Haute Ecole Libre de Bruxelles - IP
Catégorie d'enseignement supérieur technique
75, Avenue Victor Rousseau – 1190 Bruxelles

Haute École Francisco Ferrer
Catégorie d'enseignement supérieur technique
Rue de le Fontaine, 4 – 1000 Bruxelles

Pôle universitaire européen de Bruxelles Wallonie

Table tournante motorisée

Maître de stage et de TFE : Vandermeersch Eric
Maître de stage en entreprise : Ercek Rudy

Année Académique 2017-2018 Ben Ismail Selim Marc
Mémoire présenté en vue de l'obtention du titre de bachelier en Électronique

Remerciements

Pour ces quinze semaines de stage au LISA-image de l'ULB, qui ont été riches en expériences, en apprentissages et en rencontres, je souhaiterais adresser un remerciement à :

Rudy Ercek et Olivier Debeir qui ont été mes maîtres de stage au sein du laboratoire et qui m'ont guidé et encadré tout au long de ce stage.

Ainsi qu'à Eric Vandermeersch et Greta Van Vinckeroy qui, non seulement, m'ont suivi administrativement et ont veillés au bon déroulement du stage, mais se sont aussi montrés présent et présente lorsque j'en avais besoin.

Sans oublier, Arlette Grave et Marc Walravens qui se sont occupés de toutes les tâches administratives indispensables au stage.

Je voudrais remercier aussi Stephan Hahn et Yves-Rémi Van Eycke avec qui je partageais le bureau pour leur aide régulière et le soutien qu'ils m'ont apporté.

D'une manière plus générale je tiens à remercier tout le département pour leur accueil chaleureux au sein du laboratoire, et le corps professoral de la Haute école Francisco Ferrer ainsi que de la Haute École Ilya Prigogine pour les connaissances et compétences qu'ils m'ont permis d'acquérir tout au long des ces années de bachelier.

J'aimerais adresser un dernier remerciement à monsieur Philippe Gosselain pour sa relecture.

A tous, merci !

2

Sommaire

Introduction

Le projet élaboré et développé, dans le cadre de mon stage de troisième année de bachelier en électronique appliquée, a été réalisé au LISA-image de l'école polytechnique de Bruxelles. Il consiste en une table tournante motorisée qui a pour but d'automatiser partiellement l'acquisition d'images photographiques d'un objet afin d'en reconstituer une image 3D par photogrammétrie.

#Table tournante #Photogrammétrie #ArduinoNano #Moteur pas-à-pas #LISA-image

Cahier des charges

Réalisation de la table tournante [1]

Afin de partiellement automatiser la photogrammétrie d'objets (i.e. reconstruction 3D à partir de photos), une table tournante, permettant la rotation d'objets, doit être construite et répondre aux spécifications suivantes :

Interface :

- La table tournante est configurée et pilotée par une télécommande infrarouge et par des boutons poussoirs.
- Un affichage LCD est présent pour visualiser les configurations effectuées.

Conception Mécanique:

- Le plateau tournant de la table doit être en MDF noir teinté dans la masse afin de limiter au maximum les problèmes de réflexions/ombres.
- Elle doit pouvoir supporter une charge maximale de 50 kg.
- Le plateau tournant doit avoir un diamètre de 50 cm minimum.
- La surface du plateau doit être plane et lisse.
- Aucun élément statique ne peut apparaitre sur les photos.

Fonctionnalités :

- Elle doit pouvoir tourner selon un angle prédéfini, déclencher la prise de photos à l'aide d'un émetteur infrarouge (led Ir), attendre un certain temps pour permettre la prise de la photo (qui correspond au temps maximum d'exposition), et ainsi de suite jusqu'à avoir fait une rotation complète. Chaque pas de rotation est effectué automatiquement ou manuellement à l'aide de la télécommande.
- Elle doit posséder une fonctionnalité de remise à zéro.
- Les paramètres de configuration doivent rester en mémoire malgré la mise hors tension de la table.
- Le déplacement angulaire doit être précis.
- La table doit accélérer et décélérer de façon à ne pas faire bouger l'objet posé sur elle, même en cas d'instabilité de celui-ci.
- La table doit pouvoir être repositionnée à la main ou via la télécommande.

