> Detected 1:
22:42:53.984 ->   block 0: CC block! sig: 12 (10 decimal) x: 151 y: 115 width: 126 height: 107 angle -69
22:42:58.057 -> Detected 1:
22:42:58.057 ->   block 0: CC block! sig: 12 (10 decimal) x: 153 y: 108 width: 141 height: 110 angle -51
22:43:01.516 -> Detected 1:
22:43:01.516 ->   block 0: CC block! sig: 12 (10 decimal) x: 145 y: 107 width: 138 height: 110 angle -41
22:43:04.921 -> Detected 1:
22:43:04.921 ->   block 0: CC block! sig: 12 (10 decimal) x: 143 y: 110 width: 128 height: 108 angle -54
22:43:08.424 -> Detected 1:
22:43:08.424 ->   block 0: CC block! sig: 12 (10 decimal) x: 139 y: 111 width: 135 height: 108 angle -52
22:43:12.403 -> Detected 1:
22:43:12.403 ->   block 0: CC block! sig: 12 (10 decimal) x: 138 y: 111 width: 128 height: 109 angle -68
22:43:16.077 -> Detected 1:
22:43:16.077 ->   block 0: CC block! sig: 12 (10 decimal) x: 137 y: 112 width: 136 height: 107 angle -70
```

Il faut communiquer les informations d'objet identifiées par pixy à Arduino via le port série. Arduino analysera ensuite ces informations pour contrôler les moteurs.

getBlocks () : Cette fonction retournera le nombre d'objets reconnus par Pixy.

Ensuite, on peut obtenir les données de chaque objet identifié via le tableau de *pixy.blocks [ ]* (chaque membre du tableau correspond à un objet identifié). Chaque membre (i) contient les éléments suivants:

pixy.blocks [i] .signature: le numéro d'étiquette de l'objet identifié;
pixy.blocks [i] .x: les coordonnées de la position centrale de l'objet identifié dans la direction x;
pixy.blocks [i] .y: les coordonnées de la position centrale de l'objet identifié dans la direction y;
pixy.blocks [i] .width: la largeur de l'objet identifié (1 à 320);
pixy.blocks [i] .height: hauteur de l'objet identifié (1 200);
pixy.blocks [i] .print (): une fonction membre utilisée pour imprimer les informations de l'objet identifié sur le port série.

Partie électronique

Les fonctions réalisées

Tout d'abord, je dois déterminer ce que le circuit doit réaliser. J'ai décidé d'ajouter des boutons et des commutateurs pour contrôler le fonctionnement ou l'arrêt du système. Dans le même temps, j'ai ajouté trois LED pour observer l'état du système.

1). LEDs

- LED rouge: indique si le programme est mis en pause.

- LED orange: indique si le système est alimenté (couplée à l'interrupteur).

