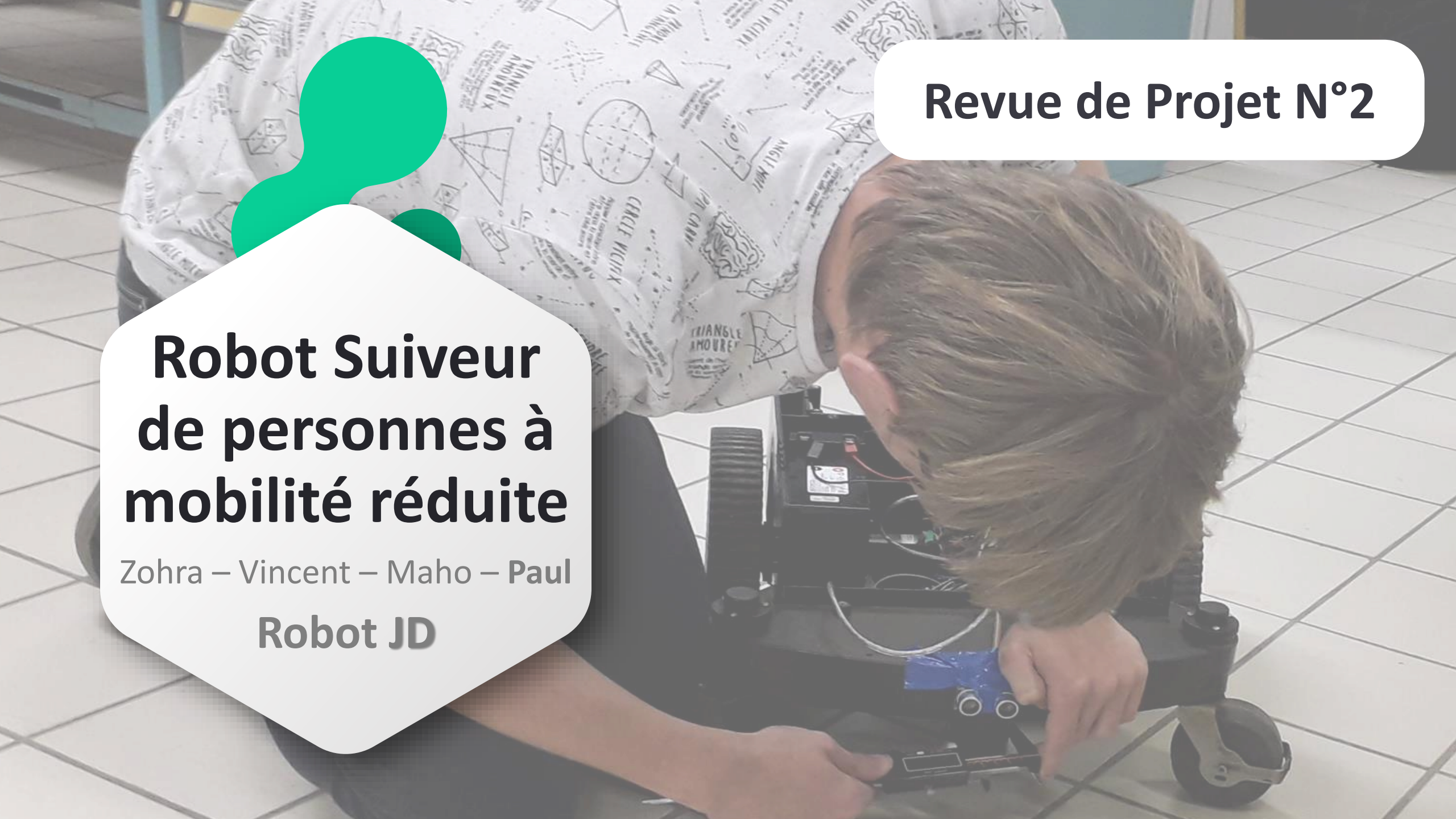


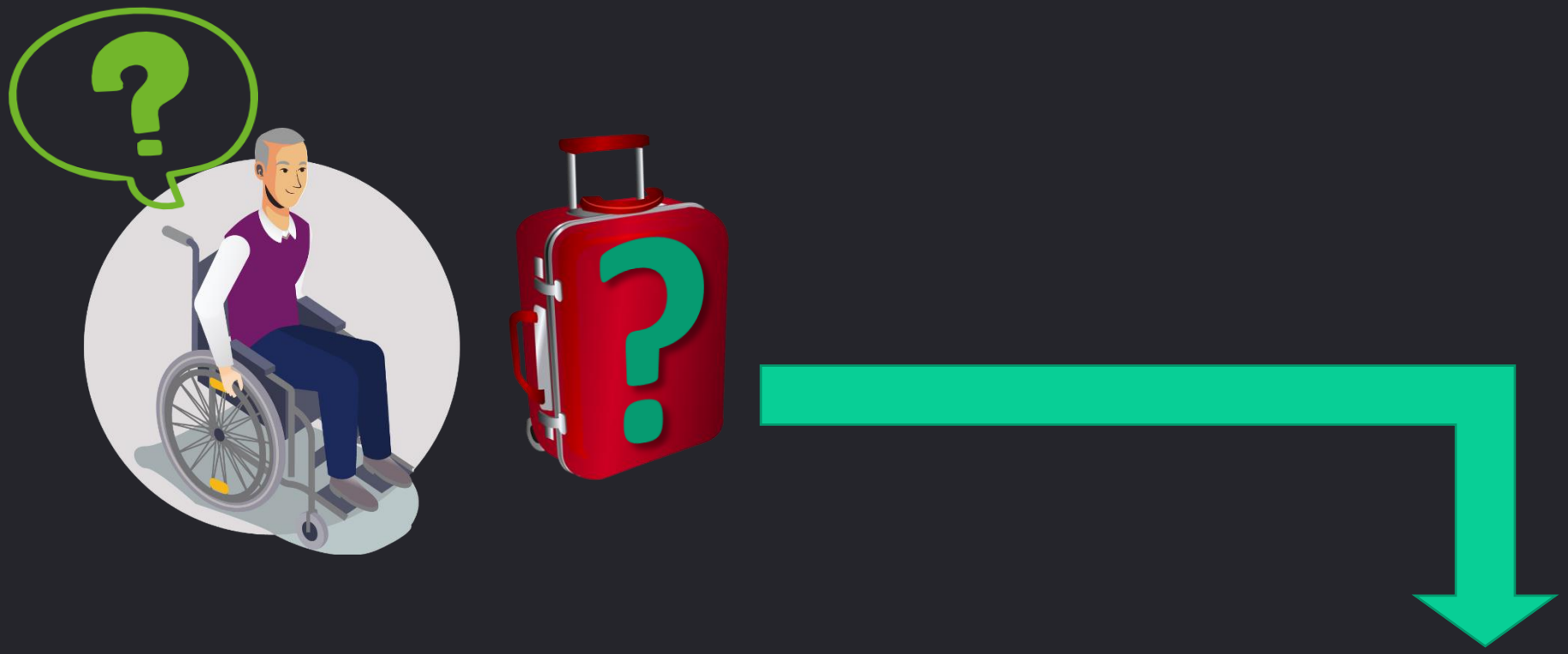
Revue de Projet N°2

Robot Suiveur de personnes à mobilité réduite

Zohra – Vincent – Maho – **Paul**

Robot JD

Sommaire :




#02 :
Analyse
Fonctionnelle

#02

Personne à mobilité réduite

Bagages

?

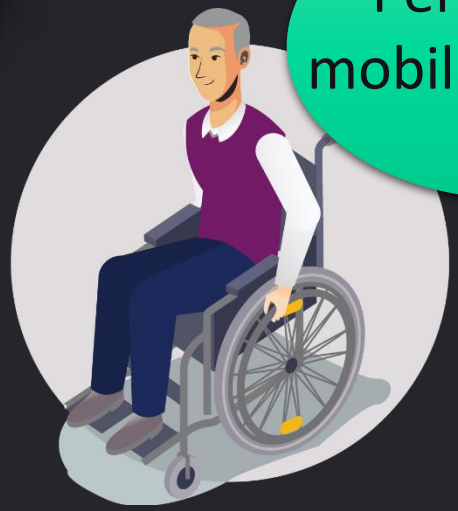
TRANSPORTER les bagages et **SUIVRE** la personne à mobilité réduite

#02

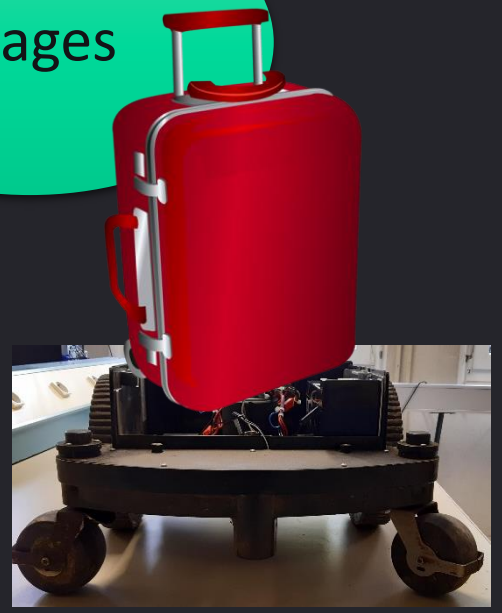
Solution	Avantage	Inconvenian
Aide humaine	Presence humaine Securiser Aucun efort à produire pour l'utilisateur	Cout elever et non rentabilisable Efort important pour l'employer
Chariot à roulette tirer par l'utilisateur	Peu cher Simple d'utilisation	Danger pour l'utilisateur et les personnes dans la gare Effort de traction à fournir
Chariot avec assistance électrique sécurisées	Prix moyen Securiser Efort reduit	Complex d'utilisation
Chariot suiveur automatique **(Solution retenu)**	Securiser Simple d'utilisation Aucun efort à produire	Cout elever