[1] Le cahier des charges de la table tournante, ainsi que du déclencheur sans fil, a été écrit par monsieur Ercek Rudy, ingénieur système au LISA-image de l'ULB et maitre de stage

Réalisation du déclencheur photo sans fil avec récepteur infrarouge

Certains appareils photos, de gamme professionnelle (ex : Nikon D810), ne disposent pas d'un capteur infrarouge pour déclencher la prise de photos à distance mais ils disposent d'un connecteur (ex : 10 broches pour les Nikons, jack 2.5 mm pour d'autres modèles, …) permettant de brancher un déclencheur mécanique à deux niveaux (focus puis photo). L'objectif de ce projet sera donc de réaliser un déclencheur Ir pour appareil photo Nikon envoyé soit par une télécommande Ir, soit directement par la led Ir de la table tournante.

Les caractéristiques du déclencheur Infrarouge seront les suivantes :

- Le module doit se connecter à l'appareil photo via le connecteur 10 broches.
- Il doit être le plus petit possible : un attiny85, avec horloge interne à 1,4 ou 8Mhz sera utilisé si possible (*pdip* 8 pattes), sinon un arduino mini pro à 8Mhz.
- Il doit pouvoir recevoir les signaux infrarouges et en émettre pour confirmer le déclenchement/la réception du signal.
- Il doit déclencher la prise de photos par l'appareil photo avec mise au point et déclenchement.
- Il doit être muni d'un système d'enregistrement de code infrarouge à la volée, c'est à dire, un bouton qu'on maintiendrait appuyé pour enregistrer un code IR dans l'EEPROM du microcontrôleur et qui servira au déclenchement de l'appareil photo.
- Il doit être alimenté par une batterie rechargeable Li-ion de 3,7V et directement rechargeable.
- Il doit disposer d'un voyant pour le niveau de batterie.
- Il doit avoir également un voyant signalant le déclenchement ou la réception du signal infrarouge de déclenchement.

Réalisation du déclencheur photo avec fil

Dans le cas où la réalisation du déclencheur sans fil ne pourrait être possible ou si celui-ci ne conviendrait pas, un déclencheur connecté par fil peut être réalisé. Le déclencheur relierait alors directement le circuit à l'appareil photo de type Nikon D810 en utilisant le connecteur 10 broches.

Les caractéristiques du déclencheur filaire seraient les suivantes:

- Le module doit se connecter à l'appareil photo via le connecteur 10 broches.
- Le module possédera un interrupteur pour permettre un déclenchement manuel
- Le câble de connexion sera composé en deux parties: d'une part le déclencheur avec le bouton et le connecteur 10 broches. Et de l'autre, un câble avec une prise jack 2,5mm qui se connectera à la partie précédente.
- Les connexions entre les deux parties devront être faites selon des normes afin que l'ensemble puisse servir d'adaptateur.

Description du cadre scientifique et technique

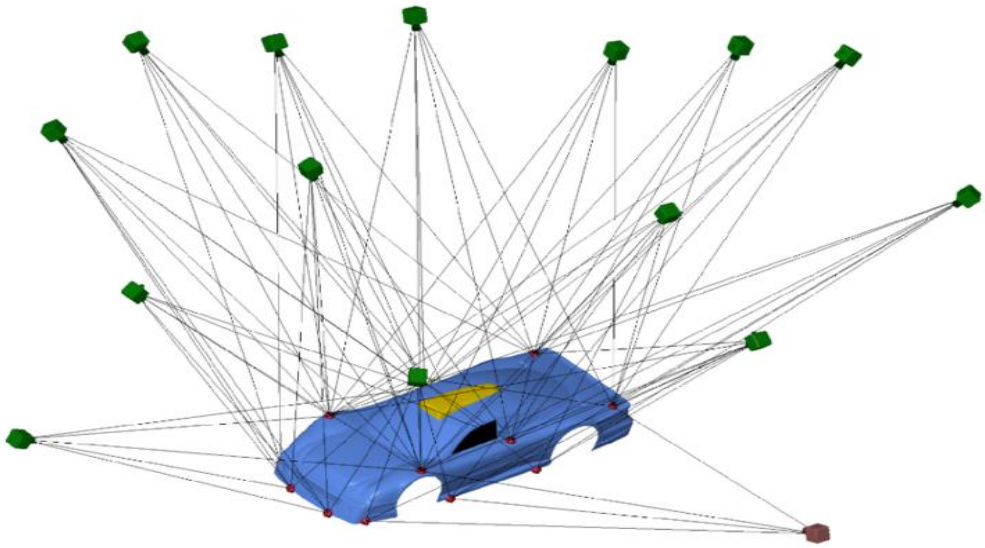
Le projet de la table tournante motorisée a été réalisé au LISA-image de l'école polytechnique de l'ULB par moi-même, sous la supervision de l'ingénieur Ercek Rudy ainsi que du professeur Debeir Olivier. Le projet était destiné au LISA-synthesis pour leurs travaux de reconstitution d'images 3D par photogrammétrie; le procédé prenant beaucoup de temps, l'intérêt du projet était de partiellement l'automatiser. De nombreuses tables tournantes existent déjà, mais peu disposent de toute les caractéristiques ou fonctionnalités souhaitées (trop petites, charge maximale trop légère, mauvaise coloris, manuelles, etc ...) et toutes sont très onéreuses.

