Rapport de la séance 12

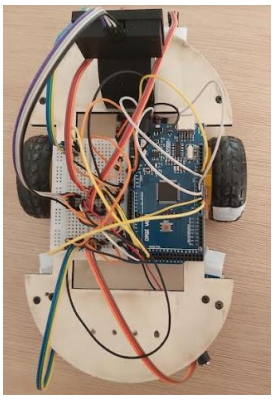
12 mars 2024

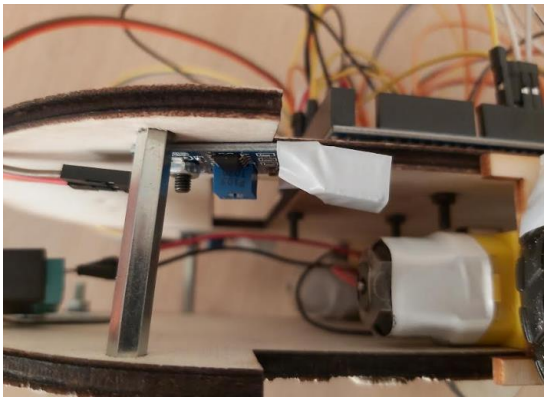
WARTSKI NARANJO Daniel

Robotique

Travail réalisé

Lors de cette séance, j'ai entamé l'installation de la carte Mega dans la base de mon robot, procédant au câblage nécessaire ainsi qu'au remplacement des broches pour passer de l'Arduino UNO à la carte Mega. J'ai également pu fixer les nouveaux supports pour les moteurs, ce qui les rend désormais beaucoup plus stables. En parallèle, j'ai intégré les capteurs infrarouges qui serviront à réaliser le PID de vitesse dans les roues du robot. Les images ci-dessous illustrent les différentes installations effectuées.





Par la suite, j'ai commencé à me familiariser avec l'utilisation des moteurs pas à pas destinés au mécanisme permettant de positionner les "plateaux" sur la table. J'ai ainsi étudié l'utilisation de la bibliothèque "Stepper". Après une phase de tests, j'ai constaté que la vitesse maximale de ces moteurs est de 10 tours par minute, tandis que la vitesse minimale est de 1 tour par minute. Vous trouverez ci-dessous le programme élaboré pour utiliser ces moteurs.

```ino
#include <Stepper.h>

#define IN1  11
#define IN2  10
#define IN3  9
#define IN4  8

double stepsPerRevolution = 2048;

Stepper myStepper(stepsPerRevolution, IN4, IN3, IN2, IN1);

void setup() {
  myStepper.setSpeed(10);
  Serial.begin(9600);
  }

void loop() {
  // 1 rotation counterclockwise:
  Serial.println("counterclockwise");
  myStepper.step(stepsPerRevolution);
  delay(1000);
 Serial.println("clockwise");
 myStepper.step(-stepsPerRevolution);
 delay(1000);
}
```

Enfin, j'ai poursuivi le développement du programme visant à implémenter le correcteur PID pour le robot. À cette fin, j'ai créé trois fonctions pour chaque moteur correspondant à un correcteur P, un correcteur PI et un correcteur PID. Il ne reste désormais plus qu'à effectuer des tests avec une impulsion échelon afin de déterminer les constantes kp, ki et kd. Les images suivantes présentent le programme du PID après les modifications effectuées.

```ino
#include <TimerOne.h>
#include <NewPing.h>

#define M1A 9
#define M1B 8
#define M2A 4
#define M2B 5
#define IR1 48
#define IR2 50
#define IR3 52
#define IRPIDR 3
#define IRPIDL 2
#define echo 53
#define trig 51

int distance;
int v=120;
String a="s";

int PWMD=0;
int PWMG=0;
unsigned int compteurD=0;
unsigned int compteurG=0;
unsigned long int deltatemps=20;
float const Mult=1/(deltatemps*0.001);
float RPMD;
float RPMG;

```

```ino
void setup() {
  // put your setup code here, to run once:
  Timer1.initialize(deltatemps*1000);
  Timer1.attachInterrupt(CalculD);
  Timer1.attachInterrupt(CalculG);
  Serial.begin(9600);
  pinMode(M1A,OUTPUT);
  pinMode(M2A,OUTPUT);
  pinMode(M1B,OUTPUT);
  pinMode(M2B,OUTPUT);
  pinMode(IR1,INPUT);
  pinMode(IR2,INPUT);
  pinMode(IR3,INPUT);
  pinMode(echo,INPUT);
  pinMode(trig,OUTPUT);
  attachInterrupt(5,changeD,FALLING);
  attachInterrupt(4,changeG,FALLING);
  stop();
  delay(1000);
}