- LED vert: indique si le programme tourne.

2). Bouton

Le bouton à contact temporaire contrôle la mise en route du système. Une fois le bouton appuyé le programme de gestion du labyrinthe se met en marche. Si le bouton est appuyé à nouveau le programme se met en pause.

3). Interrupteur

L'interrupteur contrôle l'alimentation du système. Activer l'interrupteur signifie que le système est déjà en place et que le courant circule.

4). Servomoteurs

Je me réfère au projet IMA4 de 2015 concernant la commande de servos bioloid pilotés par Arduino. J'utilise un buffer permettant de commander les servos.

Tester le servomoteur

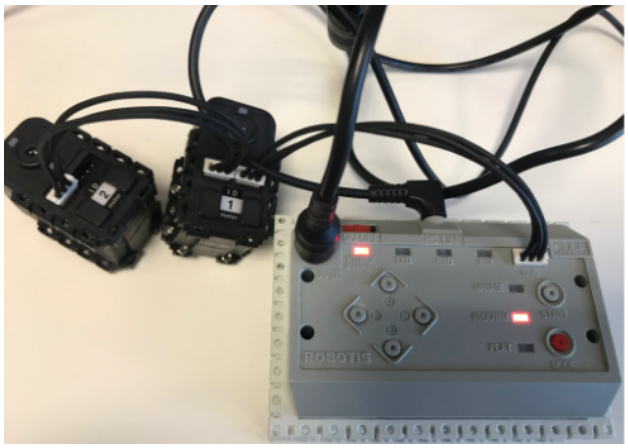
Pour tester les servomoteurs, je m'ai abord référé aux instructions d'utilisation et puis j'ai connecté les deux moteurs avec la source. Et ensuite pour les programmer, il faut installer le logiciel 'Behavior Control Programmer' avec le disque. J'ai essayé de réaliser l'action facile. Je règle séparément la vitesse et l'angle des deux moteurs, puis observe leur comportement. Il y a deux problèmes :

- Le premier moteur tourne à la vitesse spécifiée, mais le deuxième moteur ne tourne pas. J'ai essayé de changer l'ordre des connexions ou de changer le programme, mais il y a rien qui change.

- Le premier moteur tourne éternellement, l'angle cible j'ai défini est invalide.

J'ai consulté le livre de référence et demandé à mon tuteur. J'ai trouvé que la raison pour laquelle le moteur n°2 ne tournait pas était que son numéro de série était devenu le n°6. Ensuite, le programme a changé son numéro en n°2. Testez à nouveau et le moteur n°2 fonctionne correctement. Deuxièmement, il faut précédemment défini la vitesse de rotation et l'angle de rotation du moteur en sélectionnant différentes fonctions.

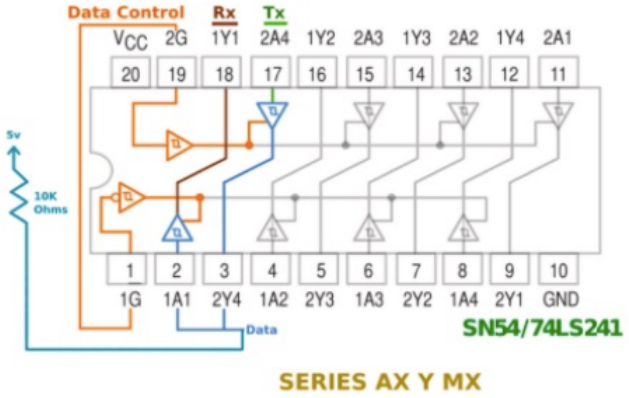
Cependant, j'ai découvert plus tard que le moteur ne tournait pas à la vitesse et à l'angle spécifiés. J'ai ensuite essayé de contrôler la rotation des deux moteurs via les boutons. Les deux boutons de haut et bas sont pour contrôler le moteur n°1, les deux de gauche et droite sont pour contrôler le moteur n°2. Et le programme est comme ci-dessous :



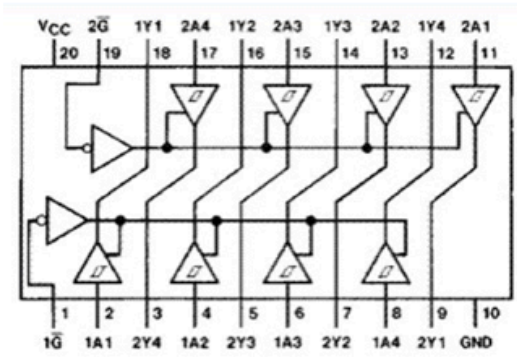