Description des principes mis en œuvres

Photogrammétrie

Originellement, la technique de la photogrammétrie à été inventé par un officier de l'armé française: Aimé Laussedat, qui voulait utiliser les photographies de paysages pour mesurer le terrain.

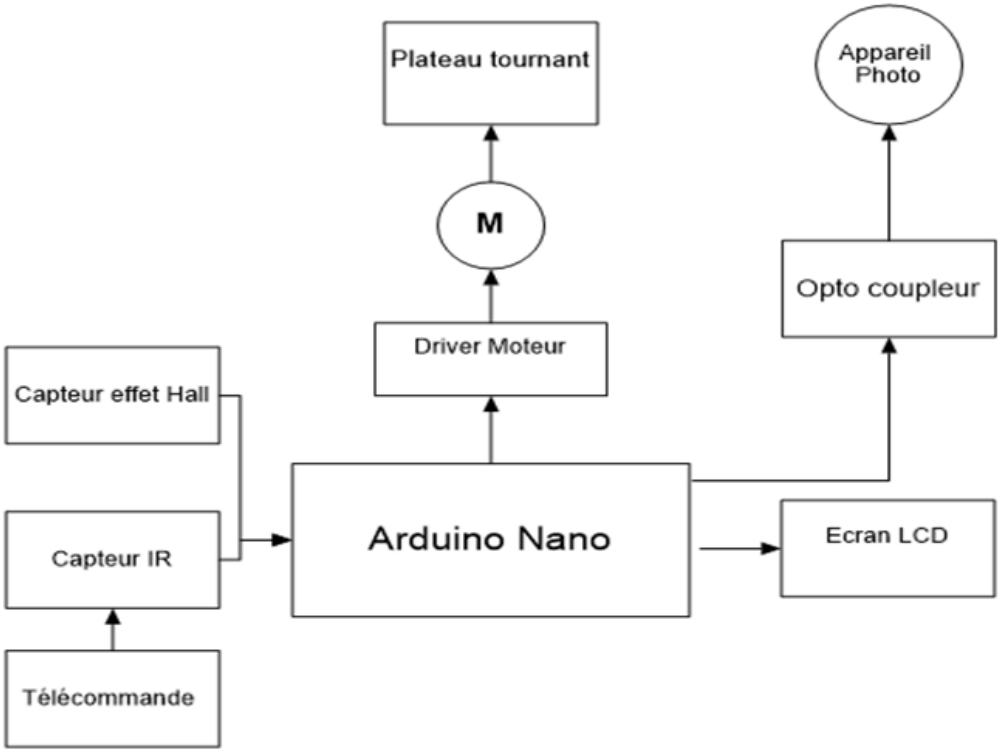
La photogrammétrie est une technique de modélisation d'images 3D basée sur le principe de la vision stéréoscopique humaine. Elle consiste à prendre des photos d'une scène sous différents angles et utiliser la parallaxe afin de calculer des points de corrélation entre les différents clichés photographiques.



(Image provenant du site : http://www.cut-messel.de)

Description du matériel

Schéma bloc



Le cœur du projet est construit autour d'un Arduino Nano.

Un capteur infrarouge connecté en entrée permet la réception des signaux émis par la télécommande pour le contrôle de la table tournante. Et un capteur à effet hall, lui aussi connecté en entrée, est utilisé pour repositionner le plateau de la table à un zéro de référence.

En sortie, tout d'abord, les signaux de commande qui passeront par un driver avant d'atteindre le moteur.

