Rapport de projet IMA4

P3 : Robot régulé

Intitulé du projet : Robot régulé

Réalisé par : Hugo DELBROUCQ

Encadrants : X.Redon, A.Boé, T.Vantroys

Mission : Réalisation d'un robot mobile effectuant les déplacements suivants :

- Avancer
- Reculer
- Tourner à gauche de 10°
- Tourner à droite de 10°

Ces déplacements sont régulés à l'aide de capteurs au choix. La précision attendue est la suivante :

- Avancer et reculer doivent se faire de manière rectiligne.
- Tourner à gauche ou à droite de 10° doit avoir une précision de l'ordre du degré.

Bonus : Mettre en place une régulation du tangage du robot afin de réaliser un auto équilibrage de celui-ci.

Technologies employées :

Réalisation du circuit imprimé : Altium Designer
Communication : Interruptions par commande infrarouge
Réalisation du châssis : Onshape puis exportation vers Inkscape
Programmation du microprocesseur : IDE Arduino
Capteur : Gyroscope 6 axes MPU6050
Microprocesseur : ATMEGA328P-AU

Sommaire

Introduction

Ce projet ayant commencé au début du mois de septembre 2018 et se terminant le 20 septembre 2018 s'inscrit dans le cadre de la formation Informatique, Microélectronique et Automatique de Polytech Lille. Nous aborderons ici les résultats ainsi que les conclusions obtenues lors de ce projet traitant des 3 principaux domaines de notre filière. Tout d'abord, nous allons traiter les différentes options disponibles puis les expérimentations réalisées et enfin le résultat final de ce projet.

Je tiens aussi à remercier les différentes personnes qui ont contribué de près ou de loin à ce projet. Cela a pu me permettre de réaliser ce projet dans de bonnes conditions.

I. Choix des différentes technologies

A. Choix des capteurs

1. Gyroscope MPU6050

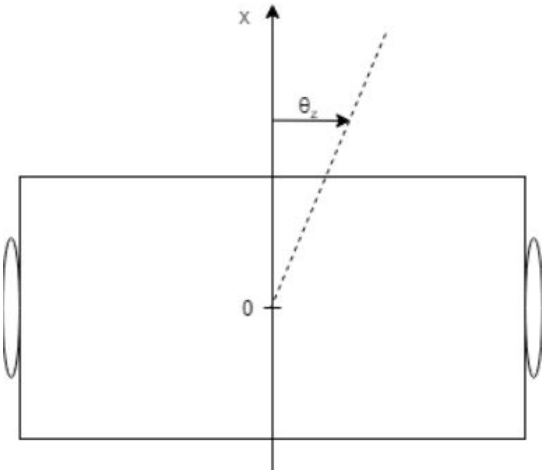
Le gyroscope MPU6050 est très utilisé pour assurer un contrôle en temps réel d'un robot. On retrouve de très nombreux exemples de son application et donc de librairies permettant de l'utiliser ainsi que les nombreux problèmes qu'il apporte avec lui.

Ce capteur utilise le bus I2C pour communiquer avec l'IDE Arduino ce qui rend son application pratique. Ce capteur est de plus bon marché (~7€) et très petit ce qui le rend intéressant d'un point de vue embarqué. On peut d'ailleurs retrouver ses données constructeur ci dessous :

- Sa tension d'alimentation est comprise entre 2.375V et 3.45V
- Il utilise le bus I2C
- Il est constitué d'un accéléromètre 3 axes ET d'un gyroscope 3 axes
- Il utilise les interruptions
- Le gyroscope a une précision allant jusqu'à ±2000°/sec
- Le gyroscope a une consommation nominale de 3.6mA

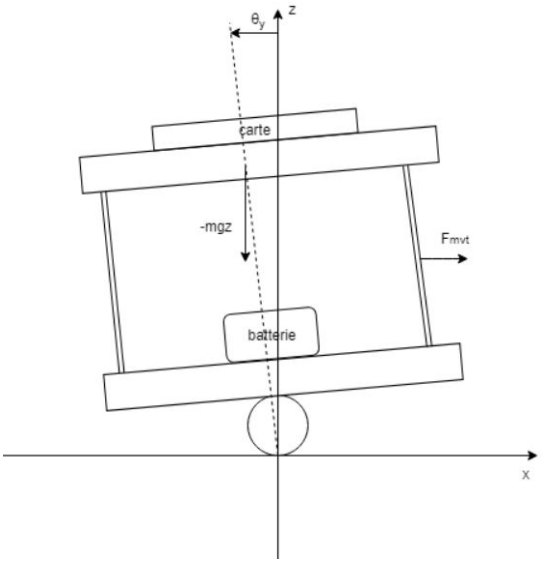
- L'accéléromètre a une précision allant jusqu'à ±16g

Dans notre cas, ce qui nous intéresse le plus est l'angle θ_z formé autour de l'axe Z qui est responsable de l'orientation du robot dans l'espace. En prenant en compte l'angle formé à un certain moment dans le temps, il est alors possible de conserver une direction et une régulation de manière simple.



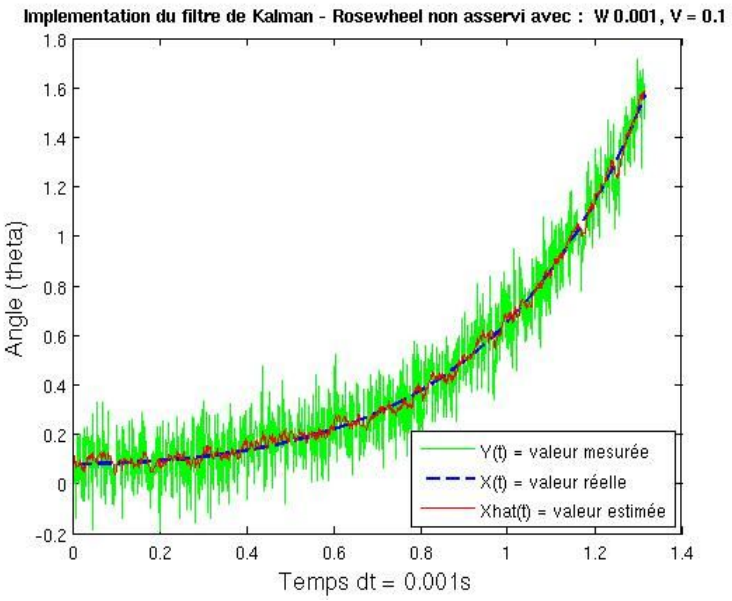
Angle concerné par la régulation de la position du mobile dans l'espace.