Pour ce projet, je dois remplacer le CM5 par une carte Arduino, ceci afin d'avoir des possibilités de programmation étendue. Après avoir étudié ces caractéristiques, j'ai remarqué que l'on pourrait directement passer par les ports RX et TX de l'Arduino pour le contrôle des servomoteurs, il suffirait juste d'ajouter un composant 74LS241N pour les contrôles alternatifs de TX et RX. En effet ce buffer trois états permettrait de

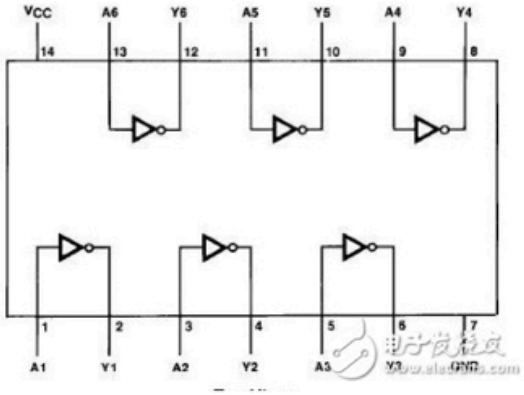
faire passer alternativement Rx ou Tx en état d'haute impédance, ceci dans le but de n'avoir qu'une unique information sur le bus data vers le servo.

En raison du composant manquant 74LS241N, j'ai choisi de remplacer ce composant par un buffer 74LS244N. J'ai réalisé les branchements entre l'Arduino et les servos via ce composant en nous aidant de les datasheets des ces composants. De cette façon, il sera possible de se substituer du CM5 et de commander directement les servos via l'Arduino. J'ai effectué quelques tests en connectant les ports TX et RX de l'Arduino sur les différentes entrées du buffer 3 états. Le test a échoué et la connexion entre les servos et Arduino n'a pas été établie avec succès.

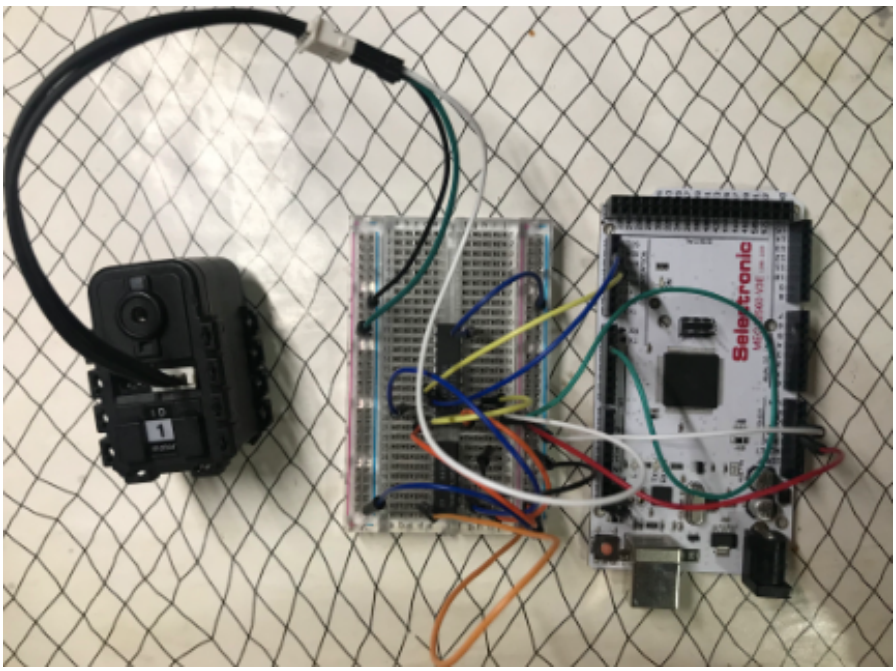


74LS244N & 74HC04N





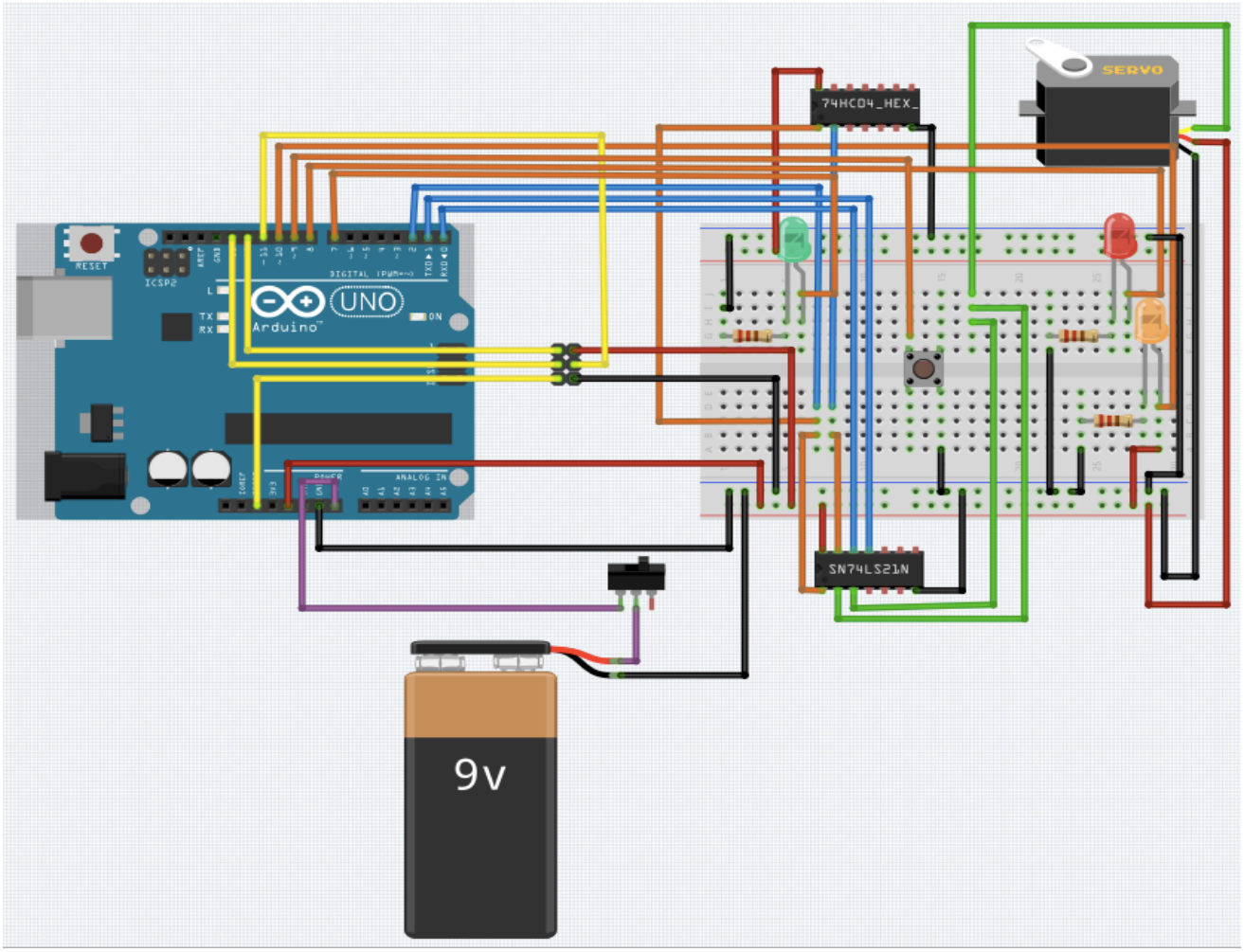
En recherchant les informations, j'ai trouvé que la différence entre 74LS241N et 74LS244N est que le port 2G de ce dernier doit être inversé. En d'autres termes, 74LS241N est un buffer non inverseur et ne peut pas être remplacée directement. Cela signifie que je dois ajouter un inverseur pour obtenir le même effet.
J'ai donc ajouté l'inverseur 74HC04N, ce qui donne pour le circuit, la figure ci-dessous :



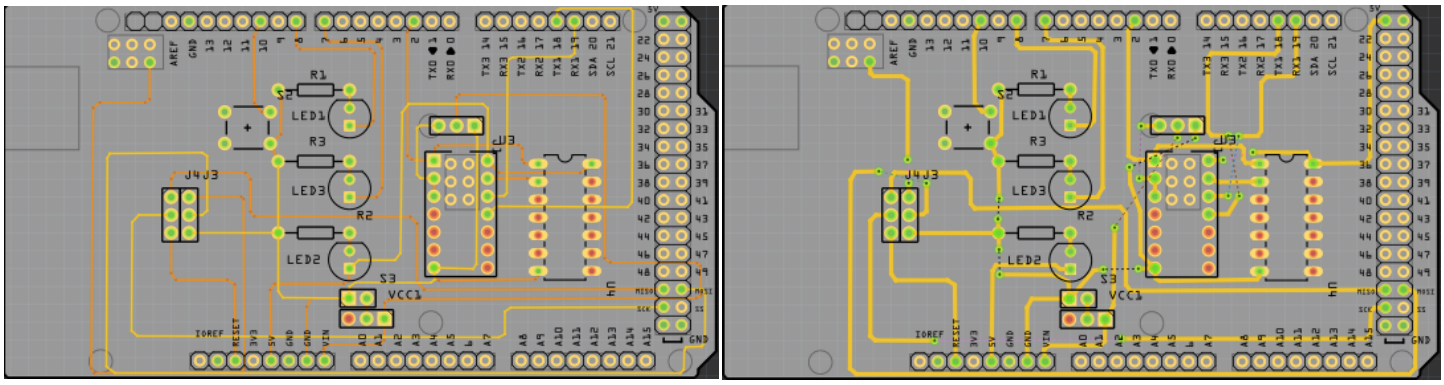
Après avoir ajouté inverseur, j'ai testé le moteur avec un exemple de code. Le moteur a tourné avec succès. J'ai ouvert le moniteur série en même temps, on peut voir les informations imprimées et connaître la rotation du moteur. On peut utiliser des programmes pour ajuster différents moteurs, y compris le sens de rotation, la vitesse de rotation et l'angle de rotation.

Dessiner de la carte électronique

Une fois le test terminé, j'ai dessiné le schéma dans le logiciel et ajouté des LEDs, bouton et interrupteur pour contrôler et afficher l'état du système. J'ai également ajouté un connecteur entre la caméra et l'Arduino. Il faut tester davantage ce circuit.

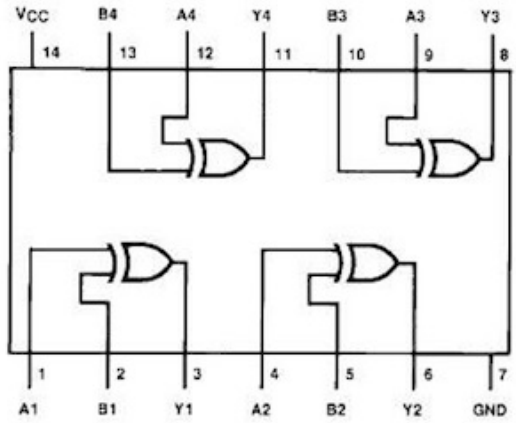