#02

Personne à mobilité réduite



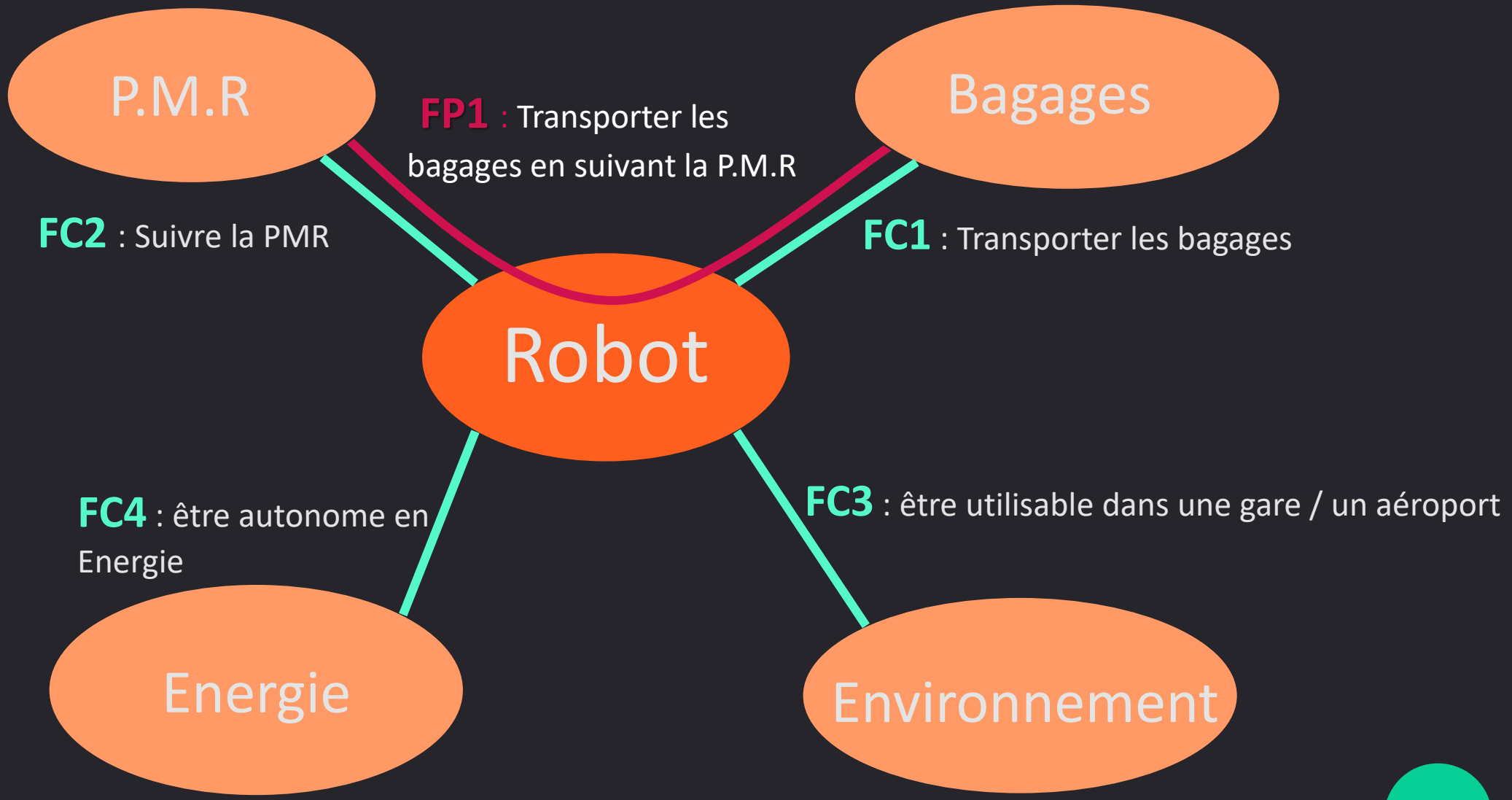
Bagages



Robot suiveur

TRANSPORTER les bagages et **SUIVRE** la personne à mobilité réduite

#02


P.M.R
Bagages
FP1 : Transporter les bagages en suivant la P.M.R
FC2 : Suivre la PMR
FC1 : Transporter les bagages
Robot
FC4 : être autonome en Energie
FC3 : être utilisable dans une gare / un aéroport
Energie
Environnement

#02

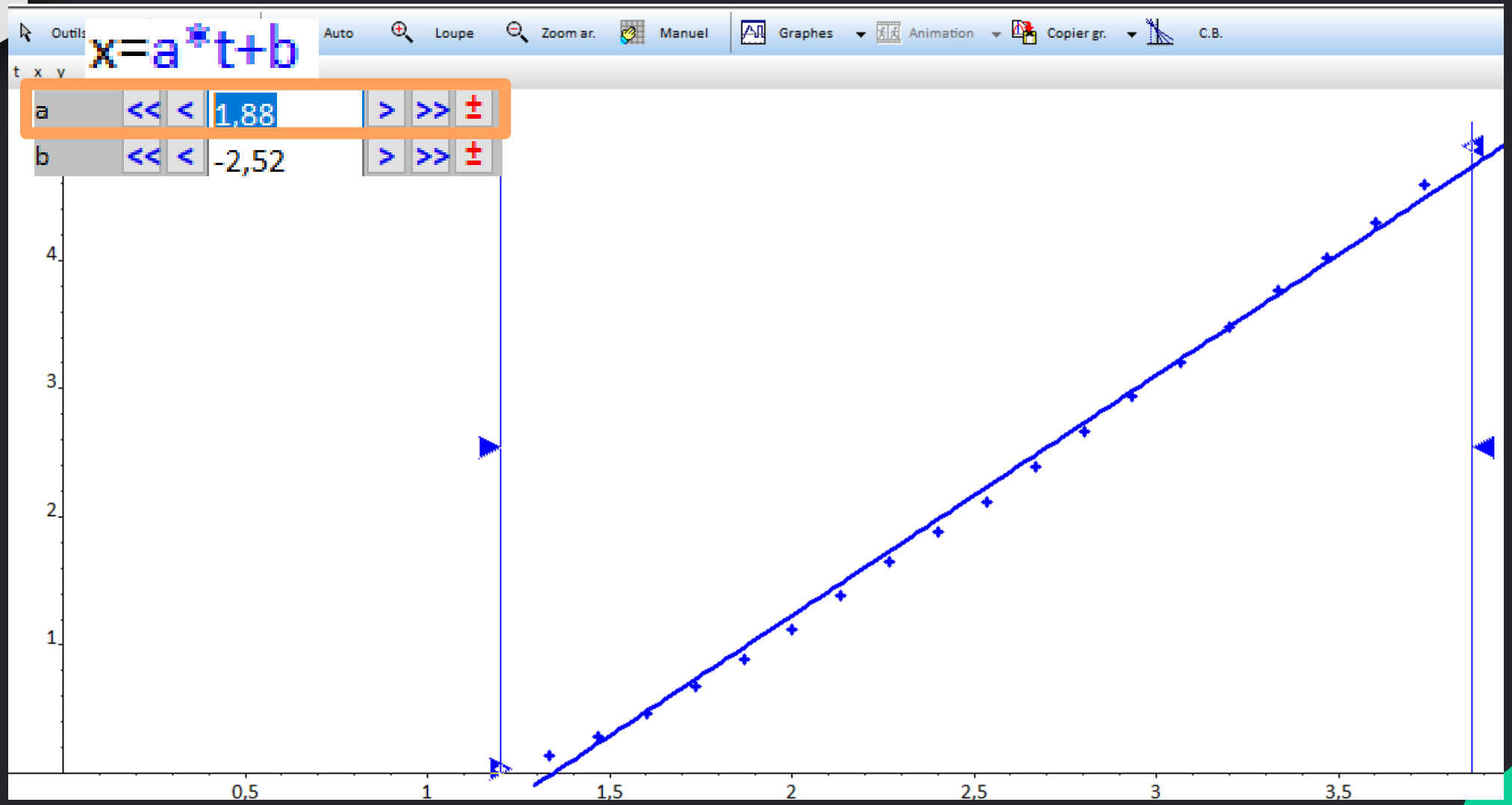
FP1	Transporter des bagages en suivant la P.M.R			F0
FC1	Transporter les bagages	Poids du bagage	<30 kg	F1
FC2	Suivre la personne à mobilité réduite	Vitesse maximale Température environnement Distance	>=2m/s <24°c <200cm	F0 F3 F2
FC3	Etre utilisable dans une gare / un aéroport	Masse Maximale	<10kg	F2
FC4	Être autonome en énergie	temps d'autonomie minimal en utilisation	6H	F2

#02 Cahiers des Charges : Définir la vitesse

Bâton de mesure de 1,42m

#02

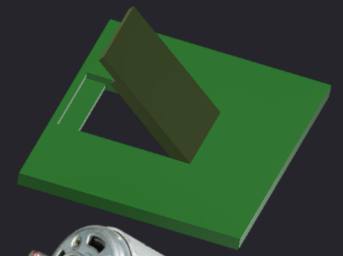
Cahiers des Charges : Définir la vitesse

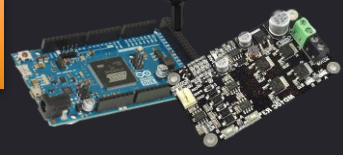


#02

FP1 : Transporter les bagages en suivant la P.M.R

- **FC1 : Transporter les bagages**
  - **FT1 : contenir les bagages** → **plateforme**
  - **FT2 : déplacer les bagages** → **moteur**
- **FC2 : Suivre la P.M.R**
  - **FT3 : définir la présence** → **Capteur thermique**
  - **FT4 : définir la localisation** → **LIDAR**
  - **FT5 : ordonner le déplacement** → **Carte Microcontrôleur Pont en « H »**
- **FC3 : être utilisable dans une gare/ aéroport**
  - **FT6 : Assurer la sécurité** → **Fin de course**
  - **FT6 : permettre une maintenance ergonomique** → **poignée**
  - **FT7 : éviter les obstacles environnants** → **LIDAR + Carte microcontrôleur**
  - **FT8 : être autonome en énergie** → **Batterie**



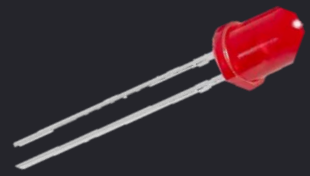





#02

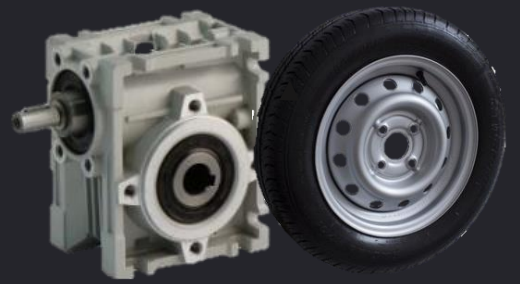
Information

Acquérir → Traiter → Communiquer



Energie

Alimenter → Distribuer → Convertir → Transmettre



Energie Electrique

Energie Mécanique de Rotation

#03 : Taches

REMY Vincent

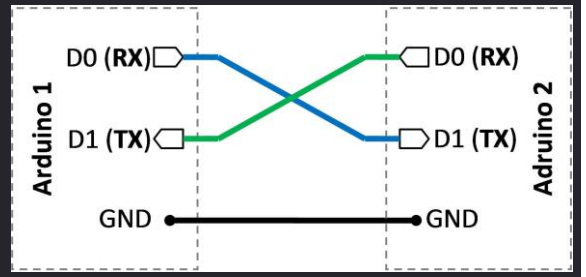
Obtenir la distance et l'angle

GROSDEMANGE Maho

Détecter la personne

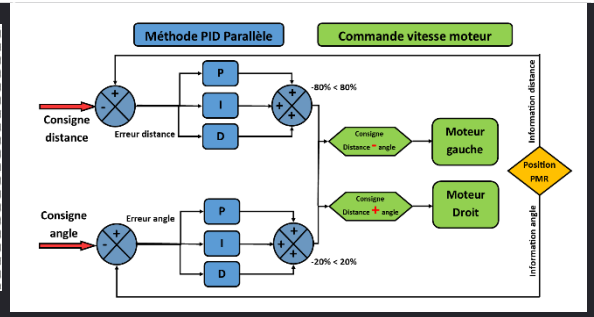
POPART Zohra

Collecter les données



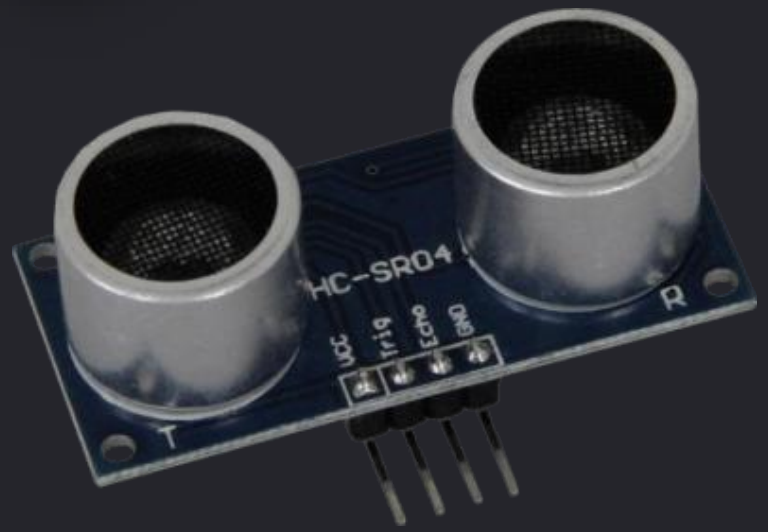
GIROUX Paul

Asservir le Robot



#03

Asservissement en 2 étapes :