Le second angle qui va nous intéresser est l'angle selon l'axe Y responsable du tangage du mobile. Cet angle nous permettra de contrôler l'équilibre du robot afin de réaliser la problématique de pendule inversé proposée en bonus dans la liste des objectifs du projet. Cette partie est bien plus difficile à réguler car la qualité du signal reçu du capteur doit être bien supérieure à celle que l'on utilise pour la position dont la dynamique est inférieure à celle de l'auto équilibrage.



Angle concerné par la régulation en équilibre du mobile dans l'espace.

Néanmoins, ce capteur a beau sembler parfait, il peut arriver qu'il soit bruité. Afin de réduire ce problème, il faut l'éloigner de toute source émettrice d'onde comme le quartz par exemple. L'autre méthode applicable est un filtrage par filtre de Kalman. Ce filtre consiste à prédire les données futures et ensuite réduire l'écart entre valeur réel



Source : Télécom ParisTech

On voit bien ci-dessus que notre valeur estimée est bien plus proche de la valeur réelle que la valeur mesurée. En théorie, ce capteur est largement suffisant pour nos besoins mais un autre capteur a attiré mon attention en début de stage et qui selon moi mérite une certaine attention.

2. Le capteur de souris

Le capteur de souris a pour but premier de permettre la traduction d'un déplacement de la souris en un déplacement du pointeur à l'écran. Pour se faire, il faut que ce capteur retourne les données de déplacement de la souris de manière très précise, ce qui nous amène à penser l'utilisation de ces souris pour observer un déplacement sous notre robot. Il faut pour cela utiliser au minimum 2 capteurs afin de pouvoir constater une éventuelle déviation de consigne.

La souris se décompose des éléments observables à droite. Les deux éléments qui nous intéressent sont la puce ainsi que son capteur intégré et la lentille qui sert à améliorer le focus du capteur.

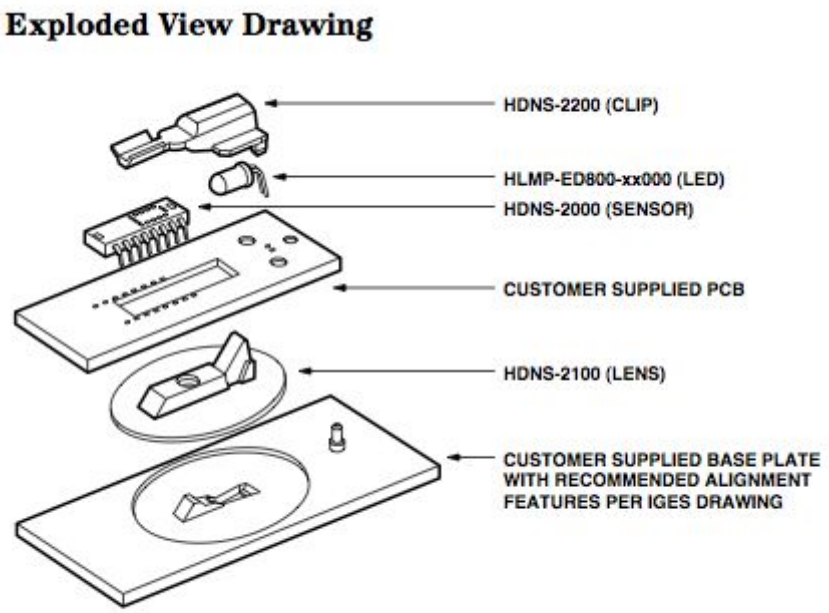
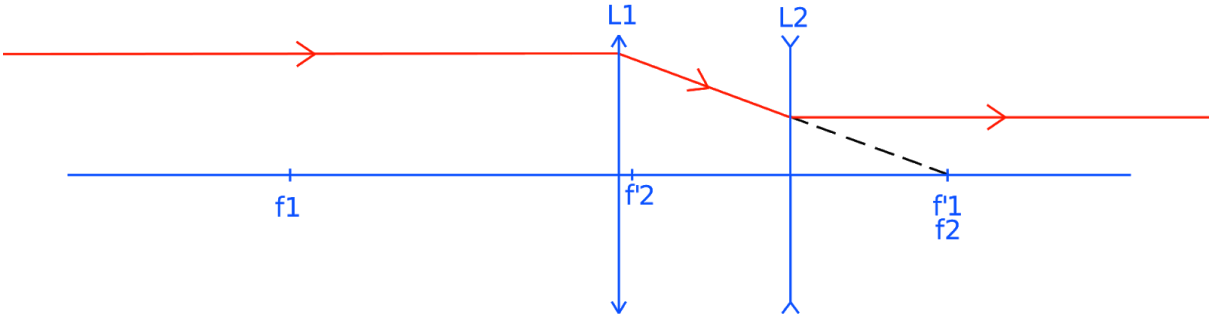


Figure 4.

Les problèmes majeurs que pose leur utilisation sont la dépendance au type de surface sur laquelle le robot se déplace et la hauteur à laquelle on peut placer le

capteur du sol qui est très faible. Ces problèmes peuvent être résolus de différentes manières :

★ L'un des gros sujets à l'étude actuellement concerne la création d'une lentille afocale. Ce type de lentille a pour avantage de prendre un objet supposé à l'infini pour ramener son image dans le plan focal image de la lentille ce qui permet d'avoir un rendu clair et net de notre objet bien qu'il soit très éloigné.



Système afocal avec f' les foyers image des lentilles, f les foyers objets, L1 une lentille convergente et L2 une lentille divergente.