Après avoir confirmé la connexion, j'ai commencé à concevoir le PCB. Au début, la carte PCB que j'ai connectée était à double face. Après avoir consulté l'encadrant, car la carte doit être empilée avec Arduino, on doit concevoir le PCB sur une seule face avec des pistes plus larges. Dans ce cas, afin d'éviter les croix, il faut souder les cavaliers en quelques connexions. Le dessin de conception est le suivant:

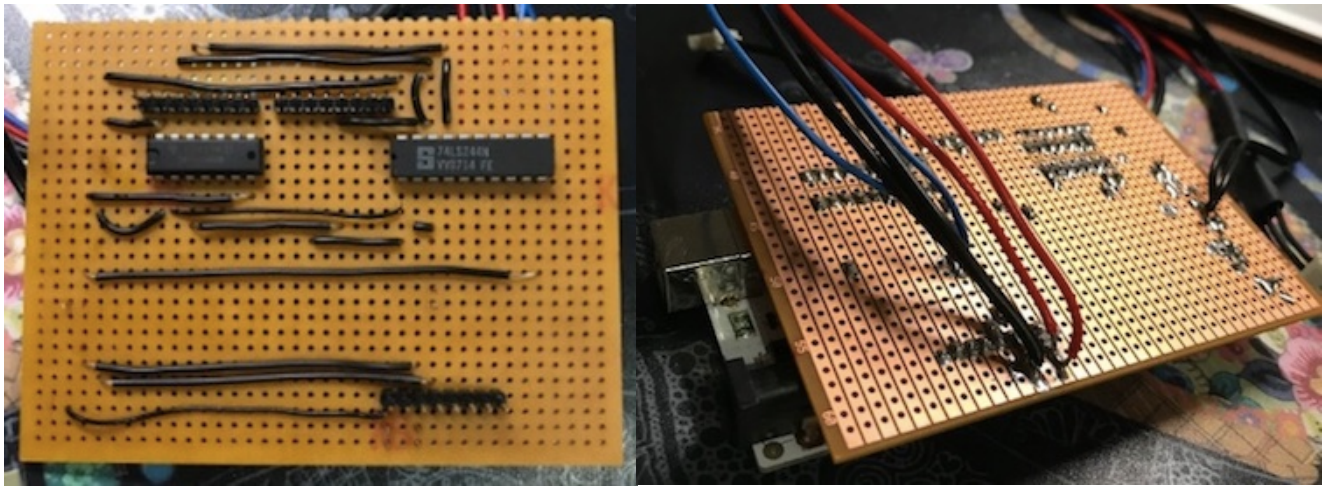


Enfin, pour des raisons techniques, après avoir écouté les conseils du encadrant et du département électronique, on a choisi d'utiliser le plaque d'essai pour réaliser la partie électronique. Après avoir fait la soudure, je l'ai testé mais j'ai fait une erreur. En fait, les composants sont alimentés par l'Arduino, l'Arduino et les servomoteurs sont

alimentés par le batterie 9V. Il faut connecter la pôle positive à Vin sur l'Arduino. Je l'ai connecté à 5V sur l'Arduino. Donc il est court-circuité. L'inverseur 74HC04N est tombé en panne. Je l'ai remplacé par le composant 74LS86AN, c'est un composant logique XOR.

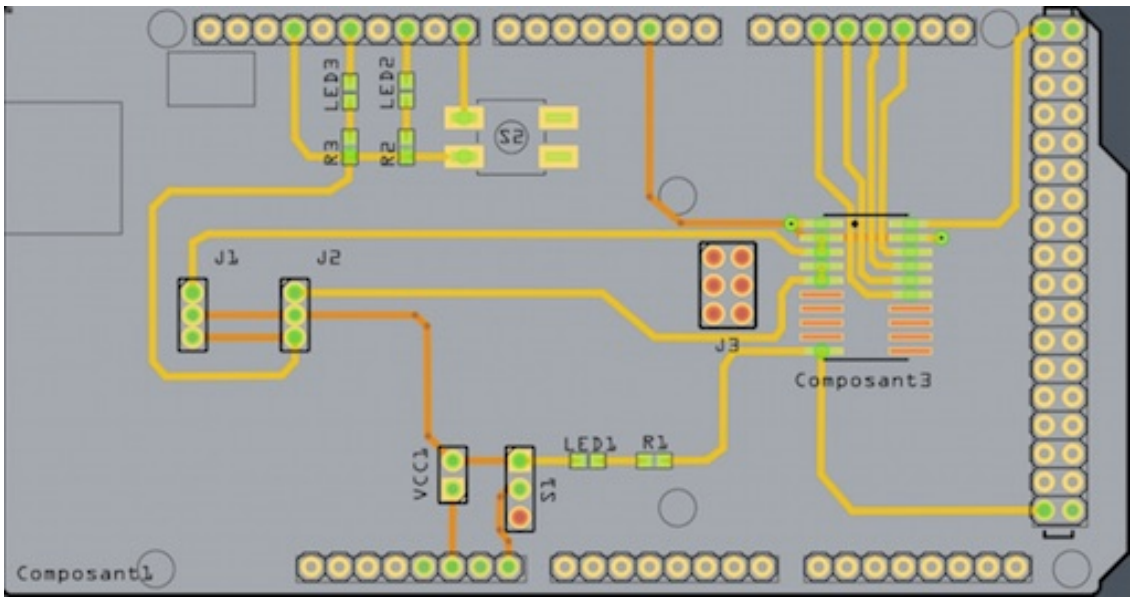


Selon le principe, la deuxième broche est comme l'entrée et la troisième broche est comme la sortie. Il faut connecter la première broche à Vcc. Dans ce cas, selon le logique, l'entrée et la sortie sont inverses. Et puis j'ai ajouté le composant 74LS86AN et réalisé la soudure. La carte soudée est comme ci-dessous :

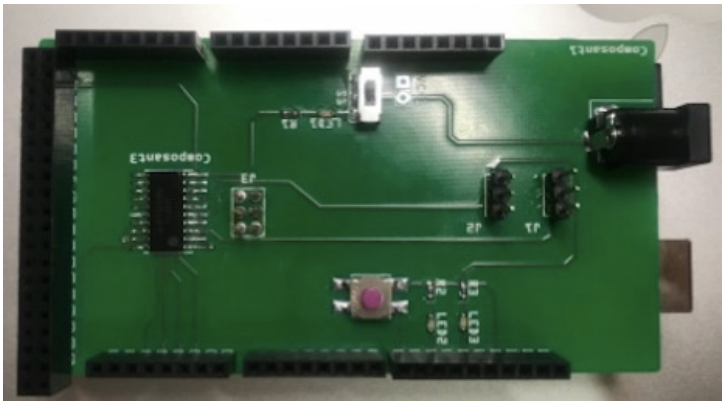


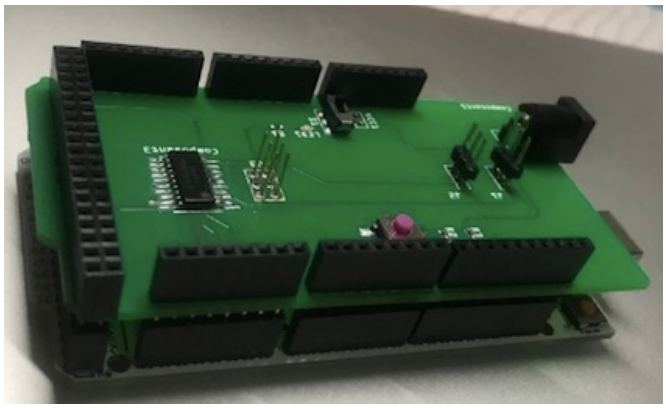
Réaliser un vraie carte

Avant, en raison du composant manquant 74LS241N, j'ai choisi de remplacer ce composant par un buffer non inverseur 74LS244N. M.REDON a acheté un composant 74LS241N beaucoup plus adapté que le LS244 qui m'a été fourni et que du coup mon PCB était plus difficile à router. M.REDON a dessiné un PCB avec le composant 74LS241N comme ci-dessous :



Sur ce PCB, il y a des LEDs pour afficher l'état du système, et il y a un bouton et un interrupteur pour contrôler le système. M.REDON a aussi ajouté les broches qui sont pour connecter avec les servomoteurs et la caméra. La carte est comme suivante :





Partie programmation

Connaître les fonctions bases (la caméra et les servomoteurs)

Tout d'abord, il faut apprendre les bibliothèques de la caméra et des servomoteurs. Les fonctions bases qu'on a besoin d'utiliser sont suivantes :

1. Pixy :