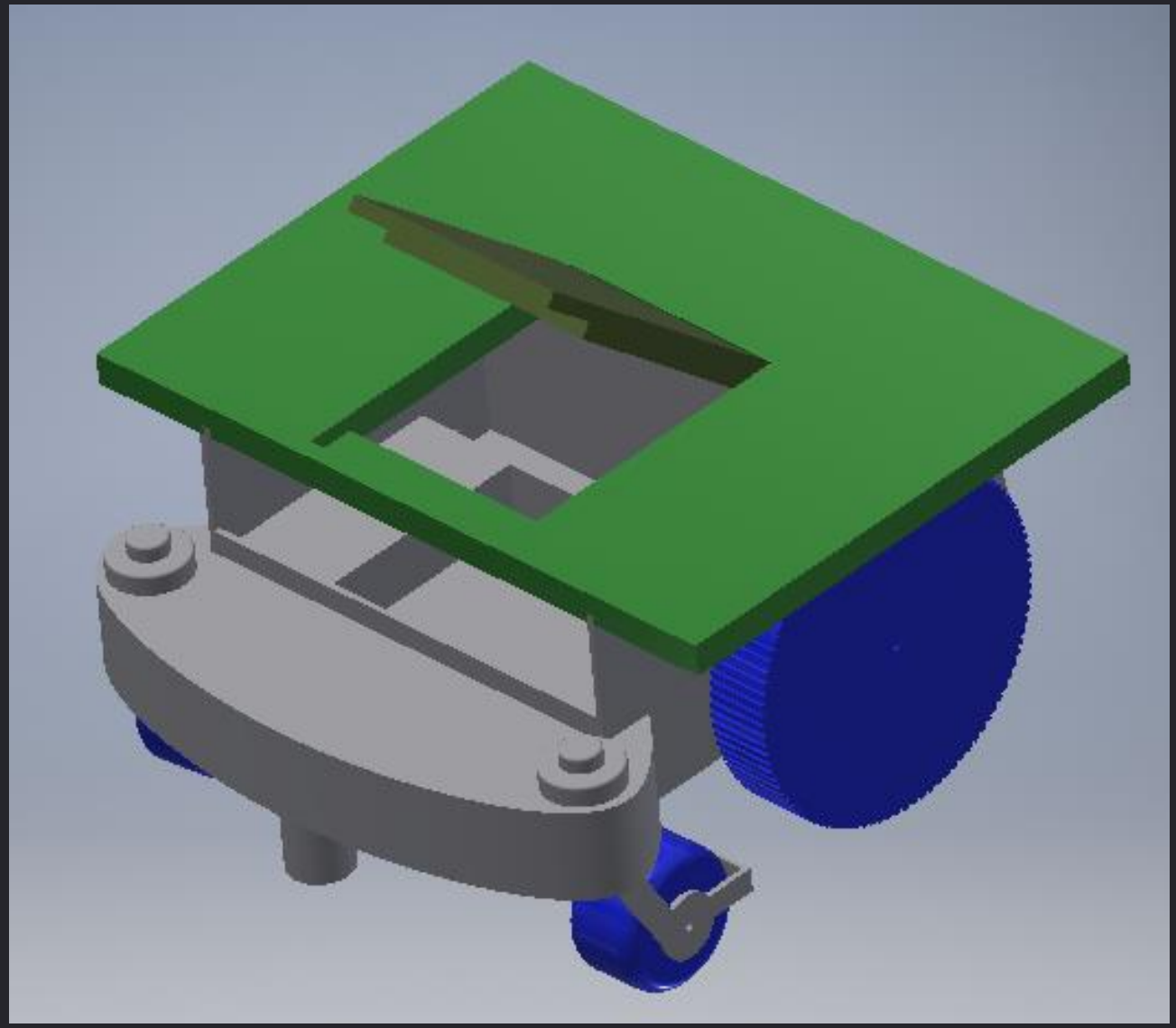


monodimensionnel

2D

#03

Modélisation : Assemblage



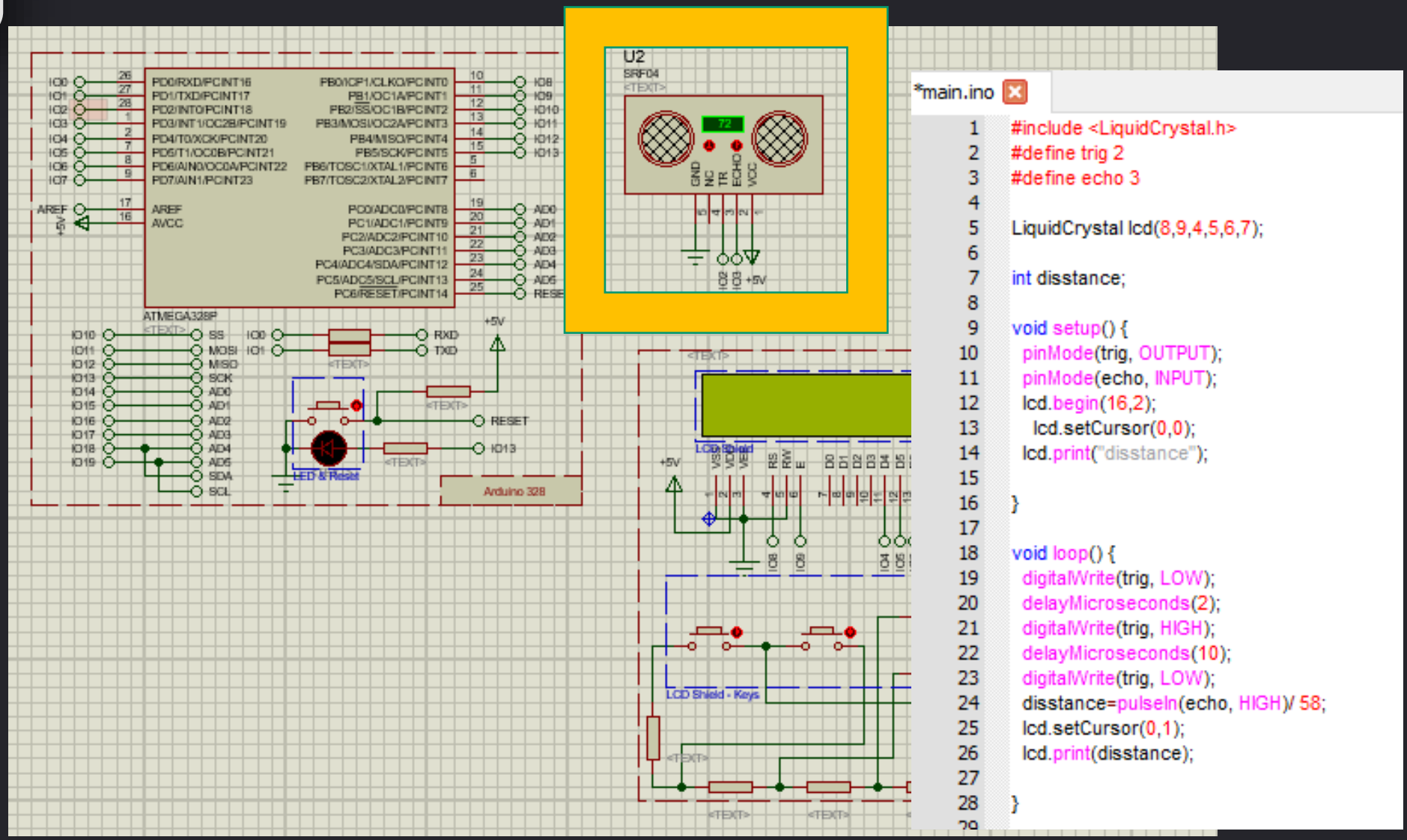
#04 : Simulation et Modelisation

Proteus

MapleSim

Inventor

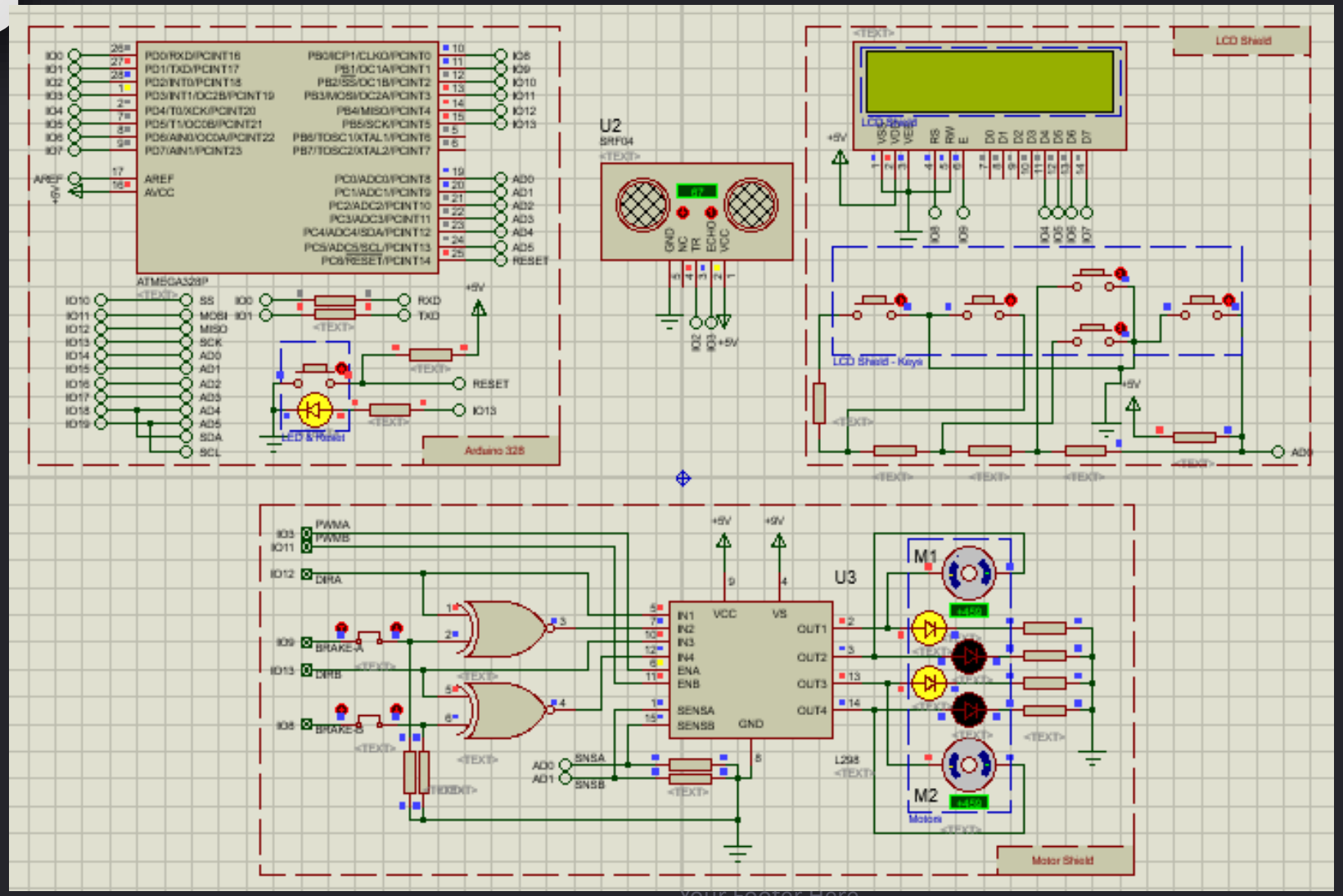
#04 Simulation Proteus : capteur Ultra-Son