★ De plus, l'utilisation de ce genre de capteur est très facile à appliquer. Leur consommation est très faible et on peut les placer en série très facilement. Avec un nombre d'entre eux suffisant, nous serions capables de créer une représentation de l'espace qui nous entoure à partir des déplacements observés par les souris. Mais cela dépend encore malheureusement des capteurs qui ne sont pas assez précis pour être utilisés sans limites.

L'idéal dans notre cas est de prendre deux capteurs infrarouges avec des connecteurs ps2 plus faciles à traduire que les bus usb. Leur utilisation n'est intéressante que si le gyroscope renvoie des données inutilisables. Dans le cas où les données seraient exploitables, il vaut mieux limiter de surcharger le microprocesseur.

B. Choix de la morphologie du robot

Notre but final étant de réguler le balancement du robot, il était nécessaire de se pencher sur la morphologie de celui-ci et de centrer, abaisser son centre de

gravité au maximum. Le mieux est donc d'opter pour une forme large afin d'avoir l'impact de l'accélération/décélération plus important ce qui nous offre une plage de puissance à appliquer sur le tangage plus importante et un meilleur contrôle.

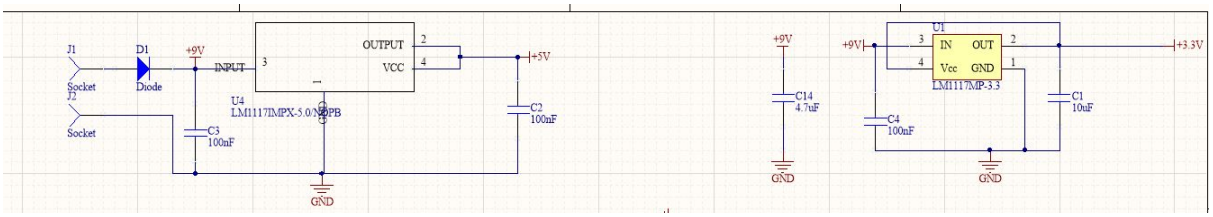
Le centrage du centre gravité permet notamment de limiter les déviations de trajectoire au cours de la commande. Il faut prendre en compte cela dans notre étude. De même pour les roues, il faut privilégier des roues épaisses avec stries pour limiter le glissement. Cette étude représente une part importante de la régulation d'un système. En effet, plus on arrive à rendre un système stable et moins la régulation à appliquer dessus est importante ce qui permet une réduction des coûts car la régulation à apporter est moins lourde.

II. Expérimentations et résultats :

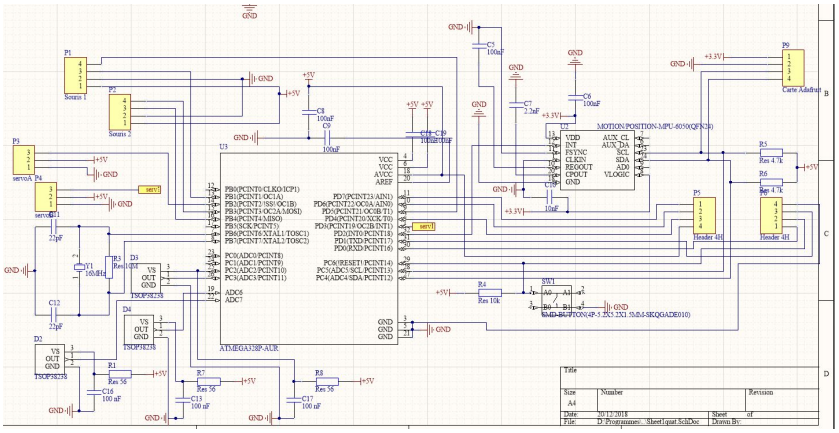
A. Le circuit imprimé

La réalisation du circuit imprimé est la partie qui m'a pris le plus de temps. Le schéma du montage est décomposé en 3 grandes parties :