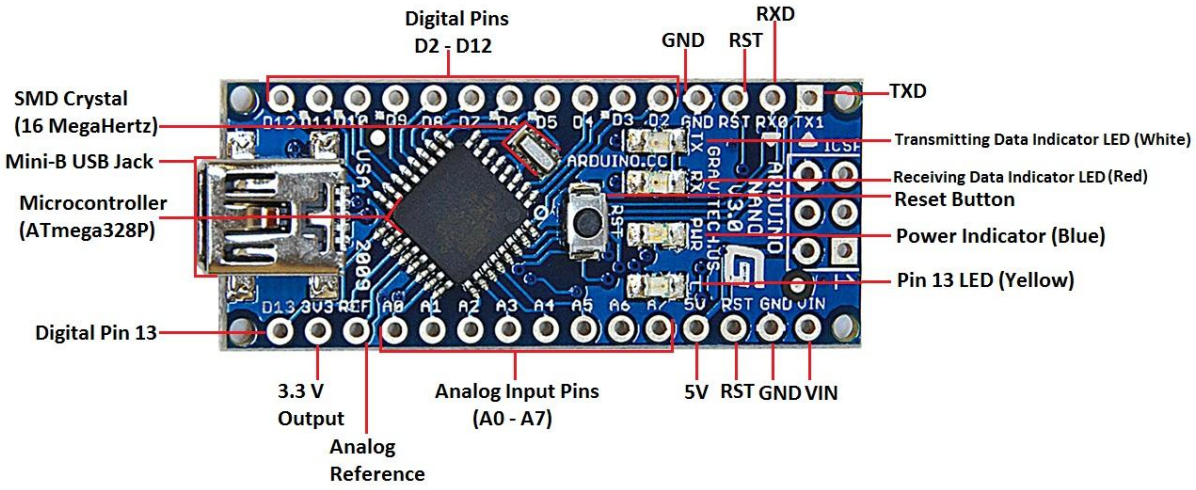
Ensuite deux opto-coupleurs qui servent au déclenchement de l'appareil photo tout en protégeant celui-ci d'éventuelles surtensions provenant du circuit.

Et finalement, on retrouve l'écran LCD pour l'affichage en temps réel des informations en fonction de l'état[2] du système

Arduino Nano

[2] cf : Machine d'état, expliquée dans la partie description modulaire

Le projet est développé sur un arduino nano, une carte à microcontrôleur sur laquelle est monté un *ATmega328p*.



(*Image provenant du site: http://www.circuitstoday.com*)

Caractéristiques techniques:

- Architecture : AVR
- Fréquence d'horloge : 16 MHz
- Tension nominale : 5V
- Mémoire flash : 32 KB des quelles 2 KB sont utilisé pour le démarrage.
- Mémoire EEPROM : 1 KB
- I/O : 22 digitales, dont 6 PWM et 8 analogiques
- Courant DC par I/O : 40 mA
- Consommation : 19 mA

L'arduino Nano a été choisi parce qu'il sied bien au prototypage: on peut facilement l'utiliser sur une platine d'essais et modifier les entrées/sorties. De plus, Arduino rassemble une grande communauté développant des outils et des librairies compatibles. Et dernièrement, pour son prix peux cher.

Moteur Pas-à-Pas

Le moteur utilisé est un *4STH38-100*: c'est un moteur pas-à-pas bipolaire sur lequel est fixé un réducteur planétaire de 1:100.

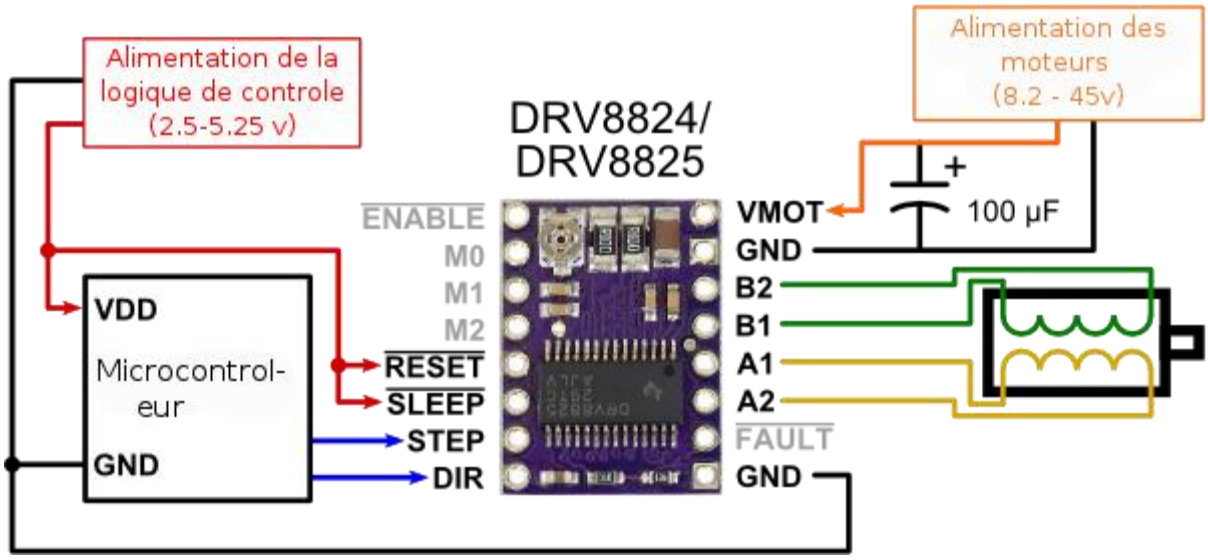
(*Image provenant du site internet : https://www.gotronic.fr*)