```
blocks = pixy.getBlocks();
```

Cette fonction est pour obtenir le nombre des objets détectés.

```
pixy.blocks[j].print();
```

Cette fonction est pour imprimer les informations d'objet détecté. On n'a que besoin des coordonnées (sur l'axe X et Y) de la bille.

```
pixy.blocks[0].x;
pixy.blocks[0].y;
```

2. Dynamixel :

```
Dynamixel.move(ID,Position);
```

Cette fonction est pour contrôler la position du servomoteur.

```
Dynamixel.moveSpeed(ID,Position,Speed);
```

Cette fonction est pour contrôler la position et la vitesse du servomoteur.

3. Initialisation :

```
Serial.begin(1000000);

pixy.init();

Dynamixel.begin(1000000,2);

Dynamixel.setEndless(2,OFF);
Dynamixel.setEndless(1,OFF);
```

Commander les moteurs par la position d'un objet

Nous devons transmettre les informations de coordonnées collectées par la caméra à Arduino pour contrôler le moteur. Cela permet au système de contrôler directement la rotation des servomoteurs selon la changement de la position de l'objet.

camera_info() : Cette fonction permet d'extraire les coordonnées x et y de l'objet reconnu par la caméra.

sport() : Dans cette fonction, imprimez les informations de coordonnées de l'objet. Le moteur changera le sens de rotation en fonction du changement des coordonnées de l'objet.

J'ai écrit les commandes simples pour le tester. Lorsque la coordonnée x est supérieure à zéro le moteur 1 tourne à gauche. Le code est le suivant :

```
void loop(){

  uint16_t location[2], x, y;
  camera_info(location);
  sport(location);
}

void camera_info(uint16_t* location)
{
  static int i = 0;
  uint16_t blocks;
  // grab blocks!
  blocks = pixy.getBlocks();

  // If there are detect blocks, print them!
  if (blocks)
  {  Serial.print("Detected number of object : ");
     Serial.println(blocks);
     location[0] = pixy.blocks[0].x;
     location[1] = pixy.blocks[0].y;
  }
}
```

```
void sport(uint16_t* loc){
  uint16_t x, y;
  x = loc[0];
  y = loc[1];
  Serial.print("x = ");
  Serial.println(x);
  Serial.print("y = ");
  Serial.println(y);
  if(x>0){
    /*i = Dynamixel.turn(1,LEFT,random(200,800));*/
    Serial.println("left1");
    Dynamixel.turn(1,LEFT,500);
    /*Serial.println("i = ");
    Serial.println(i);*/
  }
```

Les résultats et les informations d'impression obtenus après l'exécution du programme sont les suivants :

```
00:28:53.863 -> x = 214
00:28:53.863 -> y = 139
00:28:53.863 -> left1
00:28:53.863 -> x = 214
00:28:53.863 -> y = 139
00:28:53.863 -> left1
00:28:53.898 -> x = 214
00:28:53.898 -> y = 139
00:28:53.898 -> left1
00:28:53.898 -> x = 214
00:28:53.898 -> y = 139
00:28:53.898 -> left1
00:28:53.898 -> x = 214
00:28:53.898 -> y = 139
00:28:53.898 -> left1
00:28:53.936 -> x = 214
```

Calculs et analyse de la position des servomoteurs

1. Calcul de la position initiale

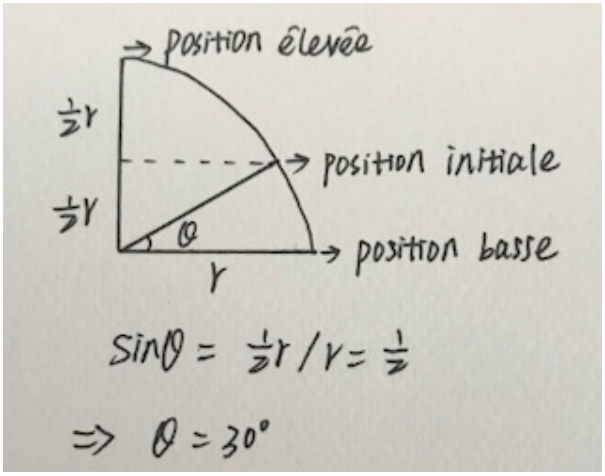
Avant fixer et commencer à commander les servomoteurs, il faut déterminer les trois positions du moteur: la valeur de position initiale, la valeur de position la plus élevée et la valeur de position la plus basse. Je les testé en utilisant la commande :