1. Régulateurs de tension

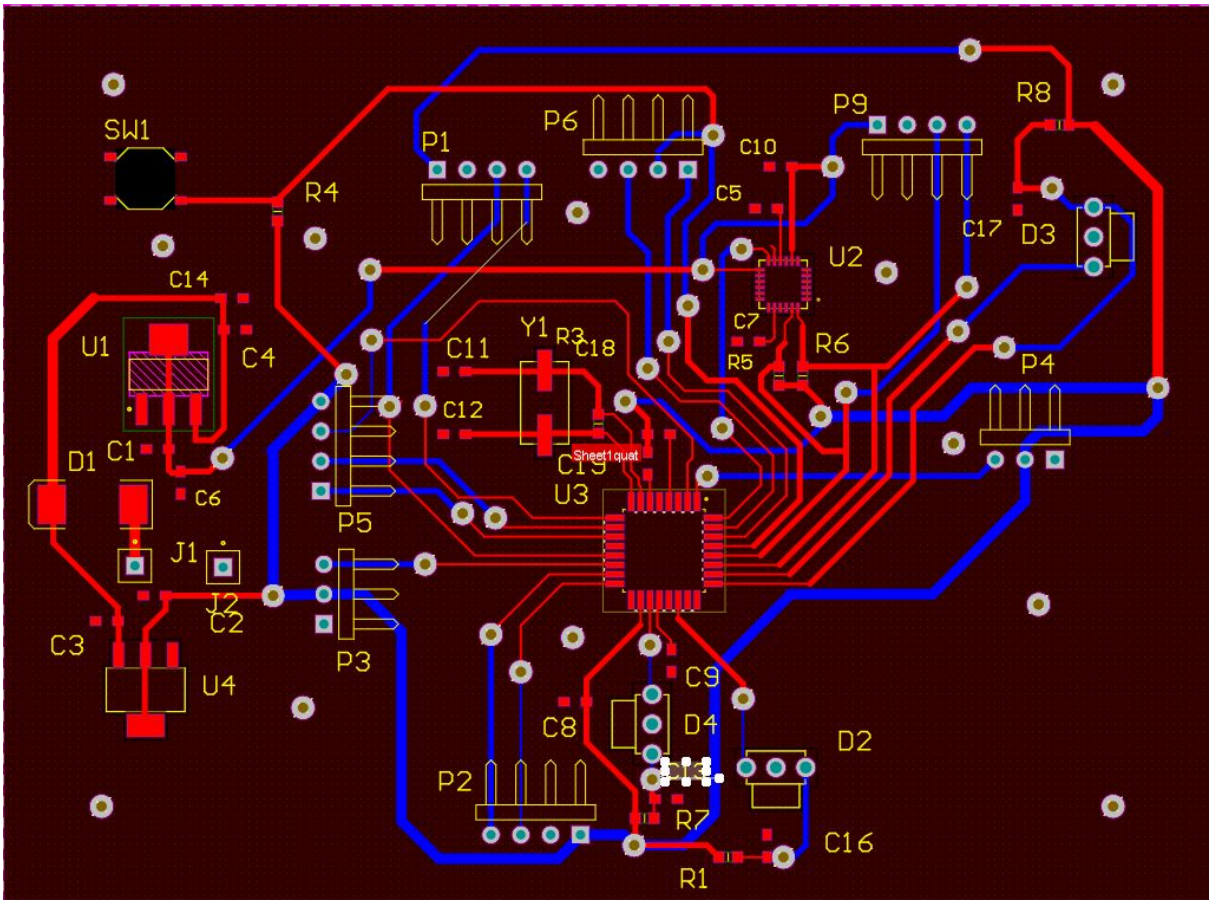


2. ATMega328P-AU



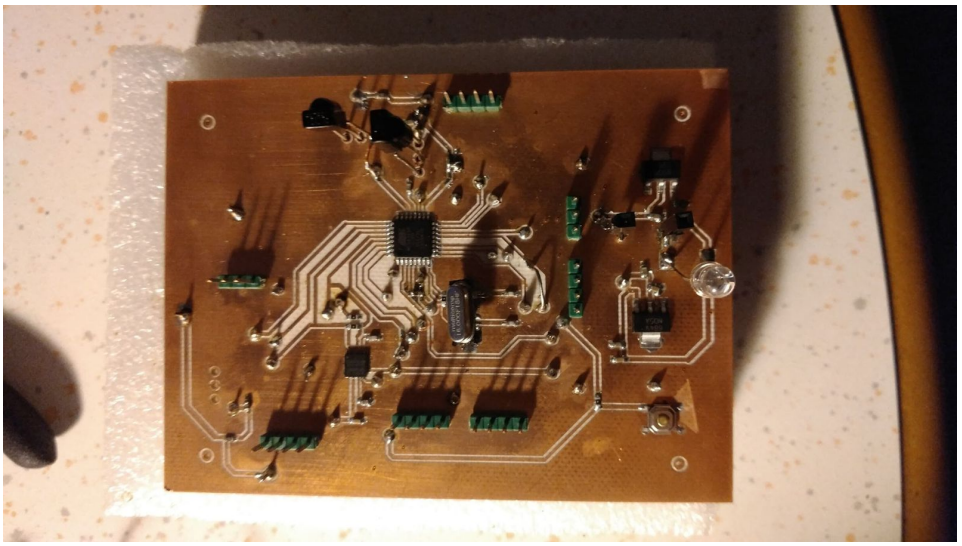
3. TSOP infra rouge et gyroscope

Après avoir réalisé les schémas de montage, je suis ensuite passé au routage de la carte. J'aurais sûrement pu améliorer ce routage en retirant le capteur du gyroscope car très petit. Cela m'aurait permis d'augmenter la distance piste/masse et ainsi faciliter les soudures.

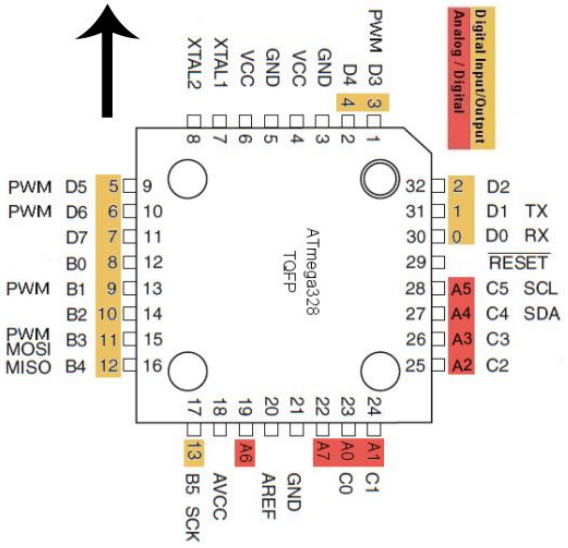


PCB routé

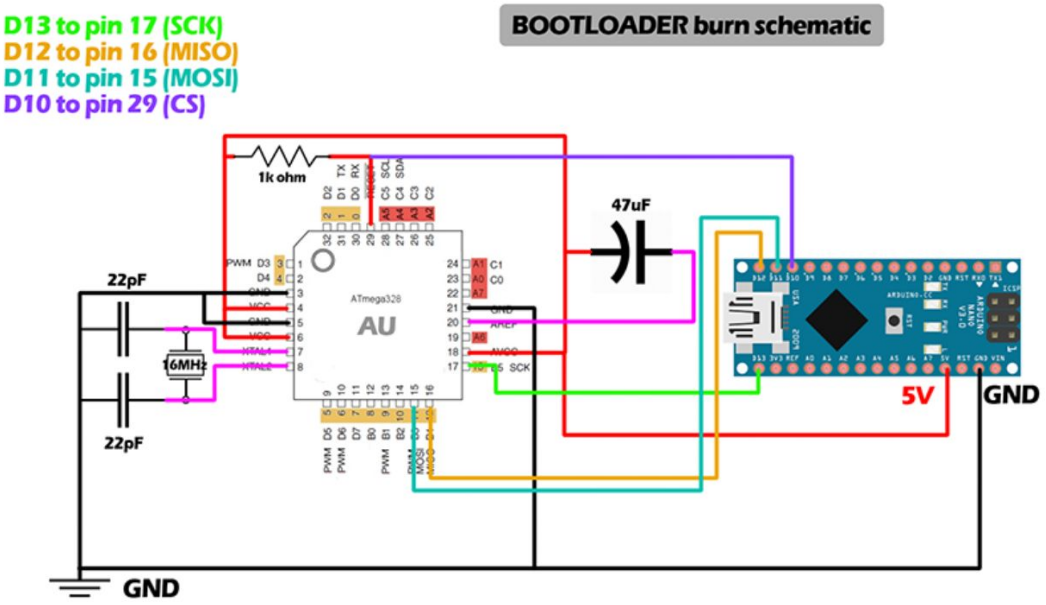
Une fois le PCB terminé sous altium, il a ensuite fallu le souder : Les composants en CMS ont été soudé à l'aide de pâte à braser et d'un four. A la sortie de celui ci, le gyroscope s'était un peu déplacé il a donc fallu le ressouder à l'aide d'une station à air chaud.