Caractéristiques techniques:

- Angle par pas : 1,8°
- Tension : 2,8 V (12 Vcc conseillé)
- Courant par phase : 1,7 A
- Résistance par phase : 1,7 ohm
- Couple de maintien : 48 kg.cm
- Charge radiale maximale : 98,1 N
- Connexion: 4 Fils

Driver

Le *DRV8825* qui est utilisé, est un driver développé par *Texas instruments* permettant un pilotage simplifié des moteurs pas-à-pas. Il dispose d'un limiteur de courant réglable ainsi qu'une protection et d'une résolution de 6 micros pas



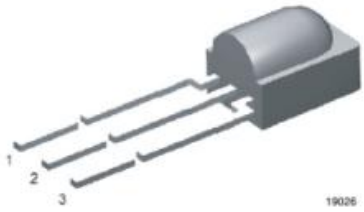
(Image provenant du site : wiki.mchobby.be)

Caractéristiques techniques:

- Tension d'entrée de la partie logique : 2,5 V à 5,25 V
- Tension d'entrée de la partie moteur : 8,2 V à 45 V
- Courant de sortie par phases: 2,2 A
- Protection contre les surtensions
- Protection contre la surchauffe

Capteur Infrarouge

Le capteur infrarouge qui sert à capteur les signaux de la télécommande est un *TSOP382*. Il est composé d'un photo-détecteur ainsi que d'un préamplificateur.



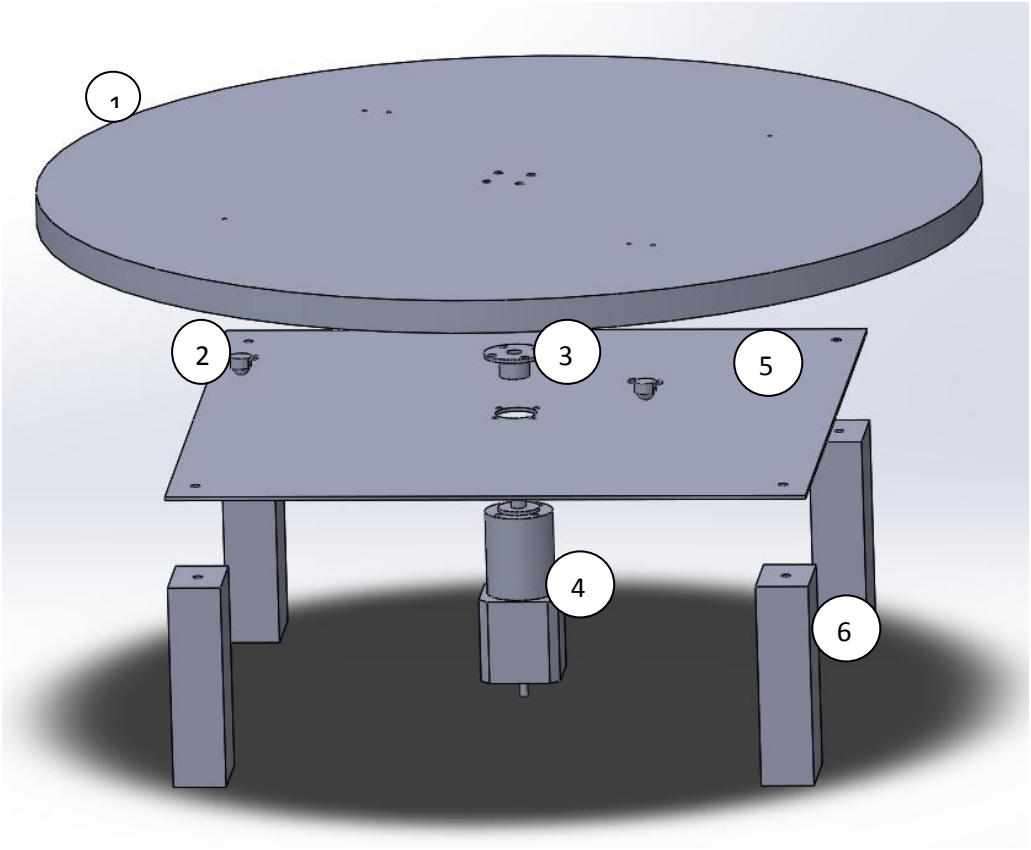
MECHANICAL DATA
Pinning for TSOP382.., TSOP384..:
1 = OUT, 2 = GND, 3 = V_S