*main.ino

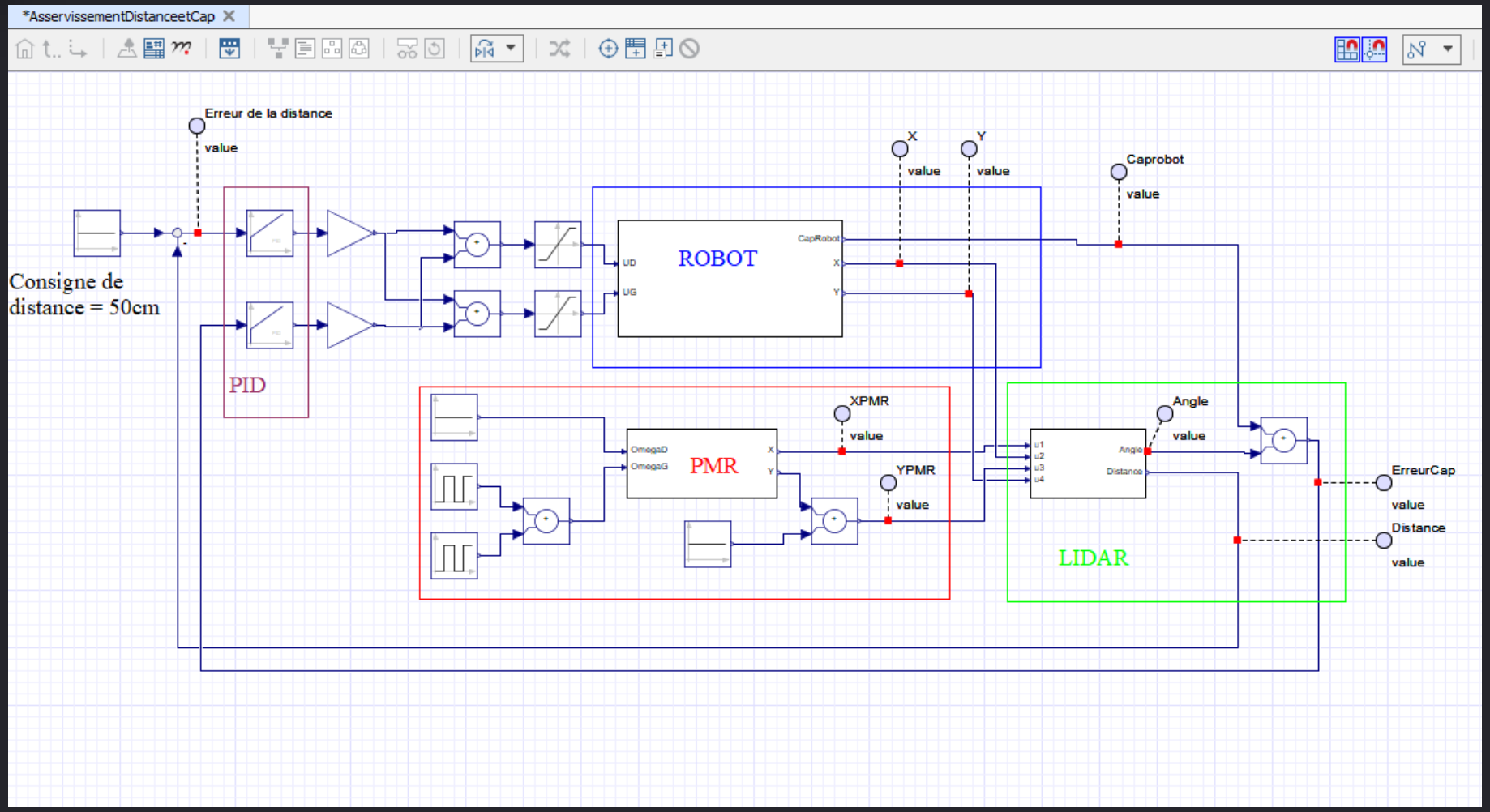
```
1  #include <LiquidCrystal.h>
2  #define trig 2
3  #define echo 3
4
5  LiquidCrystal lcd(8,9,4,5,6,7);
6
7  int disstance;
8
9  void setup() {
10   pinMode(trig, OUTPUT);
11   pinMode(echo, INPUT);
12   lcd.begin(16,2);
13    lcd.setCursor(0,0);
14   lcd.print("disstance");
15
16 }
17
18 void loop() {
19   digitalWrite(trig, LOW);
20   delayMicroseconds(2);
21   digitalWrite(trig, HIGH);
22   delayMicroseconds(10);
23   digitalWrite(trig, LOW);
24   disstance=pulseIn(echo, HIGH)/ 58;
25   lcd.setCursor(0,1);
26   lcd.print(disstance);
27
28 }
```

#04 Simulation Proteus : asservissement




```
void setup() {
 TCCR1B = TCCR1B & 0b11111000  | 0x01;
 pinMode(pwmmd,OUTPUT);
 pinMode(pwmmg,OUTPUT);
 pinMode(dirmd,OUTPUT);
 pinMode(dirmg,OUTPUT);
 pinMode(trig,OUTPUT);
 pinMode(echo,INPUT);
 Serial.begin(9600);
}
void loop() {
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  distance = pulseIn(echo, HIGH) / 58;
  if (distance<500){

    erreur=distance-distance_cst;
    temps_actuel=millis();
    temps=temps_actuel-temps_precedent;
    integrer += erreur*temps;
    deriver = (erreur-erreur_precedente)/te
    erreur_precedente=erreur;
    temps_precedent=temps_actuel;
    action=erreur*kp+integrer*ki+deriver*kd;

    if(erreur>0){
      digitalWrite(dirmd, LOW);
      digitalWrite(dirmg, LOW);
      Serial.print("av");
```