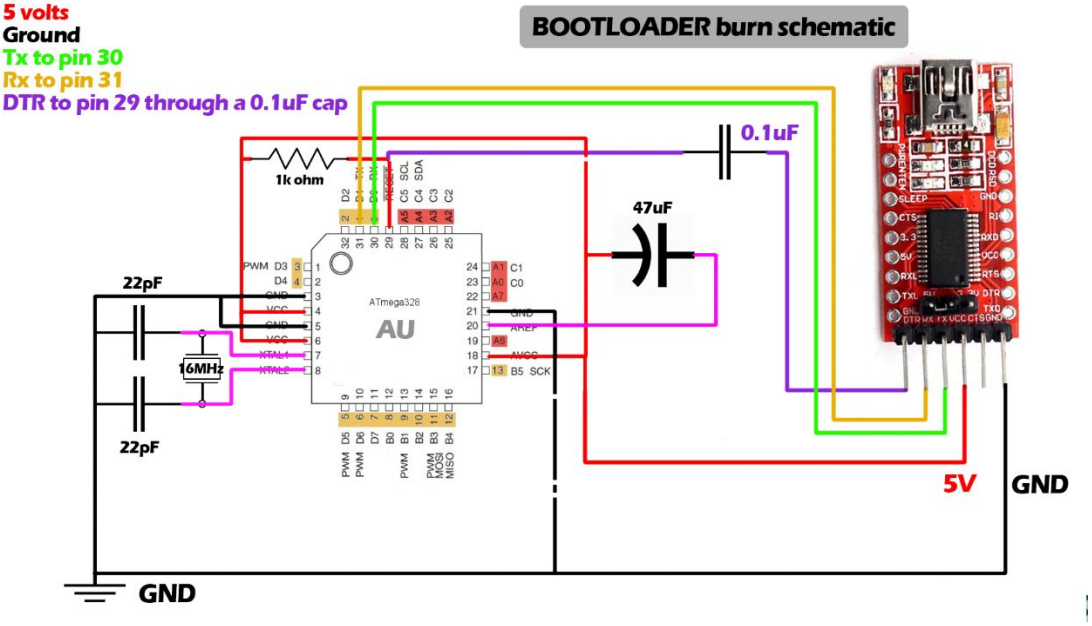
Une fois le PCB routé, de très nombreux courts circuits étaient présents sur la carte. Il m'a donc fallu les enlever afin de pouvoir alimenter la carte sans risques. La méthode pour les déceler a été la suivante : retirer les vias à des endroits stratégiques afin de pouvoir tester la ou les parties qui présentent ou non un court circuit et enfin ressouder l'élément criminel. Une fois tous les courts circuits retirés, j'ai enfin pu alimenter la carte. Une fois que j'étais sur que la carte n'allait pas fondre devant mes yeux, j'ai réalisé le bootloader de mon ATmega328P-AU. L'ATmega328P-AU est agencée de la manière suivante :



On peut voir ci contre comment l'ATmega 328P-PU est agencée avec un comparatif des pins de funarduino. A l'aide ce schéma et des deux autres ci dessous, j'ai pu effectuer le bootloader ainsi que le téléversement de programmes sur mon ATmega.