```

```ino
//Calcul de la vitesse des moteur en tr/s
void CalculD(){
  RPMD=Mult*compteurD;
  Serial.println(millis());
  Serial.print(" ");
  Serial.println(RPMD,3);
  compteurD=0;
}

void CalculG(){
  RPMG=Mult*compteurD;
  Serial.println(millis());
  Serial.print(" ");
  Serial.println(RPMG,3);
  compteurG=0;
}

//******Interruptions******
void changeD(){
  compteurD++;
}

void changeG(){
  compteurG++;
}
```

```arduino
void moteurD(int v, byte sens){
  if (sens==0){
    analogWrite(M2A,0);
    analogWrite(M2B,v);
  }
  else{
    analogWrite(M2B,0);
    analogWrite(M2A,v);
  }
}

//******Fonction pour le moteur gauche******
void moteurG(int v, byte sens){
  if (sens==0){
    analogWrite(M1A,0);
    analogWrite(M1B,v);
  }
  else{
    analogWrite(M1B,0);
    analogWrite(M1A,v);
  }
}
```

```arduino
float erreurD=0.0; //difference entre la consigne et la mesure
const int  kpD=100; //coefficient de proportionnalité
void correcteur_P_D(int consD, byte sens){
  erreurD=consD-RPMD; // calcul de l'erreur
  PWMD=kpD*erreurD; //determination de la valeur du PWM
  if (PWMD<0){
    PWMD=0; //bornage du PWM dans l'intervalle [0,255]
  }
  else if (PWMD>255){
    PWMD=255;
  }
  if(sens==0){
    moteurD(PWMD,0);
  }
  else{
    moteurD(PWMD,sens); //application du nouveau signal PWM
  }
}

float erreurG=0.0; //difference entre la consigne et la mesure
const int  kpG=100; //coefficient de proportionnalité
void correcteur_P_G(int consG, byte sens){
  erreurG=consG-RPMG; // calcul de l'erreur
  PWMG=kpG*erreurG; //determination de la valeur du PWM
  if (PWMG<0){
    PWMG=0; //bornage du PWM dans l'intervalle [0,255]
  }
  else if (PWMG>255){
    PWMG=255;
  }
  moteurG(PWMG,sens); //application du nouveau signal PWM
}
```

```arduino
const int kiD=0.2; //coefficient de l'intégrale
float somme_erreurD=0.0; //somme des erreurs
void correcteur_PI_D(int consD, byte sens){
  erreurD=consD-RPMD; // calcul de l'erreur
  somme_erreurD=somme_erreurD+erreurD;
  PWMD=kpD*erreurD+kiD*somme_erreurD; //determination de la valeur du PWM
  if (PWMD<0){
    PWMD=0; //bornage du PWM dans l'intervalle [0,255]
  }
  else if (PWMD>255){
    PWMD=255;
  }
  moteurD(PWMD,sens); //application du nouveau signal PWM
}

const int kiG=0.2; //coefficient de l'intégrale
float somme_erreurG=0.0; //somme des erreurs
void correcteur_PI_G(int consG, byte sens){
  erreurG=consG-RPMG; // calcul de l'erreur
  somme_erreurG=somme_erreurG+erreurG;
  PWMG=kpG*erreurG+kiG*somme_erreurG; //determination de la valeur du PWM
  if (PWMG<0){
    PWMG=0; //bornage du PWM dans l'intervalle [0,255]
  }
  else if (PWMG>255){
    PWMG=255;
  }
  moteurG(PWMG,sens); //application du nouveau signal PWM
}
```

```arduino
float erreurD_avant=0.0; //erreur précédente
float delta_erreurD=0.0; //variation de l'erreur
void correcteur_PID_D(int consD, byte sens){
  erreurD=consD-RPMD; // calcul de l'erreur
  somme_erreurD=somme_erreurD+erreurD;
  delta_erreurD=erreurD-erreurD_avant;
  erreurD_avant=erreurD;
  PWMD=kpD*erreurD+(kiD*somme_erreurD)+(kdD*delta_erreurD); //determination de la valeur du PWM
  if (PWMD<0){
    PWMD=0; //bornage du PWM dans l'intervalle [0,255]
  }
  else if (PWMD>255){
    PWMD=255;
  }
  moteurD(PWMD,sens); //application du nouveau signal PWM
}
const float kdG=10; //coefficient de la dérivée
float erreurG_avant=0.0; //erreur précédente
float delta_erreurG=0.0; //variation de l'erreur
void correcteur_PID_G(int consG, byte sens){
  erreurG=consG-RPMG; // calcul de l'erreur
  somme_erreurG=somme_erreurG+erreurG;
  delta_erreurG=erreurG-erreurG_avant;
  erreurG_avant=erreurG;
  PWMG=kpG*erreurG+(kiG*somme_erreurG)+(kdG*delta_erreurG); //determination de la valeur du PWM
  if (PWMG<0){
    PWMG=0; //bornage du PWM dans l'intervalle [0,255]
  }
  else if (PWMG>255){
    PWMG=255;
  }
  moteurG(PWMG,sens); //application du nouveau signal PWM
}
```

Objectifs pour la prochaine séance

- Poursuivre la mise en place du programme pour intégrer le correcteur PID.
- Continuer le développement du programme du robot afin qu'il réponde aux exigences spécifiées.
- Imprimer à nouveau la boîte pour les piles et procéder à son installation dans le robot.
- Réfléchir à la manière d'implémenter un mécanisme permettant au robot de déposer les "plateaux" sur la table.