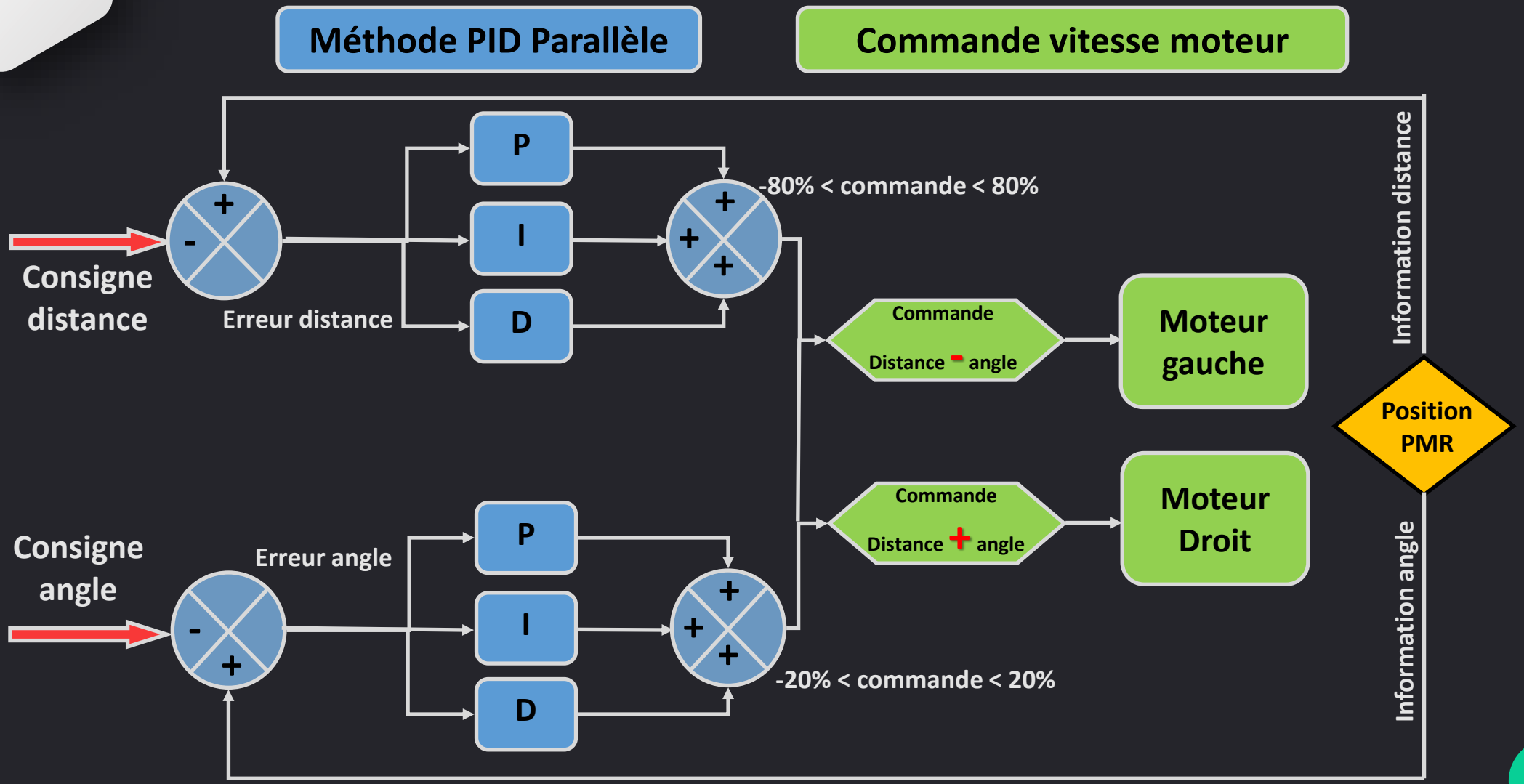
#04 Simulation MapleSim




#05 :
Asservissement

#05

L’asservissement



#05

Consigne Distance : 50cm
Consigne Angle : 0°

Distance > 50cm
Erreur Distance > 0cm

Angle < 0°
Erreur Angle < 0°

0°

ROBOT

80% > Commande Distance > 0%
0% > Commande Angle > -20%

Commande
Distance - angle

Moteur gauche

Moteur Gauche > Moteur Droit

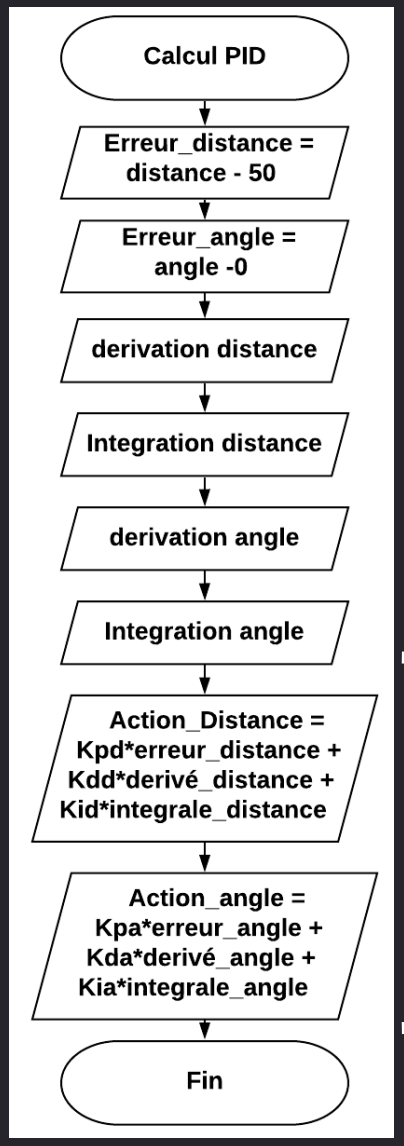
Commande
Distance + angle

Moteur Droit

#05

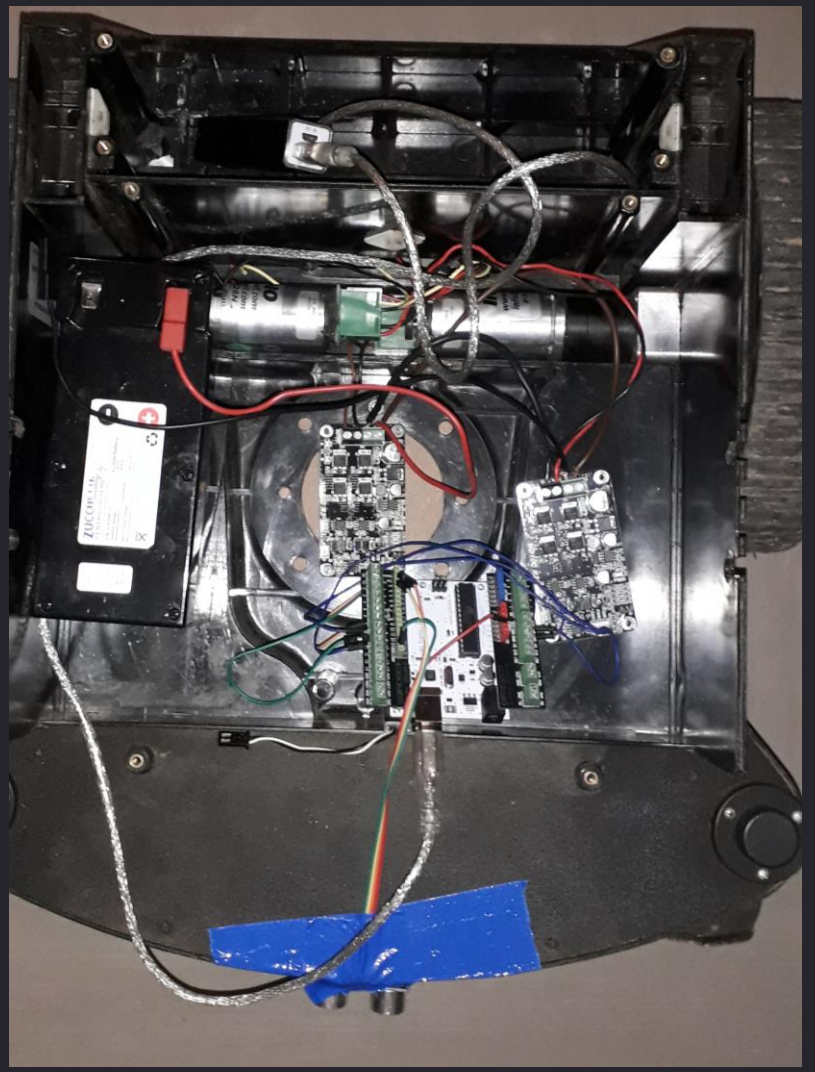
Code Arduino : 125 lignes

```
void calculePID(){
    temps_actuel=millis();
    temps=temps_actuel-temps_precedent;

    erreurd=abs(distance-distance_cst);
    erreura=angle-angle_cst;

    integrerd += erreurd*temps;
    deriverd = (erreurd-erreurd_precedente)/temps;

    integrera += erreura*temps;
    derivera = (erreura-erreura_precedente)/temps;

    erreurd_precedente=erreurd;
    erreura_precedente=erreura;
    temps_precedent=temps_actuel;

    action_distance=erreurd*kpd+integrerd*kid+deriverd*kdd;
    action_angle=erreura*kpa+integrera*kia+derivera*kda;

  }
```



Kp, Kd et Ki sont définit en distance et en angle par test successif

#06 PID Monodimensionnel : Capteur Ultra-Son

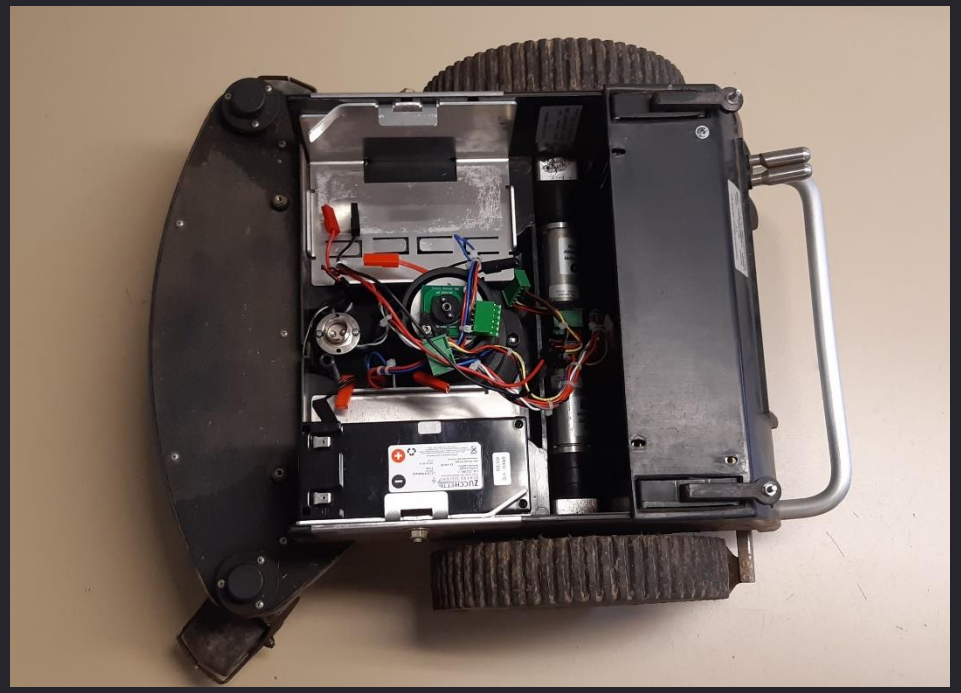


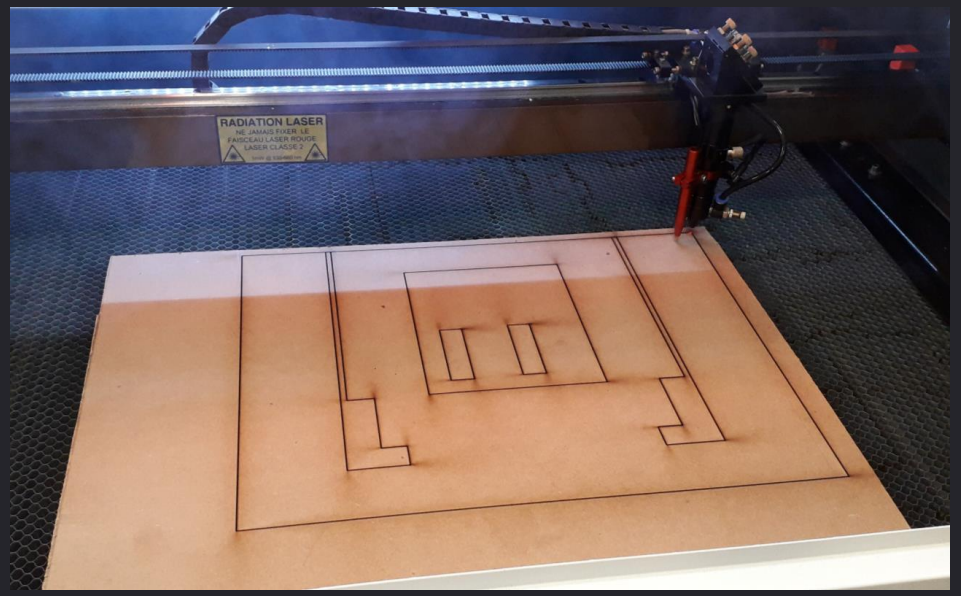
#06 PID Monodimensionnel : Capteur Ultra-Son