(*Image provenant du site: https://andicelabs.com*)

Caractéristiques techniques:

- Tension d'alimentation : 2,5 V à 5,5 V
- Fréquence de la porteuse : 30 KHz
- Tolérance améliorée à la lumière ambiante
- Insensible aux ondulations de la tension d'alimentation
- Tension de sortie : -0,3 V à (Tension d'alimentation +0,3 V)
- Courant de sortie : 5 mA

Construction mécanique



(*Modélisation 3D réalisée sur le logiciel SolidWorks*)

La structure mécanique de la table est composée de :

1. Un plateau de 50 cm de diamètre et 18 mm d'épaisseur en MDF noir
2. Quatre billes de roulement
3. Une bride de serrage pour l'axe du moteur[3]
4. Un moteur pas-à-pas avec son réducteur planétaire
5. Une plaque d'aluminium de 3 mm d'épaisseur
6. Quatre pieds ayant une section de 3cm / 4cm et une hauteur de 12 cm.

A cela, s'ajoute[4] un carré de MDF noir de 18 mm d'épaisseur fixé au plateau circulaire et sur lequel sont fixées les billes de roulement. Ainsi qu'un autre carré de MDF 18mm venant rigidifier la plaque d'aluminium.

[3] Traduction approximative. Le terme exact anglais est "motor shaft coupling".

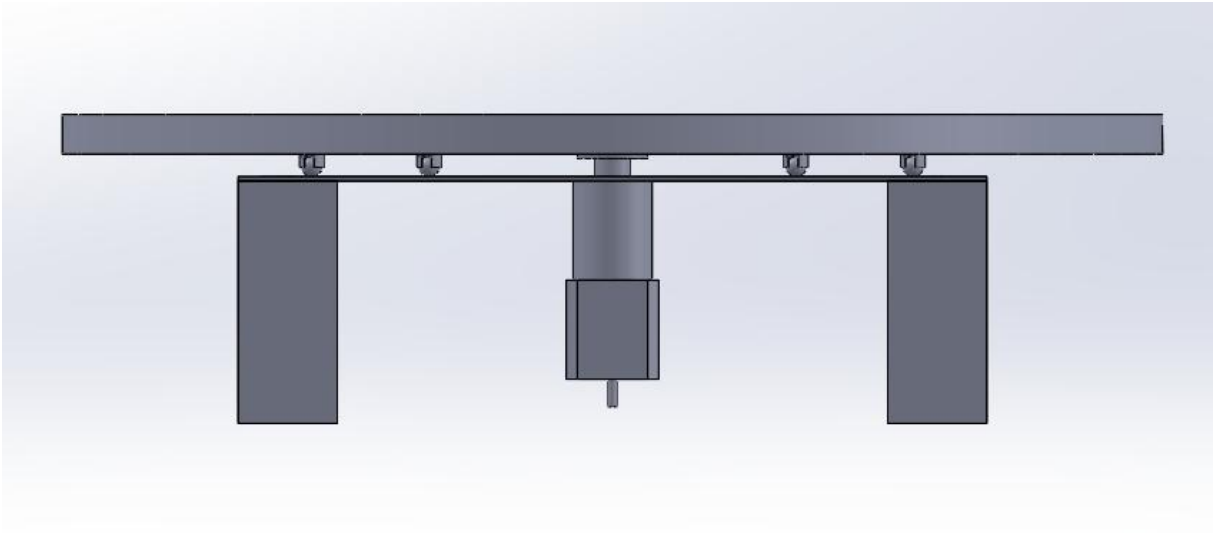
Le plateau de la table est en MDF noir teinté dans la masse. Le MDF est un bois reconstitué à partir de fibre de bois et d'un liant synthétique; il en résulte un produit d'une grande régularité. La couleur noire permet de simplifier le travaille numérique qui suivra. Quant à la teinture dans la masse assure que la couleur noire restera malgré les coups et les griffures. L'épaisseur de 18 mm été choisi de sorte que le plateau reste bien droit et plane malgré son large diamètre.

Sous le plateau est fixé un carré de MDF provenant du même panneau auquel sont fixées les billes de roulements. Cette pièce est là, car initialement, un étau autocentré devait être placé dans la table et arriver à fleur avec le plateau[5].