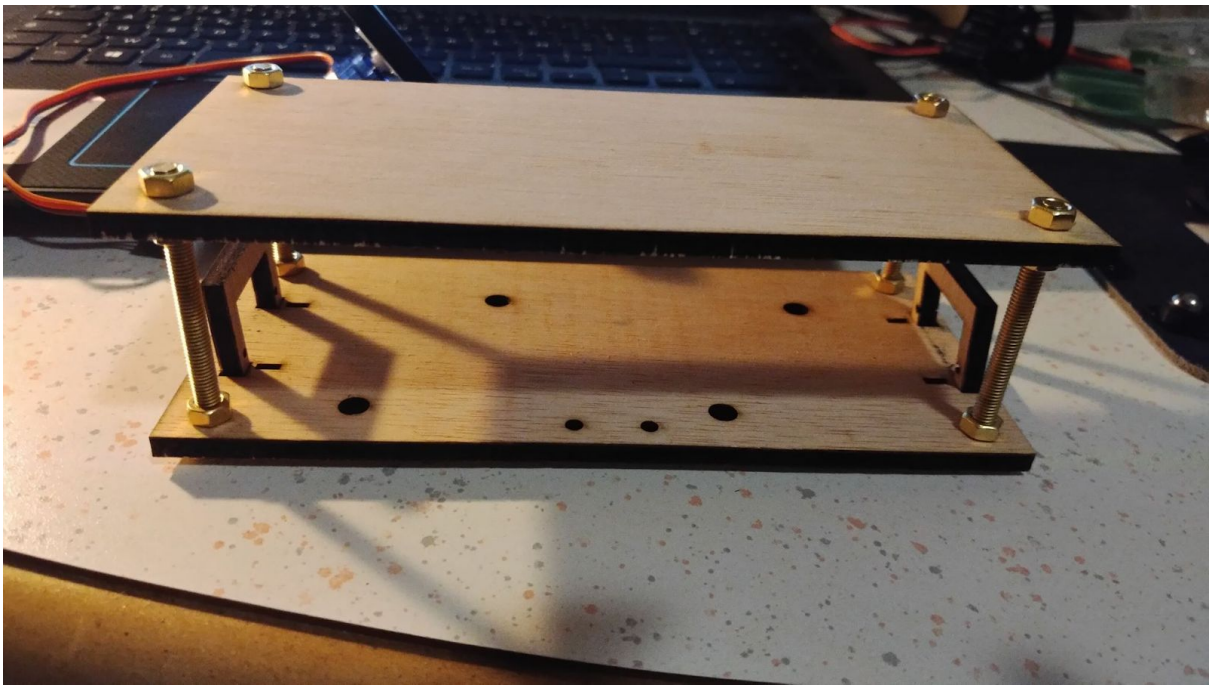
Bootloader montage



Téléversement de programmes

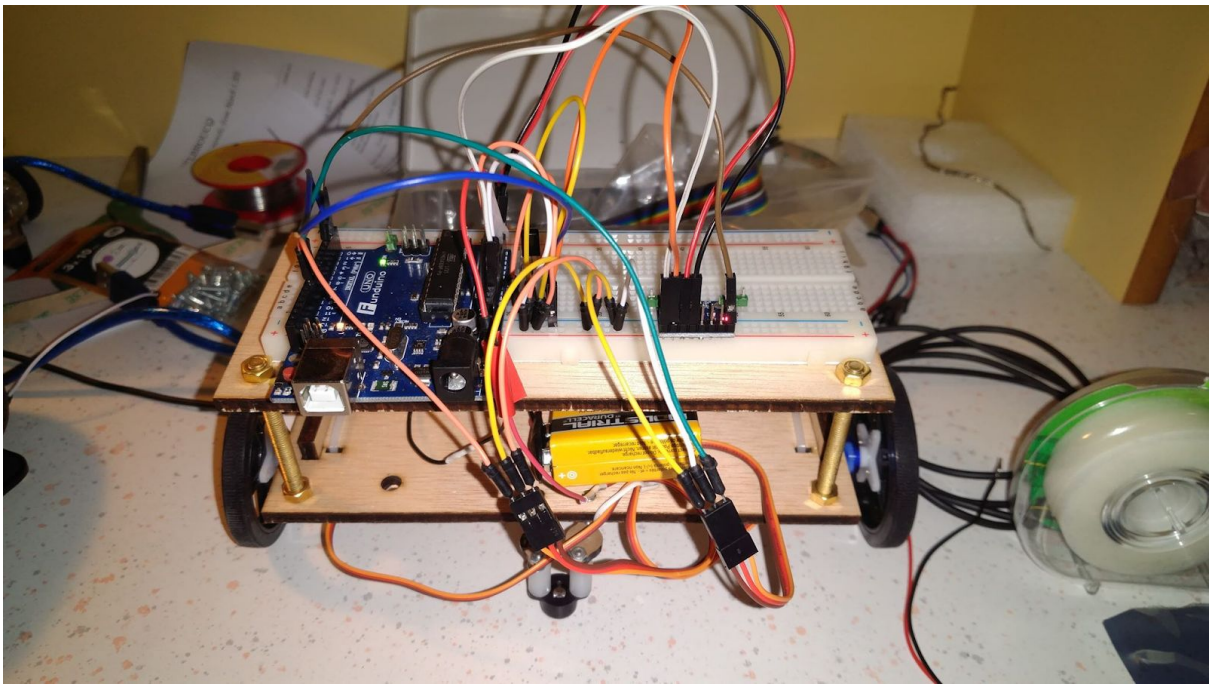
B. Le châssis

Le châssis a été réalisé et réussi du premier coup. La réalisation en contextuel de Onshape a beaucoup aidé lors de la conception des pièces notamment suivant leurs interactions dans l'espace. Une fois découpé, j'ai procédé au montage du châssis et le résultat est le suivant :



C. Codage des différentes fonctions sur un prototype.

En attendant les retours concernant la réalisation du PCB, j'ai réalisé un prototype à l'aide d'une carte arduino ainsi que d'un gyroscope MPU6050 sous forme de carte. J'ai alors fait la programmation des différentes commandes que doit réaliser le robot ce qui a été fait assez rapidement sur ce prototype :



PIP1

Le programme ci dessous utilise les interruptions sur les récepteurs infrarouges pour recevoir une consigne sous forme de code hexadécimal et l'attribut ensuite dans le switch. à la fonction concernée. Cela exécute une des commandes présente dans la liste des livrables à savoir avancer, reculer, tourner à gauche de 10°, tourner à droite de 10°.

```
#include <IRremote.h>
#include <Servo.h>
int RECV_PIN = 3
;
int outputmotG = 9; //cmd moteur gauche
int outputmotD = 10; //cmd moteur droite)
int VmotG = 90;
int VmotD = 90;
int isON = 0;
```

```
Servo ServoG;
Servo ServoD;
#define codeG  0x2FD //aller à gauche
#define codeD  0xC23D// aller à droite
#define ONOFF  0xA25D //ONOFF button
#define accel 0x906F //acceleration par palier
#define decel 0xA857 //décélération par palier
IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{

  Serial.begin(9600);
   irrecv.enableIRIn(); // Start the receiver
  pinMode(outputmotG,OUTPUT);
  pinMode(outputmotD,OUTPUT);
  ServoG.attach(3);
  ServoD.attach(10);
  ServoG.write(VmotG); //
  ServoD.write(VmotD);
}


void loop(){
  if (irrecv.decode(&results)) {
    unsigned int value = results.value;
    switch(value) {

    case ONOFF :
    if (isON == 0){
     isON = 1;
     ServoG.write(VmotG); //
     ServoD.write(VmotD);}
    else {
```

```
    isON = 0;
    VmotG=90;
    VmotD=90;
    ServoG.write(VmotG); //
    ServoD.write(VmotD);
  }
  break;
  case codeD :
  if(isON == 1) { //
    ServoD.write(VmotD*19.0/18);
    ServoG.write(VmotG*19.0/18); //
    delay(1000);
  }
  break;

  case codeG :
  if(isON == 1) { //
    ServoG.write(VmotG*17.0/18);
    ServoD.write(VmotD*17.0/18);
    delay(1000);
  }
  break;

  case accel :
  if (VmotG<=170){
    VmotG = VmotG+10;
    VmotD = VmotD-10;
    ServoG.write(VmotG);
    ServoD.write(VmotD);
    }
  break;

    case decel :
    if (VmotG>=10){
      VmotG = VmotG-10;
      VmotD = VmotD+10;
```

```
        ServoG.write(VmotG);
        ServoD.write(VmotD);
      }
      break;
    }
    ServoD.write(VmotD);
    ServoG.write(VmotG);
    Serial.println(value,HEX); // you can comment this line
    irrecv.resume(); // Receive the next value
  }
}
```