#06

PID 2D : LIDAR

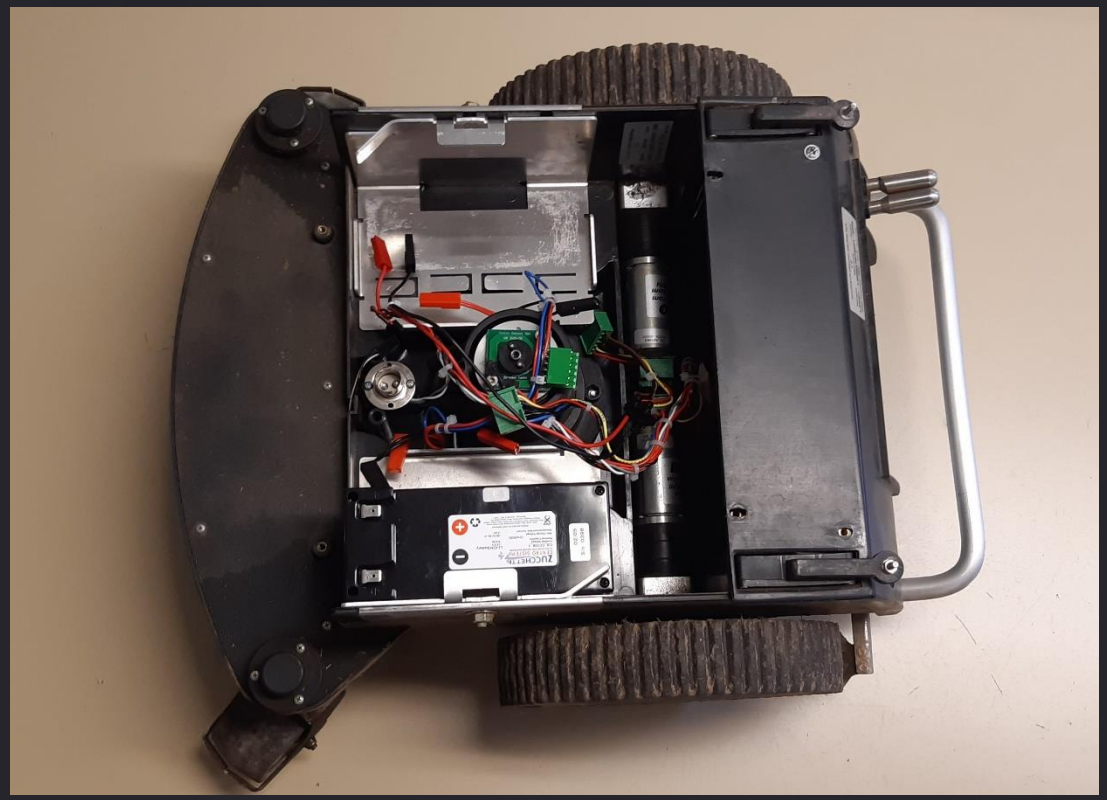
#06 : Fabrication

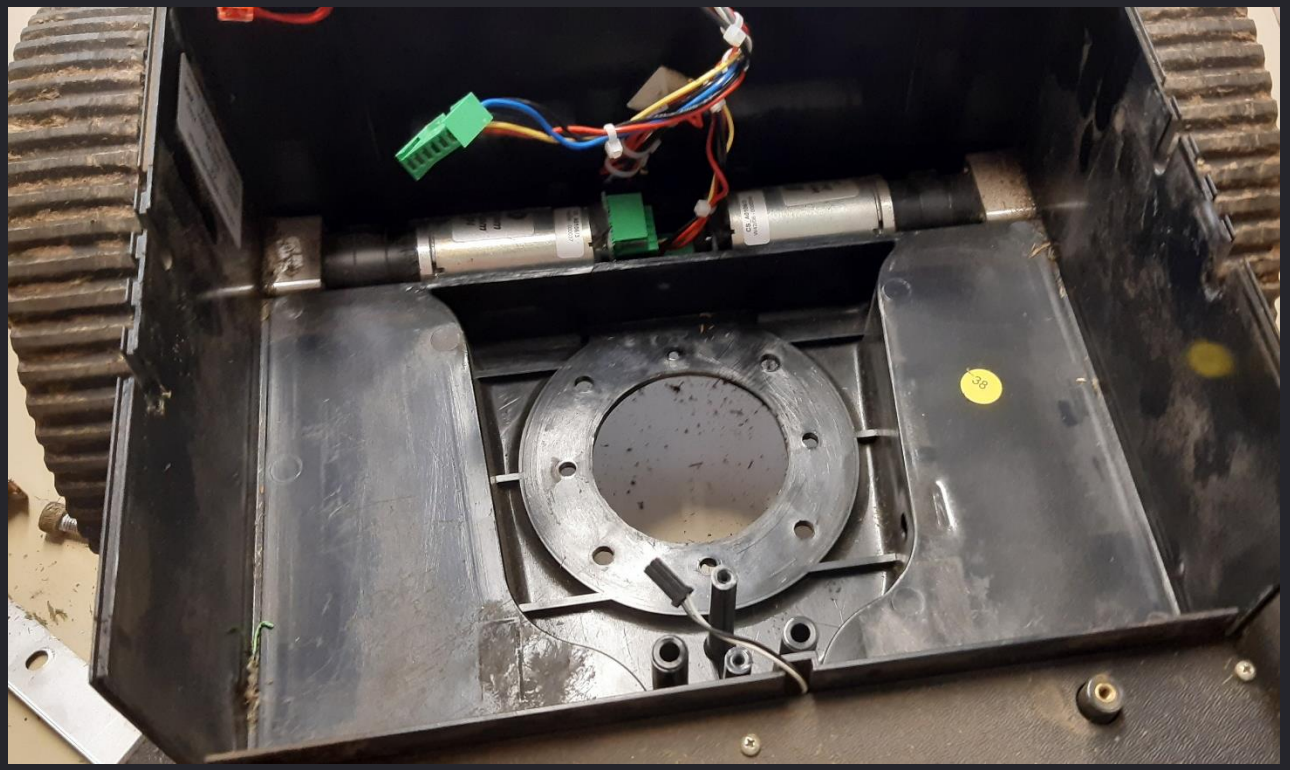




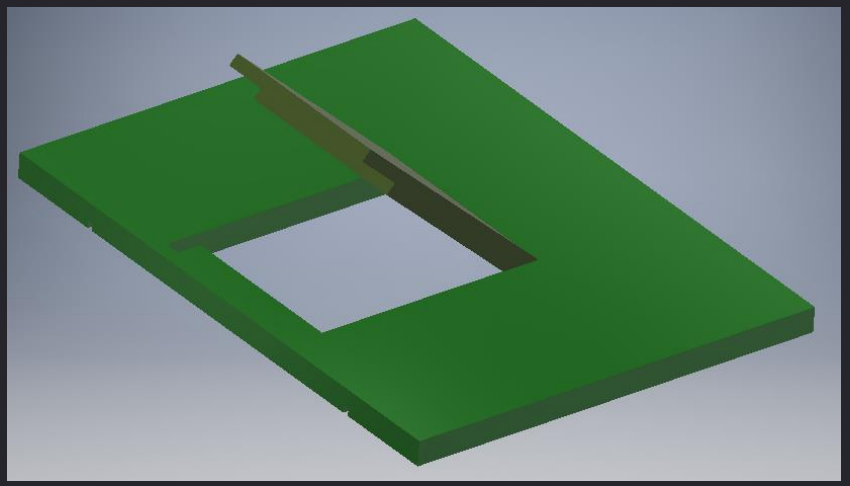
#06

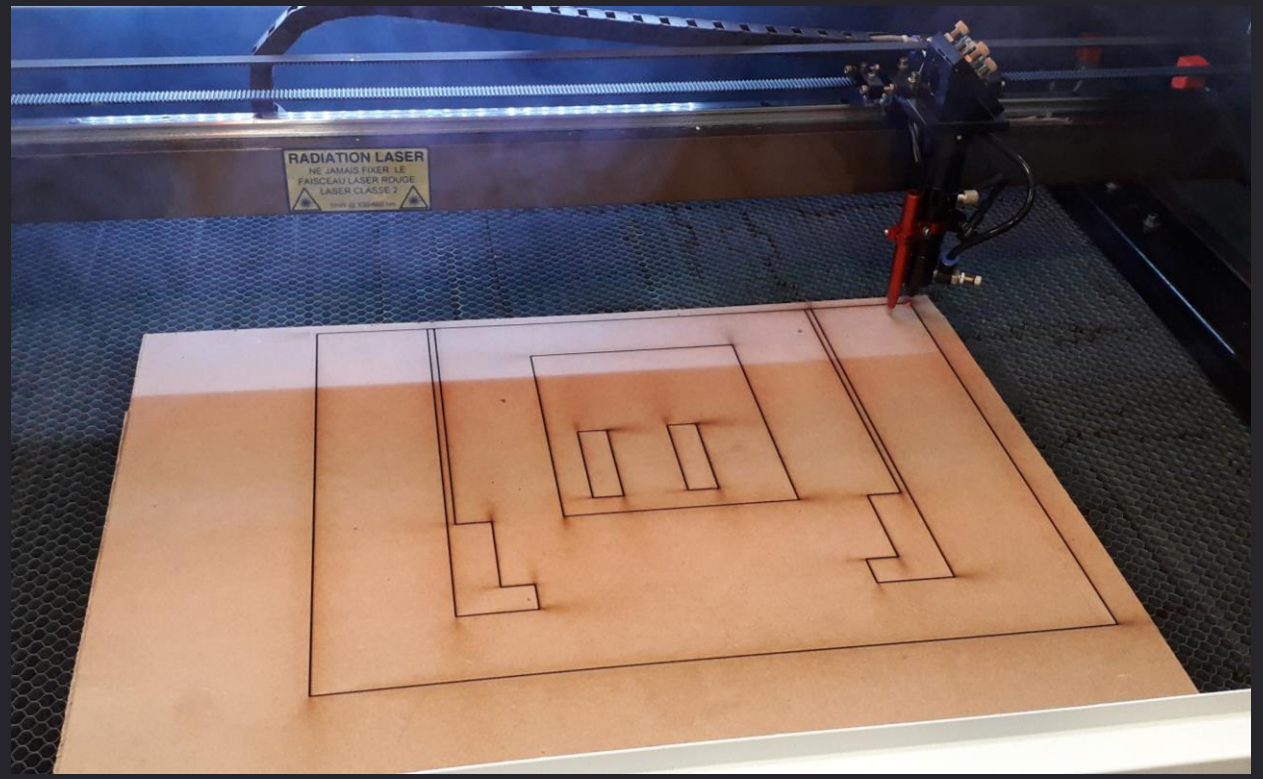
La base

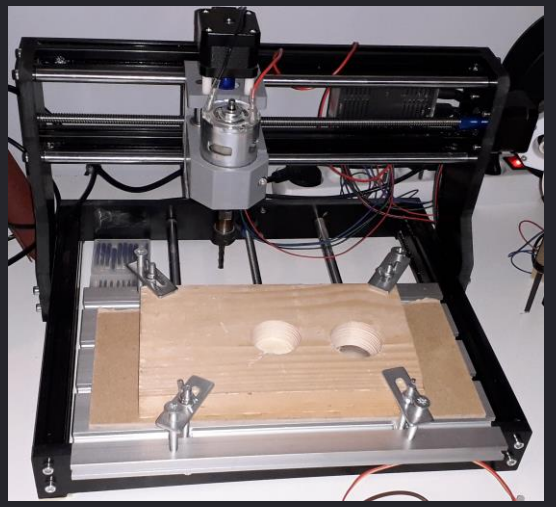




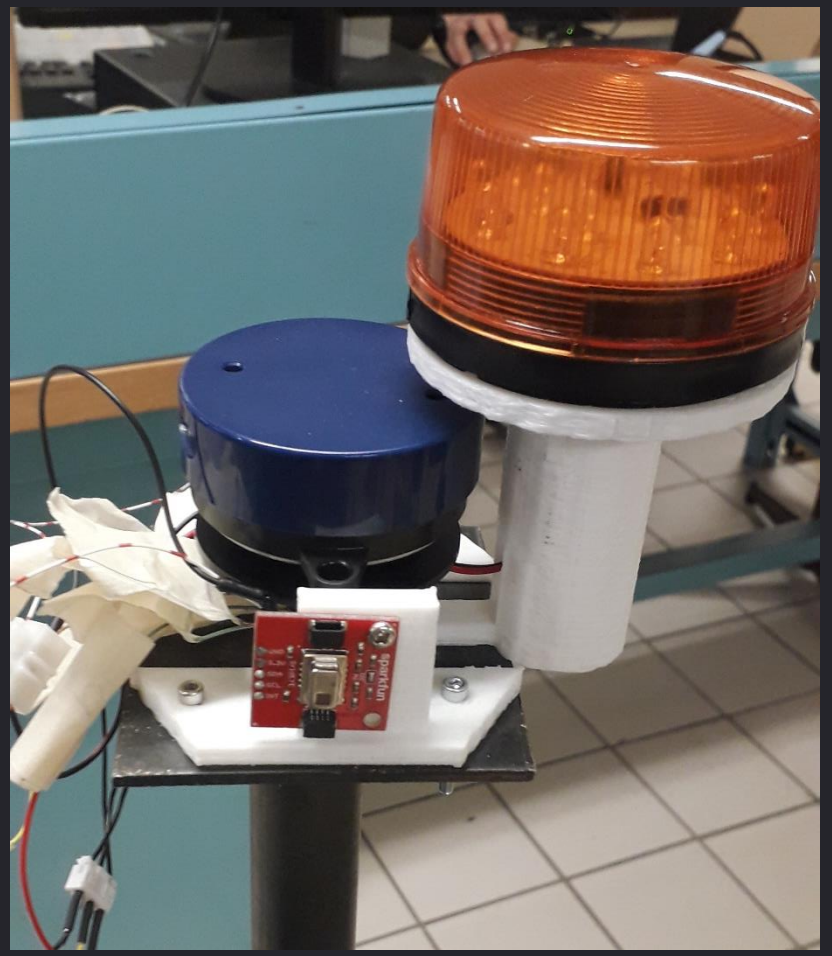
#06 Le support des Valises

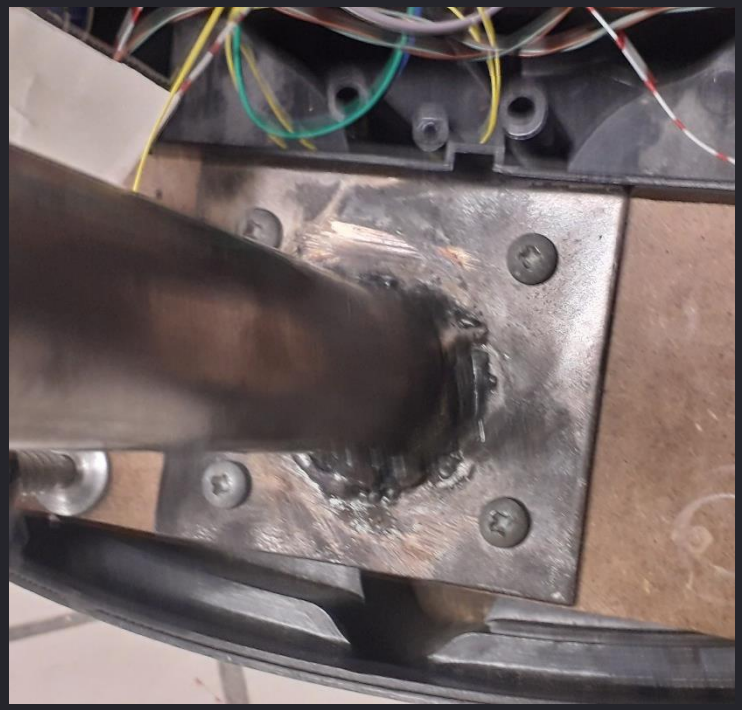




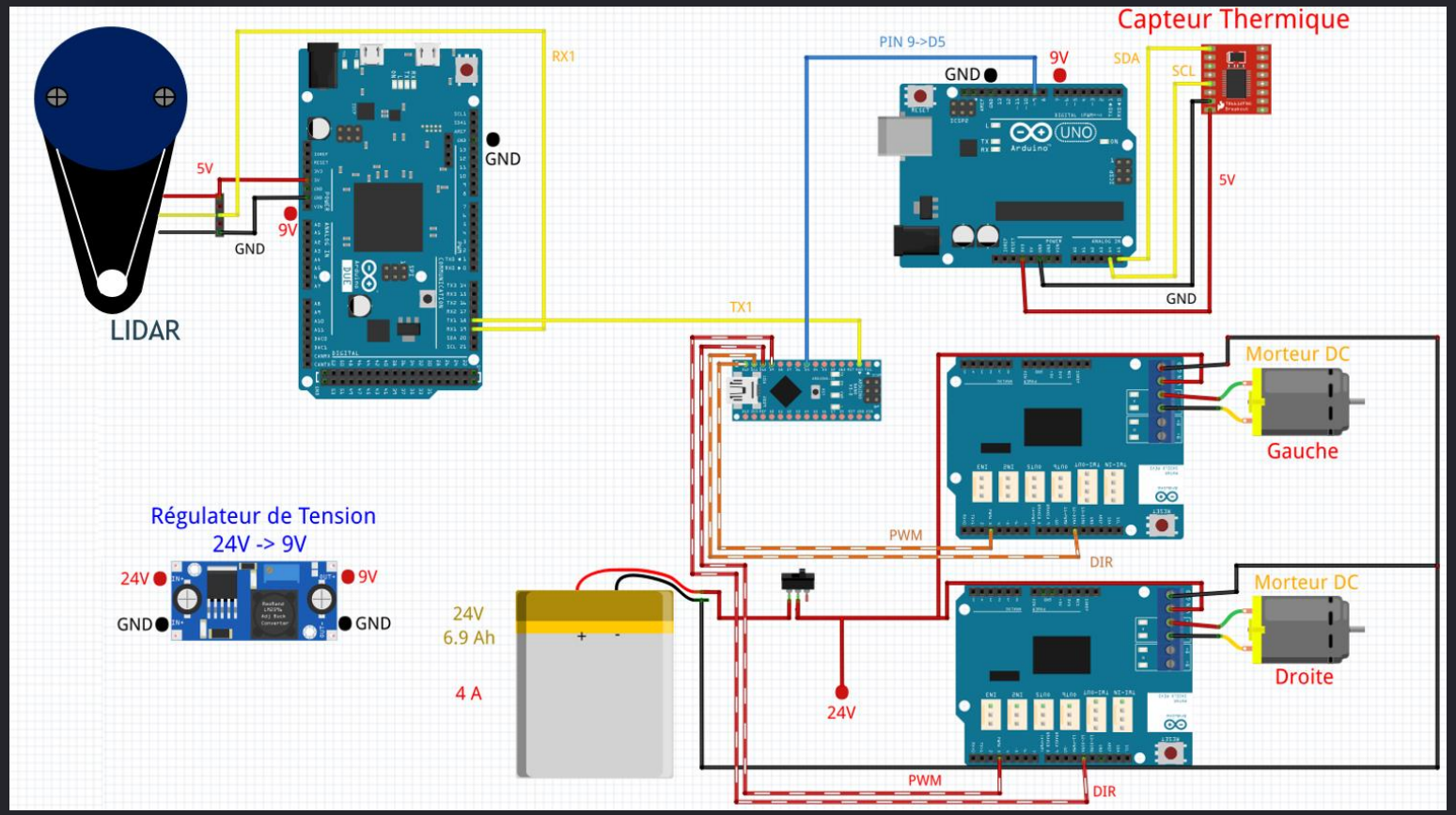


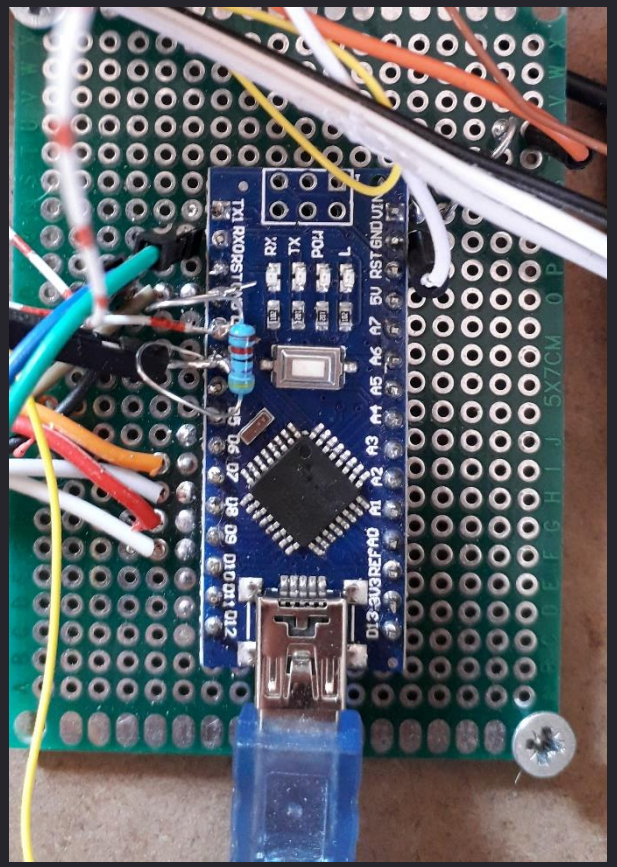
#06 Le support des Capteurs





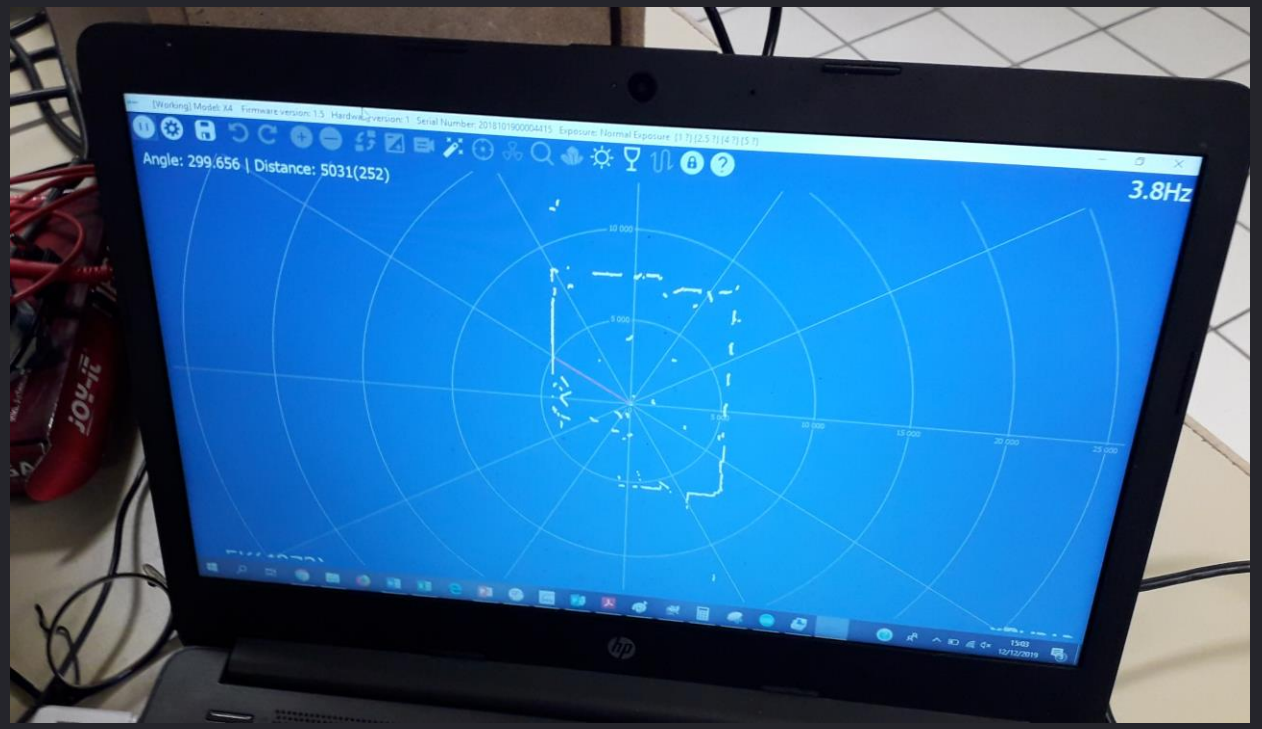
#06 Câblage électronique



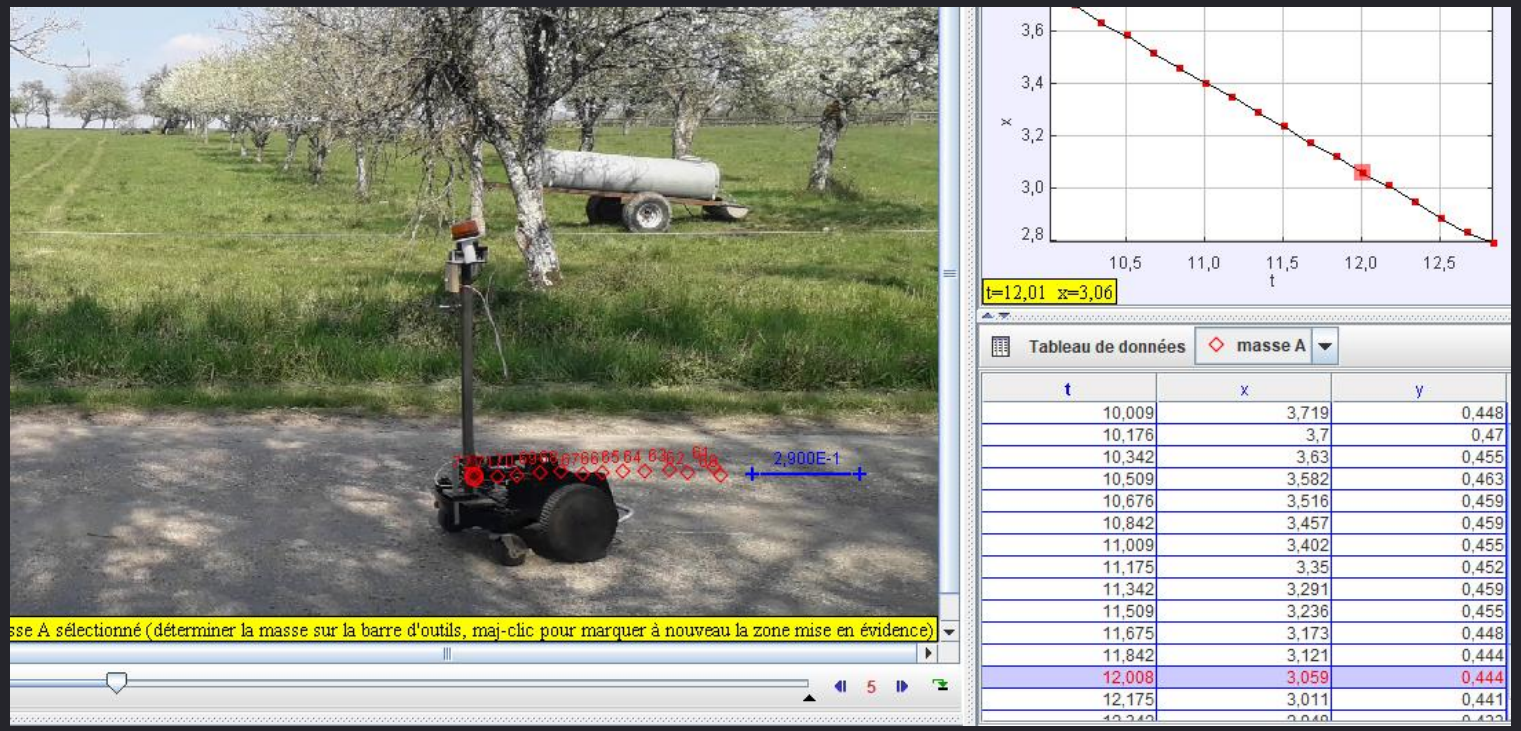


#07 : Difficultés et mesure d’equart

#07 LIDAR



#07 Limites de la vitesse du Robot

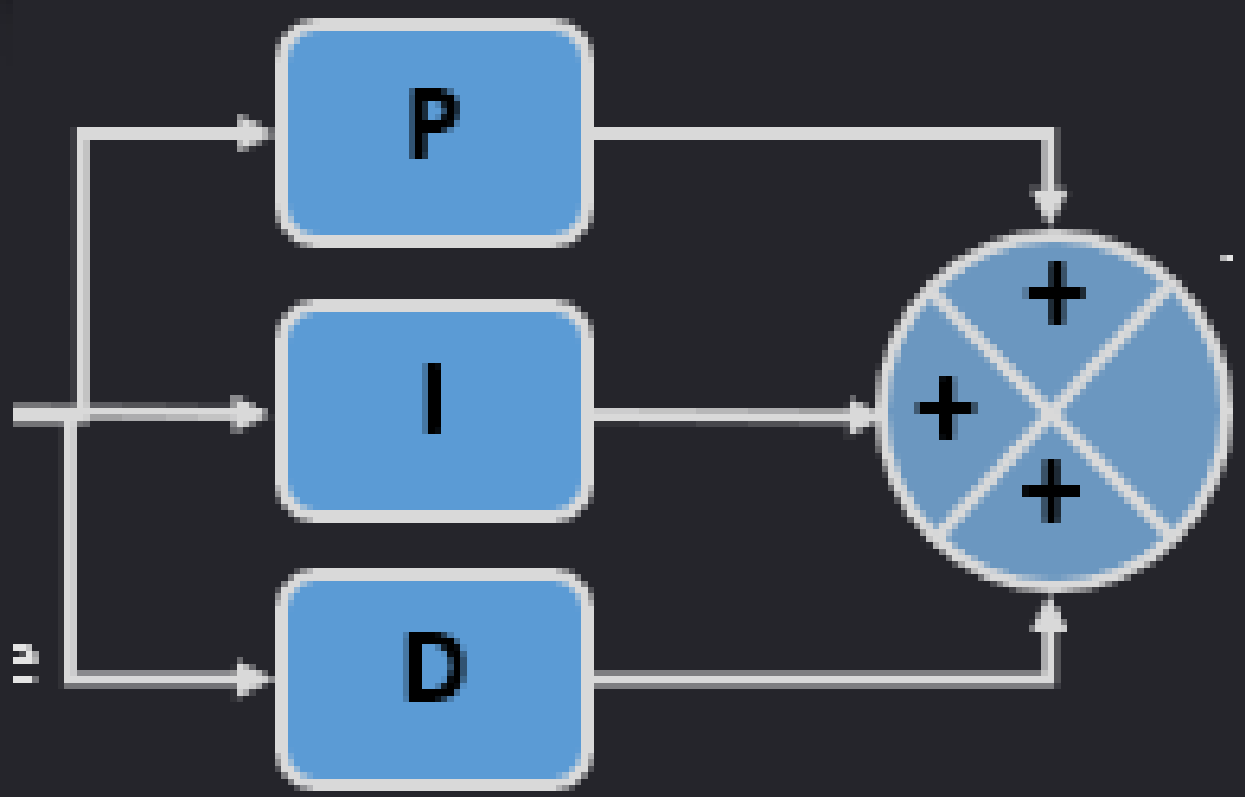


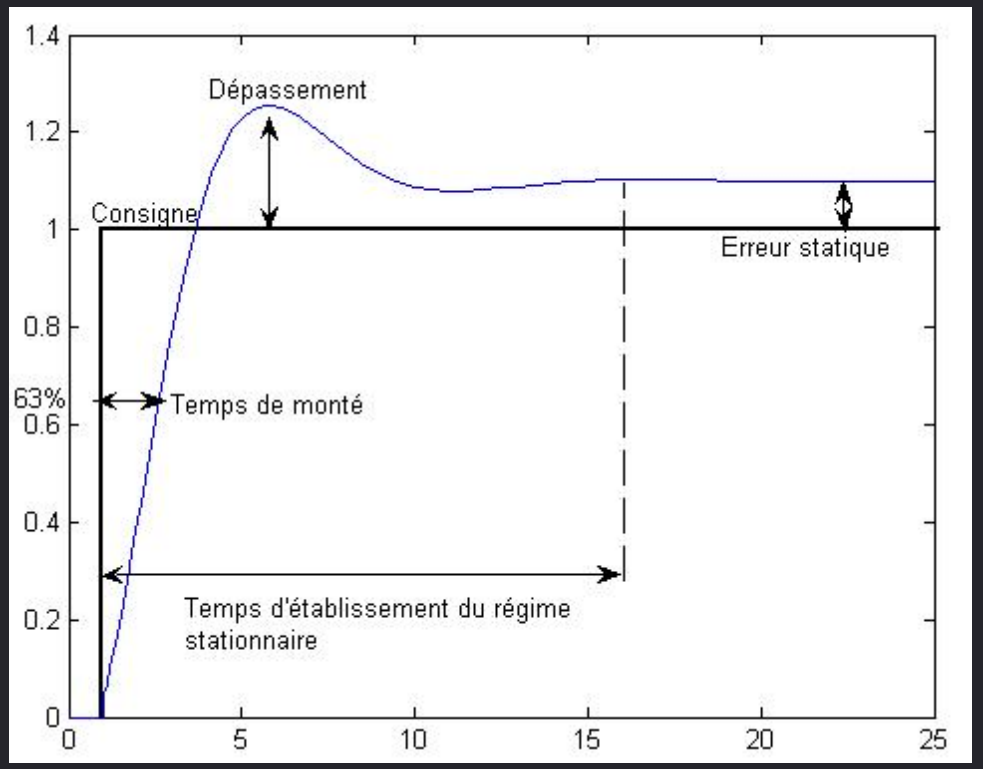
Vitesse Max = 0,4m/s < 2m/s

#07 Distance entre le Robot et la PMR

#07

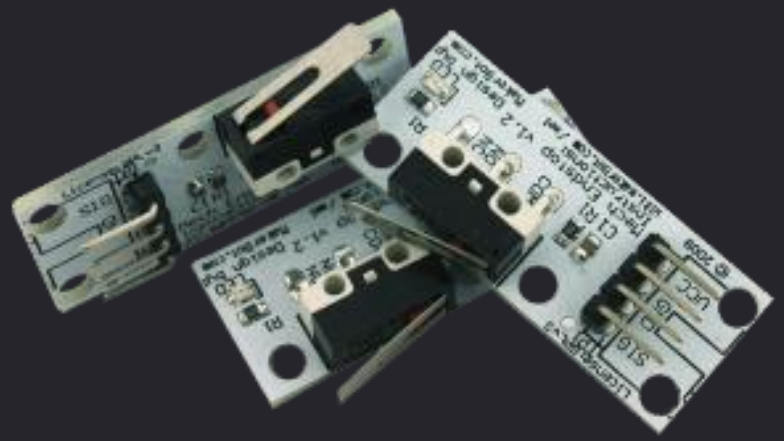
Définition des Coefficients PID



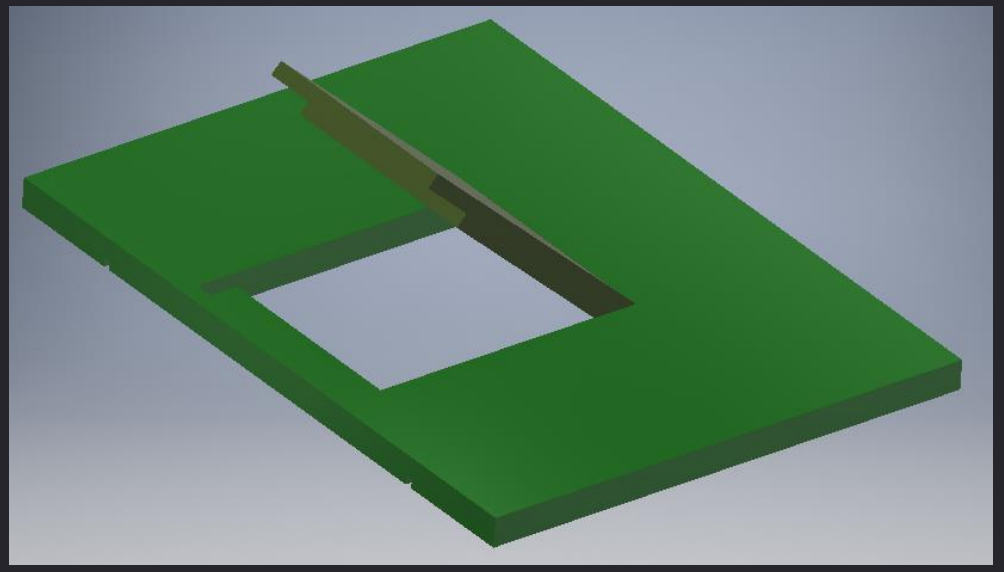


#07

Manque de Temps :



Capteur fin de course



Support des Valises

Merci de votre attention

Avez-vous des questions ?

#05

Code Arduino :

```
#define pwmmd 9   //vitesse moteur droit
#define pwmmg 10  //vitesse moteur gauche
#define dirmd 7   //sens moteur droit
#define dirmg 8   //sens moteur gauche

#define echo 3    //pin echo capteur USon
#define trig 2    //pin triger capteur USon

#define kpd 4    //coef proportionel PID distance
#define kid 0     //coef integration PID distance
#define kdd 2.5    //coef derivation PID distance

#define kpa 0.25    //coef proportionel PID angle
#define kia 0    //coef ingration PID angle
#define kda 0.07    //coef derivation PID angle

#define distance_cst 60 //distance constante entre robot et PMR


double erreurd, erreurd_precedente, action_distance, erreura, erreura_precedente, action_angle, action_droit, action_gauche, acgp, v;
double acdp = 60;
unsigned long temps_actuel,temps_precedent,temps;
int integrerd, deriverd, integrera, derivera;