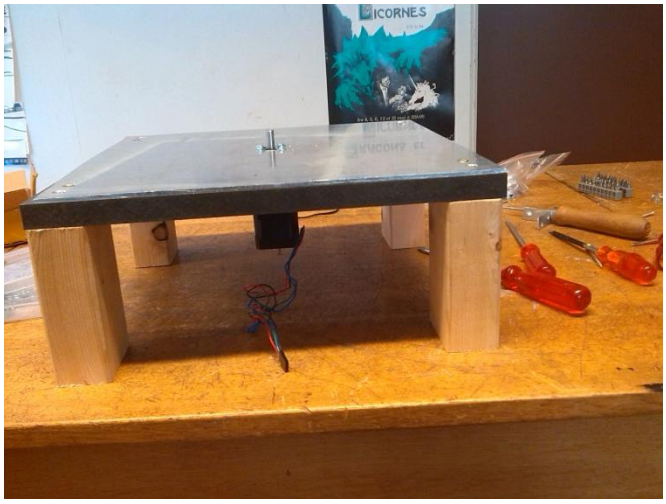
Les billes de roulement permettent de stabiliser le plateau avec un frottement minime.

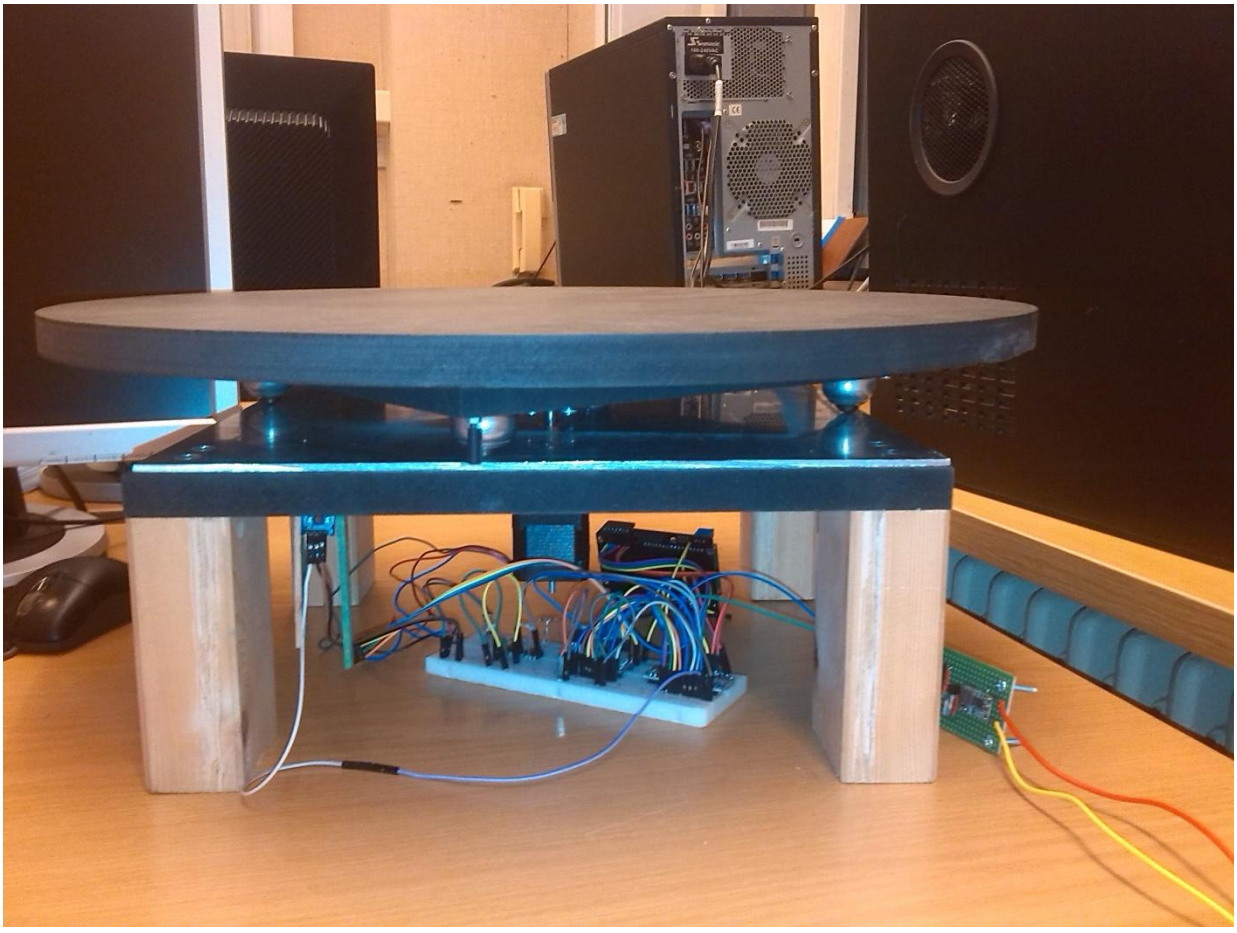
La bride, elle, est serrée sur le rotor du moteur et vissée au plateau. Elle sera donc la liaison entre le plateau et le moteur.