```
Dynamixel.move(ID, Position);
```

- Pour la moteur 1 : 450 (élevée), 757 (basse)

- Pour la moteur 2 : 512 (élevée), 819 (basse)

Enfin, il faut déterminer la valeur de position initiale. Il faut assurer que le moteur tourne pour incliner la plaque, et que la hauteur de sa montée et de sa descente doit être la même, afin que l'inclinaison soit la même dans les deux sens. L'angle entre les positions haute et basse que je fixe est de 90 degrés. Donc, dans la plus grande mesure, l'ascension est la longueur du rayon r de la bascule. Ensuite, la position initiale du moteur doit être à 1 / 2r du sol. Calculé comme suit :



Donc, sur la rapport, les positions initiales des deux moteurs peuvent être déterminées :

- Pour la moteur 1 : 757 - (757-450)/3 = 654.7

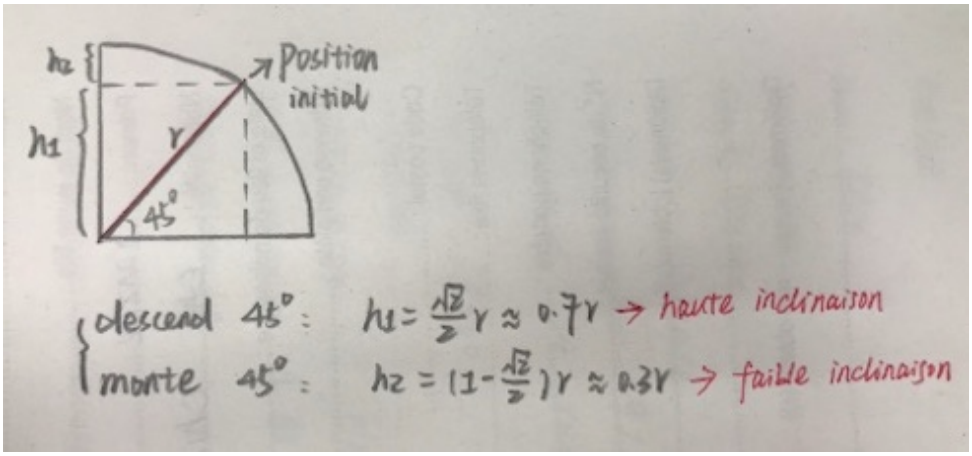
- Pour la moteur 2 : 819 - (819-512)/3 = 716.7

Avant fonctionner le système, il faut initialiser chaque fois les positions des deux servomoteurs.

```
Dynamixel.move(1, 654.7);
Dynamixel.move(2, 716.7);
```

2. Analyser l'inclinaison du labyrinthe

Selon les tests, lorsque le moteur monte et descend à la même angle, l'inclinaison de la planche dans les deux sens opposés est différente. Grâce à l'analyse géométrique, les résultats sont les suivants :



J'ai donc besoin d'ajuster le coefficient d'inclinaison de la plaque des deux côtés. Le coefficient du côté où la pente est faible est grand et le coefficient du côté où la pente est élevée est petit. Cela garantit que l'inclinaison est presque la même lorsque la plaque est inclinée des deux côtés.

```
if(pi.voltageX>=0){
    pi.angleX=640.7-2*pi.voltageX;
}
else if(pi.voltageX<0){
    pi.angleX=640.7-1*pi.voltageX;
}
```

Programme sur le contrôleur P (proportion)

P : Effet d'ajustement proportionnel : l'écart du système est reflété de manière proportionnelle. Une fois qu'un écart se produit dans le système, l'ajustement proportionnel produit immédiatement un effet d'ajustement pour réduire l'écart. Le grand effet proportionnel peut accélérer le réglage et réduire l'erreur, mais une proportion trop grande réduira la stabilité du système et même rendra le système instable.

Le principe de P est comme ci-dessous :

$$U(t) = Kp * Err(t)$$

Plus la bille est éloigné de la position cible, plus le moteur tourne; plus la bille est proche de la position cible, plus le moteur est proche de la position initiale. Sur le principe, il faut définir plusieurs variables.

```
void Pid_controlClass :: PID_init(){
    printf("PI_init begin \n");
    pi.SetPositionX=0.0;
    pi.ActualPositionX=0.0;
    pi.errX=0.0;
    pi.voltageX=0.0;
    pi.SetPositionY=0.0;
    pi.ActualPositionY=0.0;
    pi.errY=0.0;
    pi.voltageY=0.0;
    pi.Kp=0.2;
    pi.angleX=0.0;
    pi.angleY=0.0;
    printf("PI_init end \n");
}
```

L'erreur est la distance entre la position actuelle et la position cible. Définissez une valeur intégrale pour additionner chaque erreur. Utilisez ensuite le principe de contrôle P pour stocker les résultats du calcul dans pi.voltage. Les servomoteurs étant fixés sur la planche du bas, la plage de rotation est limitée. Et j'ai déjà déterminé la plage haute et basse de rotation du moteur. J'ai donc fixé les valeurs maximales et minimales pour limiter la plage de rotation du moteur. J'ai écrit deux fonctions pour contrôler les axes X et Y respectivement. Prenons l'exemple de l'axe des X :

```
float Pid_controlClass :: PID_realizeX(float PositionX){
   pi.SetPositionX=PositionX;
   pi.errX=pi.SetPositionX-pi.ActualPositionX;
```

```
    pi.integralX+=pi.errX;
    pid.voltage=pid.Kp*pid.err;
    pi.angle1=-3*pi.voltageX+624.7;
    if (pi.angle1>757) pi.angle1=757;
    if (pi.angle1<450) pi.angle1=450;
    return pi.angle1;
}
```

Ensuite, dans le programme principal. Il faut ajouter des instructions à la fonction d'initialisation afin que le moteur puisse revenir à sa position initiale avant que le système ne fonctionne à chaque fois.