double action[3];
double distance;
```

#05

Code Arduino

```
void calculePID(){                          //calcule des PID
  temps_actuel=millis();                              //mesure du temps
  temps=temps_actuel-temps_precedent;                 //diference du temps

  erreurd=distance-distance_cst;                      //erreur de distance
  integrerd += erreurd*temps;                         //integration de distance
  deriverd = (erreurd-erreurd_precedente)/temps;      //derivation de distance

  integrera += erreura*temps;                         //integration angle
  derivera = (erreura-erreura_precedente)/temps;      //derivation angle

  erreurd_precedente=erreurd;                         //sauvegarde de l'erreur de distance
  erreura_precedente=erreura;                         //sauvegarde de l'erreur d'angle
  temps_precedent=temps_actuel;                       //sauvegarde du temps

  action_distance=erreurd*kpd+integrerd*kid+deriverd*kdd; //aplication des coef. PID en distance
  action_angle=erreura*kpa+integrera*kia+derivera*kda;    //aplication des coef. PID en angle

}
```

#05

Code Arduino

```
void executionPID(){
  action_distance=abs(action_distance);   //valeur absolue de l'action de distance
  if(action_distance>200) action_distance=200;  //limite de l'action de distance à 200

  if(erreurd>0){                        //selection du sens avencer
    digitalWrite(dirmd, LOW);
    digitalWrite(dirmg, LOW);

    if(action_angle>50) action_angle=50;          //limite de l'action d'angle <50
    if(action_angle<-50) action_angle=-50;        //limite de l'action d'angle >-50

    action_droit=action_distance+action_angle;  //calcule action moteur droit
    action_gauche=action_distance-action_angle; //calcule action moteur gauche
  }

  if(erreurd<0){                        //selection du sens reculer
    digitalWrite(dirmd, HIGH);
    digitalWrite(dirmg, HIGH);