1. Régulation

La régulation a été plus compliqué à mettre en oeuvre, j'ai d'abord eu l'idée saugrenue de placer un while dans mon interruption infrarouge qui compare les valeurs de l'interruption du MPU6050 avec une valeur de base. Après avoir résolu cette erreur en opérant une modification dans mes fonctions Tourner gauche/droite qui consiste à simplement changer la valeur de la consigne, il faut maintenant observer la réponse du système en boucle fermée afin de savoir s'il est utile ou non de placer un PID. On prend aussi en compte le fait que le capteur peut être bruité. Si c'est le cas, on va par la suite ajouter un filtre de Kalman sur les valeurs de notre gyroscope afin d'obtenir le résultat le plus juste possible. On peut voir le code de la régulation en boucle fermée associé ci dessous :
PS : la calibration du MPU6050 a été réalisée avec une bibliothèque ainsi que le code d'une tierce personne

```
//bibliothèques
#include <IRremote.h>
#include <Servo.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "Wire.h"
//Variables
int RECV_PIN = 3;
```

```
int outputmotG = 9; //cmd moteur gauche
int outputmotD = 10; //cmd moteur droite)
int VmotG = 90;
int VmotD = 90;
int isON = 0;
int direc;
MPU6050 mpu;
// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from
MPU
uint8_t devStatus;      // return status after each device
operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42
bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// Orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorFloat gravity; // [x, y, z]            gravity vector
float ypr[3];           // [yaw, pitch, roll]   lacet/tangage/roulis
container and gravity vector

volatile bool mpuInterrupt = false;     // Indicates whether MPU
interrupt pin has gone high
Servo ServoG;
Servo ServoD;
//commandes ir
#define codeG  0x2FD //aller à gauche
#define codeD  0xC23D// aller à droite
#define ONOFF  0xA25D //ONOFF button
#define accel 0x906F //acceleration par palier
#define decel 0xA857 //décélération par palier
//angles MPU6050
#define YAW      0
```

```
#define PITCH    1
#define ROLL     2
IRrecv irrecv(RECV_PIN);

decode_results results;

void dmpDataReady() {
    mpuInterrupt = true;
}

void setup()
{
  Wire.begin();
  TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
  Serial.begin(9600);
  mpu.initialize();
  Serial.println(mpu.testConnection() ? F("MPU6050 connection
successful") : F("MPU6050 connection failed"));
  irrecv.enableIRIn(); // Start the receiver
  pinMode(outputmotG,OUTPUT);
  pinMode(outputmotD,OUTPUT);
  ServoG.attach(9);
  ServoD.attach(10);
  ServoG.write(VmotG); //
  ServoD.write(VmotD);
  devStatus = mpu.dmpInitialize();
  //offset mpu
  mpu.setXAccelOffset(526);
  mpu.setYAccelOffset(-467);
  mpu.setZAccelOffset(1722);
  mpu.setXGyroOffset(99);
  mpu.setYGyroOffset(14);
  mpu.setZGyroOffset(-3);
  if (devStatus == 0) {
    // Turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
```

```
    mpu.setDMPEnabled(true);

    // Enable Arduino interrupt detection
    Serial.println(F("Enabling interrupt detection (Arduino
external interrupt 0 : #pin2)..."));
    attachInterrupt(0, dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // Set our DMP Ready flag so the main loop() function knows
it's okay to use it
    Serial.println(F("DMP ready! Waiting for first
interrupt..."));
    dmpReady = true;

    // Get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
   }
   else {
     // ERROR!
     // 1 = initial memory load failed
     // 2 = DMP configuration updates failed
     // (if it's going to break, usually the code will be 1)
     Serial.print(F("DMP Initialization failed (code "));
     Serial.print(devStatus);
     Serial.println(F(")"));
   }

}


void loop(){
 // If programming failed, don't try to do anything
    if (!dmpReady) {
        return;
    }
```

```
    // Wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // Do nothing...
    }

    // Reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // Get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // Check for overflow (this should never happen unless our
code is too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));

        // Otherwise, check for DMP data ready interrupt (this
should happen frequently)
    } else if (mpuIntStatus & 0x02) {
        // Wait for correct available data length, should be a
VERY short wait
        while (fifoCount < packetSize) {
            fifoCount = mpu.getFIFOCount();
        }

        // Read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // Track FIFO count here in case there is > 1 packet
available
        // (this lets us immediately read more without waiting for
an interrupt)
        fifoCount -= packetSize;
```