```
Dynamixel.move(1,654.7);
Dynamixel.move(2,716.7);
```

Avant le test, j'ai défini la position cible sur le point central de la planche du haut (170, 80). La bille doit se rapprocher du point central à n'importe quelle position. La position actuelle de la bille est mise à jour en temps réel en fonction des coordonnées collectées par la caméra. La position où le moteur tourne est la valeur de retour de la fonction de contrôleur P.

```
 Pid_control.pi.ActualPositionX = x;
 Pid_control.pi.ActualPositionY = y;
 Serial.println(Pid_control.PID_realizeX(Px));
 Serial.println(Pid_control.PID_realizeY(Py));
 //Serial.println(Pid_control.pi.voltageX);
 Dynamixel.moveSpeed(1,Pid_control.PID_realizeX(Px),200);
 Dynamixel.moveSpeed(2,Pid_control.PID_realizeY(Py),200);
```

Le résultat du test est que la bille se déplace dans un mouvement circulaire autour du point central. Mais parce que la structure n'est pas stable, les résultats des tests ne sont pas précis et stables.

Réaliser la trajectoire carée

À base du programme avant, l'objectif prochain est de réaliser un trajectoire en carré. D'abords, j'ai mis quatre set-points au coin de la planche.

```
typedef struct cible_position{
      uint16_t x;
      uint16_t y;
}point_parcours;
point_parcours *parcours;
```

```
parcours = (point_parcours*)malloc(4 * sizeof(struct cible_position));
      parcours[0].x=108;
      parcours[0].y=138;
      parcours[1].x=108;
      parcours[1].y=33;
      parcours[2].x=233;
      parcours[2].y=33;
      parcours[3].x=233;
      parcours[3].y=138;
```

Dans le même temps, car la position de la balle est sensible au changement, et il faut un certain temps à la caméra pour récupérer la balle. Par conséquent, si la position cible que nous définissons est des coordonnées de point, il n'y a aucune garantie que la caméra puisse collecter des données lorsque la balle atteint la position cible. Dans ce cas, j'ai décidé de définir la position cible à un intervalle. Lorsque la balle atteint l'intervalle cible désigné, l'intervalle cible est automatiquement changé pour le suivant.

```
void Pid_controlClass :: position_cible(){
   if((pi.X_reel<=parcours[pi.i].x+15)&(pi.X_reel>=parcours[pi.i].x-
15)&(pi.Y_reel<=parcours[pi.i].y+15)&(pi.Y_reel>=parcours[pi.i].y-15)){
        pi.i++;
        if(pi.i==4){
           pi.i=0;
        }
   }
}
```

Après les tests, j'ai constaté que le ballon accélère toujours en se déplaçant vers la position cible. En effet, sous le contrôle du contrôleur P, lorsque la petite balle se déplace du côté de la position cible, la direction d'inclinaison de la tablette est toujours la même. Cela fait que le ballon continue d'accélérer, et lorsqu'il atteint la position cible, la vitesse est trop élevée pour s'arrêter. Par conséquent, j'ai changé la position cible dans la fonction de commande P au milieu de la longueur du côté carré. Cela maintient le ballon proche de 0 lorsqu'il atteint les 4 zones cibles définies précédemment, ce qui est plus stable.

```
if (pi.i==3){
   pi.X_cible=(parcours[0].x+parcours[3].x)/2;
}else{
   pi.X_cible=(parcours[pi.i].x+parcours[pi.i+1].x)/2;
 }
```

Pour que le ballon se déplace en ligne droite, il faut assurer qu'un seul moteur est actionné par mouvement. L'autre moteur reste en position initiale. Par conséquent, en fonction de la relation définie entre les moteurs et l'axe x et l'axe y, le moment où les deux moteurs reviennent à la position initiale peut être déterminé. Pour le moteur qui fonctionne, l'angle de rotation est contrôlé selon le principe du contrôle proportionnel.

```
if(pi.i==1|pi.i==3){
    pi.angleX=640.7;
}else{
        if(pi.voltageX>=0){
            pi.angleX=640.7-2*pi.voltageX;
        }
        else if(pi.voltageX<0){
            pi.angleX=640.7-1*pi.voltageX;
        }
 }
if(pi.i==0|pi.i==2){
    pi.angleY=716.7;
}else{
        if(pi.voltageY>=0){
           pi.angleY=716.7-3*pi.voltageY;
        }
        else if(pi.voltageY<0){
           pi.angleY=716.7-2*pi.voltageY;
        }
}
```

Répéter la trajectoire spécifiée

Après avoir mis en œuvre le mouvement carré de la balle, j'ai changé la position cible. Puisque le carré est un chemin régulier, mais notre objectif est que la balle effectue un mouvement répétant selon la trajectoire prescrit. J'ai donc défini une trajectoire irrégulier et défini plusieurs positions cibles.
De plus, il y a une ligne droite entre les deux positions cibles pour garantir que la balle se déplace dans une seule direction. De cette façon, le principe que j'ai utilisé pour implémenter le mouvement carré s'applique également ici. On a juste besoin de changer les positions cibles.

```
parcours = (point_parcours*)malloc(8 * sizeof(struct cible_position));
      parcours[0].x=250;
      parcours[0].y=167;
```

```
parcours[1].x=140;
parcours[1].y=165;
parcours[2].x=140;
parcours[2].y=120;
parcours[3].x=93;
parcours[3].y=120;
parcours[4].x=115;
parcours[4].y=36;
parcours[5].x=170;
parcours[5].y=50;
parcours[6].x=170;
parcours[6].y=10;
parcours[7].x=237;
parcours[7].y=27;
```

Conclusion

J'ai trouvé le projet très intéressant car il m’a permis de mettre en pratique les connaissances acquises au cours de notre formations IMA, notamment la conception et la réalisation des cartes électroniques, la programmation des microcontrôleurs mais aussi comment lire et manipuler les données issues d'une caméra (pixy). Dans le même temps, j’ai également une nouvelle compréhension du contrôle. Bien que seul le contrôle proportionnel soit utilisé, une nouvelle compréhension du contrôle PID est obtenue en cherchant les informations.

J'ai également expérimenté le travail et la gestion d'un projet sur le long terme. Un bon plan, la répartition des tâches et la capacité d'auto-apprentissage sont les valeurs que je retiens de cette expérience.

Annexes

Bibliothèque Arduino – Servomoteur Dynamixel :

https://www.savageelectronics.com/blog/arduino-biblioteca-dynamixel

Bibliothèque Arduino – La camèra Pixy :

https://pan.baidu.com/s/1dDpDlvV - list/path=%2F

Bibliothèque Arduino – Contrôleur P :