    if(action_angle>50) action_angle=50;          //limite de l'action d'angle <50
    if(action_angle<-50) action_angle=-50;        //limite de l'action d'angle >-50
    if(action_angle>action_distance) action_angle=action_distance;//limite de la vitesse de rotation à la distance entre la perssone et le robot
    if(action_angle<(-action_distance)) action_angle=(-action_distance);//limite de la vitesse de rotation à la distance entre la perssone et le robot

    action_droit=action_distance+action_angle;  //calcule action moteur droit
    action_gauche=action_distance-action_angle; //calcule action moteur gauche
```

#05

Code Arduino

```
    if (action_droit < 0){
      digitalWrite(dirmd, !digitalRead(dirmd));
      action_droit=abs(action_droit);
    }
    if (action_gauche < 0){
      digitalWrite(dirmg, !digitalRead(dirmd));
      action_gauche=abs(action_gauche);
    }
  }

void setup() {
 TCCR1B = TCCR1B & 0b11111000 | 0x01; //frequence PWM 32kHz

 pinMode(pwmmd,OUTPUT);
 pinMode(pwmmg,OUTPUT);      //setup
 pinMode(dirmd,OUTPUT);      //des
 pinMode(dirmg,OUTPUT);      //moteur
 pinMode(trig,OUTPUT);       //setup
 pinMode(echo,INPUT);        //Ultra-Son

 Serial.begin(115200);
 Serial3.begin(115200);

 delay(3000);
}
```

#05

Code Arduino

```
void loop() {
    while (!Serial3.available()){
      //Serial.print(".");
      }        //recuperation info LIDAR
    distance = Serial3.read();      //recuperation distance
    while(!Serial3.available()){}  //attente de reception de l'angle
    erreura=Serial3.read();          //recuperation angle
    erreura=180-erreura;
    v=abs(distance-acdp);
    if (v<20){
    calculePID();
    executionPID();
    analogWrite(pwmmd, action_droit);              //envoie de la comande moteur droit
    analogWrite(pwmmg, action_gauche);             //envoie de la comande moteur gauche

    Serial.print(action_gauche);
    Serial.print(" : ");
    Serial.println(action_droit);
    acdp=distance;
    }
}
```