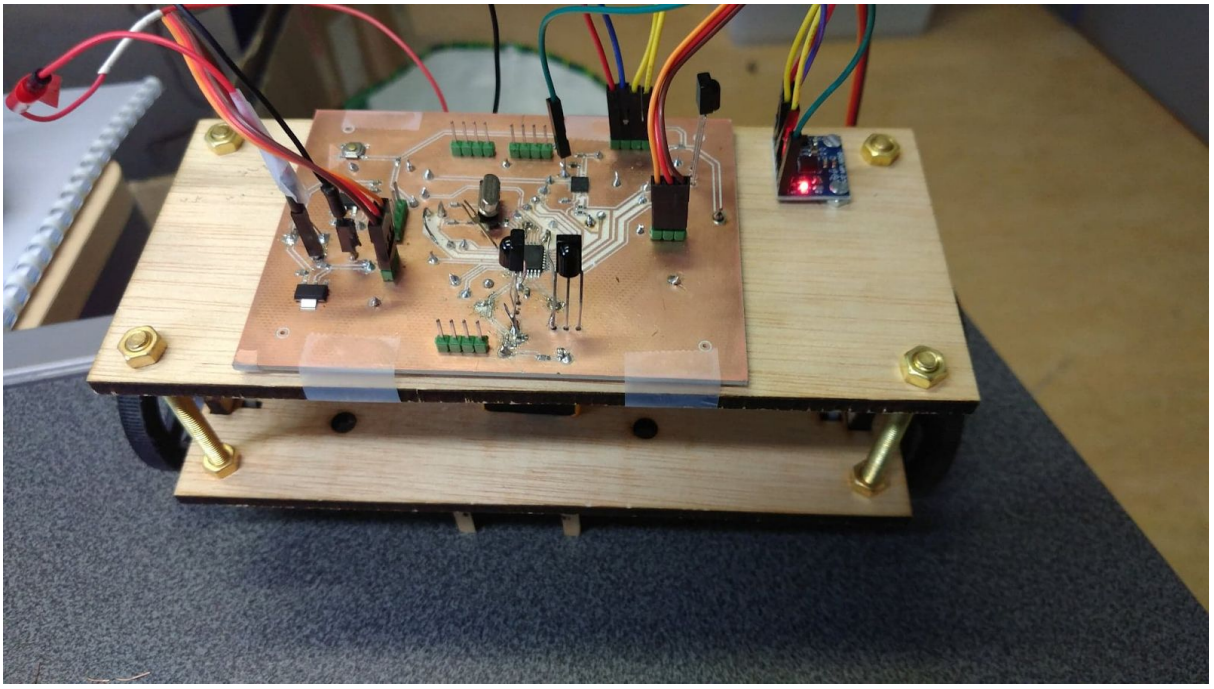
```
        // Convert Euler angles in degrees
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
        // Print angle values in degrees.
       /* Serial.print(ypr[YAW] * (180.0 / M_PI));
        Serial.print("\t");
        Serial.print(ypr[PITCH] * (180.0 / M_PI));
        Serial.print("\t");
        Serial.println(ypr[ROLL] * (180.0/ M_PI));*/
    }
  if (irrecv.decode(&results)) {
    unsigned int value = results.value;
    switch(value) {

    case ONOFF :
    if (isON == 0){
     isON = 1;
     ServoG.write(VmotG); //
     ServoD.write(VmotD);
     direc=ypr[0]*180.0/M_PI;
    }
    else {
      isON = 0;
      VmotG=90;
      VmotD=90;
      ServoG.write(VmotG); //
      ServoD.write(VmotD);
    }
    break;
    case codeD :
    if(isON == 1) {//
      direc+=10;
    }
```

```
    break;

    case codeG :
    if(isON == 1) {
      direc-=10;
    }
    break;

    case accel :
    if (VmotG<=170){
      VmotG = VmotG+10;
      VmotD = VmotD-10;
      ServoG.write(VmotG);
      ServoD.write(VmotD);
      }
    break;

      case decel :
      if (VmotG>=10){
        VmotG = VmotG-10;
        VmotD = VmotD+10;
        ServoG.write(VmotG);
        ServoD.write(VmotD);
      }
   break;
   }
   ServoG.write(VmotG - (direc-ypr[0]*180.0/M_PI)/18);
   ServoD.write(VmotD - (direc-ypr[0]*180.0/M_PI)/18);
   Serial.println(VmotD);
   Serial.println(VmotG);
   Serial.println(value,HEX); // you can comment this line
   irrecv.resume(); // Receive the next value
  }
}
```

Vidéo du test sur PIP1 :

https://drive.google.com/file/d/1JhL-7_rXhMDk4sk4wrQNET_57lx_ZAjY/view?usp=drivesdk

Une fois la carte routée, j'ai remplacé la breadboard par la carte et envoyé les programmes sur mon circuit. Le rendu est le suivant :



Le programme fonctionnait à la dérive prêt des valeurs du MPU (il manquait un filtre de Kalman pour stabiliser les valeurs) sur le prototype avec breadboard et Funduino mais il semblerait qu'il y ait des problèmes de connexion sur la connectique carte/MPU6050.

En effet, le problème vient sûrement de la connexion à l'interruption 0 (pin 31 de l'ATMEGA328P-AU ou pin 2 de l'arduino).

Le filtre de Kalman consiste à supposer une valeur et réduire ou augmenter la valeur mesurée afin de coller aux prédictions faites. Cela permet de limiter grandement le bruit pour obtenir des valeurs proches de la réalité.

Après quelques débogages et modifications du circuit, le premier programme fonctionnant sans régulation marche sans problèmes. Les complications arrivent avec le gyroscope. J'ai retiré le gyroscope de invensense pour placer le gyroscope au format carte que j'ai utilisé pour réaliser mes tests précédents. Toutes les

connectiques semblent être bonnes aux vues des résultats du multimètre mais il reste peut être la qualité de ces connexions qui reste à douter.

Conclusion :

Pour conclure, ce projet m'a permis d'améliorer mes compétences en création de PCB ainsi qu'en soudures de manière très importante. Je trouve ça dommage de ne pas avoir eu le temps de réaliser la régulation de l'auto balancement du robot qui aurait pu ajouter un attrait mécanique et d'automatique au projet. C'est agréable de mener un projet en autonomie et d'en voir la fin arriver même si j'aimerais bien essayer de le terminer avant les stages. Les problèmes que j'ai pu rencontrer sont la création de PCB qui m'a pris beaucoup trop de temps. De plus, après avoir soudé on se rend compte que certaines soudures sont plus ou moins optimales. Par contre comme mon PCB était mal conçu au départ et unique vu mon retard j'ai pu apprendre énormément de choses en ce qui concerne l'élimination des courts circuits ou les différentes techniques pour réparer un PCB avec peu de moyens (même si pour une production à plus grande échelle je doute que passer 30h sur la réparation d'un PCB soit une grande chose.). C'est pourquoi je m'arrangerais pour que le prochain soit mieux amené que celui ci. La partie capteurs de souris n'a pas été faite par manque de temps et surtout parce que le gyroscope envoyait des données exploitables qui pouvaient être améliorées à l'aide d'un filtre.