```
#if defined(ARDUINO) && ARDUINO >= 100  // Arduino IDE Version
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#include "Pid.h"

void Pid_controlClass :: PID_init(){
    pi.X_cible=0.0;
    pi.Y_cible=0.0;
    pi.X_reel=0.0;
    pi.Y_reel=0.0;
    pi.deltaX=0.0;
    pi.deltaY=0.0;
    pi.voltageX=0.0;
    pi.voltageY=0.0;
    pi.Kp=0.2;
    //pi.Ki=0.02;
    pi.angleX=0.0;
    pi.angleY=0.0;
    pi.i=0;
```

```
  parcours = (point_parcours*)malloc(8 * sizeof(struct cible_position));
  parcours[0].x=250;
  parcours[0].y=167;
  parcours[1].x=140;
  parcours[1].y=165;
  parcours[2].x=140;
  parcours[2].y=120;
  parcours[3].x=93;
  parcours[3].y=120;
  parcours[4].x=115;
  parcours[4].y=36;
  parcours[5].x=170;
  parcours[5].y=50;
  parcours[6].x=170;
  parcours[6].y=10;
  parcours[7].x=237;
  parcours[7].y=27;
```

```
float Pid_controlClass :: corriger_X(){
    //for(int i=0;i<20;i++){
        //while(pi.X_reel!=parcours_cible[i].x){
            //pi.X_cible=parcours[pi.i].x;
            if (pi.i==3){
                pi.X_cible=(parcours[0].x+parcours[3].x)/2;
            }else{
                pi.X_cible=(parcours[pi.i].x+parcours[pi.i+1].x)/2;
            }
            pi.deltaX=pi.X_cible-pi.X_reel;

            pi.voltageX=pi.Kp*pi.deltaX;
            if(pi.i==0|pi.i==2){
                pi.angleX=654.7;
            }else
            {
                if(pi.voltageX>=0){
                    pi.angleX=654.7-3*pi.voltageX;
                }
                else if(pi.voltageX<0){
                    pi.angleX=654.7-2*pi.voltageX;
                }
            }
            if (pi.angleX>757) pi.angleX=757;
            if (pi.angleX<450) pi.angleX=450;
        //}
    //}
    return pi.angleX;
}
```

```
float Pid_controlClass :: corriger_Y(){

        if (pi.i==3){
            pi.Y_cible=(parcours[0].y+parcours[3].y)/2;
        }else{
            pi.Y_cible=(parcours[pi.i].y+parcours[pi.i+1].y)/2;
        }
        pi.deltaY=pi.Y_cible-pi.Y_reel;

        pi.voltageY=pi.Kp*pi.deltaY;
        if(pi.i==1|pi.i==3){
                pi.angleY=716.7;
            }else{
                if(pi.voltageY>=0){
                    pi.angleY=716.7-3*pi.voltageY;
                }
                else if(pi.voltageY<0){
                    pi.angleY=716.7-2*pi.voltageY;
                }
            }

        if (pi.angleY>819) pi.angleY=819;
        if (pi.angleY<512) pi.angleY=512;
    return pi.angleY;
}

Pid_controlClass Pid_control;
```

Codes:

```
#include <pid.h>
#include <DynamixelSerial.h>
#include <SPI.h>
#include <Pixy.h>

/*String comdata="";*/
float Px;
float Py;

int orange=12;

/*This is the main Pixy object */
Pixy pixy;

void setup(){
  pinMode(orange,OUTPUT);

  Dynamixel.setSerial(&Serial1); // &Serial - Arduino UNO/NANO/MICRO, &Serial1,
  Dynamixel.begin(1000000,2);  // Initialize the servo at 1 Mbps and Pin Contro
  Serial.begin(1000000);

  Pid_control.PID_init();
  Dynamixel.moveSpeed(2,716.7,60);     //恢复电机初始位置
  Dynamixel.moveSpeed(1,654.7,60);
```

```
  Dynamixel.setEndless(2,OFF);
  Dynamixel.setEndless(1,OFF);
  delay(1000);


  pixy.init();

  Serial.println("start...");
}

void loop(){
  //digitalWrite(orange,HIGH);

  uint16_t location[2], x, y;
  camera_info(location);

  sport(location);
     /*if(Pid_control.pi.i==17){
        Pid_control.pi.i=0;
      }*/
  delay(20);
}

void camera_info(uint16_t* location)
{
  static int i = 0;
  uint16_t blocks;
  // grab blocks!
  blocks = pixy.getBlocks();

  // If there are detect blocks, print them!
  if (blocks)
  {  Serial.print("Detected number of object : ");
     Serial.println(blocks);
     location[0] = pixy.blocks[0].x;
     location[1] = pixy.blocks[0].y;
  }
}
```

```
void sport(uint16_t* loc){
  uint16_t x, y;
  x = loc[0];
  y = loc[1];
  Serial.print("x = ");
  Serial.println(x);
  Serial.print("y = ");
  Serial.println(y);
  /*if(x>0){
    i = Dynamixel.turn(1,LEFT,random(200,800));
    Serial.println("left1");
    Dynamixel.turn(1,LEFT,500);
    Serial.println("i = ");
    Serial.println(i);
  }*/

  Pid_control.pi.X_reel = x;
  Pid_control.pi.Y_reel = y;


  Pid_control.position_cible();

  Serial.println(Pid_control.corriger_X());
  Serial.println(Pid_control.corriger_Y());
  //Serial.println(Pid_control.pi.voltageX);
  Dynamixel.moveSpeed(1,Pid_control.corriger_X(),400);
  Dynamixel.moveSpeed(2,Pid_control.corriger_Y(),400);
  //delay(1000);
}
```