La plaque d'aluminium permet un bon roulement des billes, et empêche que celles-ci ne creusent un sillon dans la plaque de bois sous-jacente.



[4] La modélisation 3D étant antérieure à certains changements, certains éléments de renfort n'apparaissent pas dessus.
[5] Cet élément a été retiré du cahier des charges


LICORNES




FUJIFILM

Description du logiciel

Description fonctionnelle

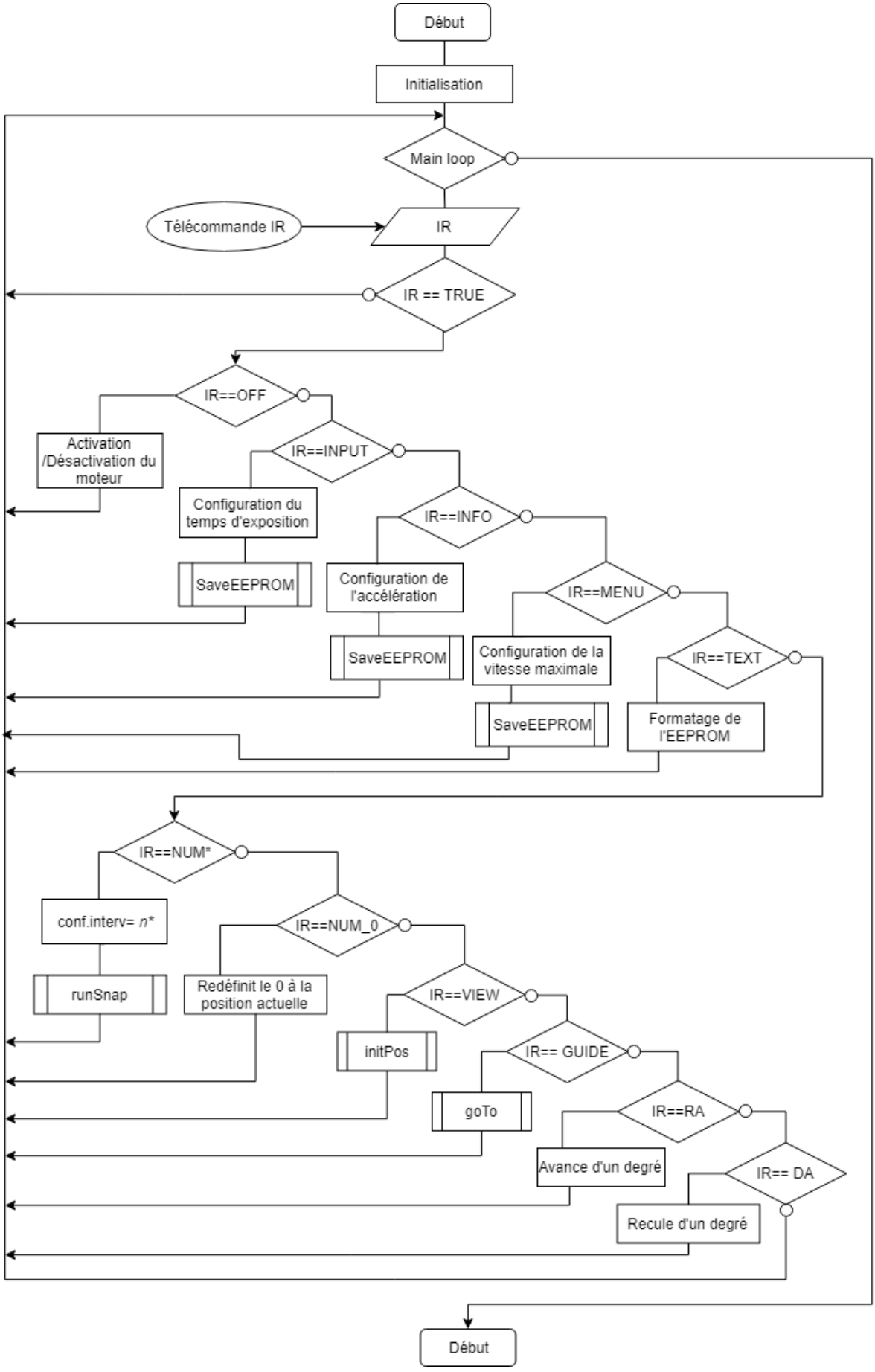
L'objectif du logiciel est d'automatiser partiellement le processus de photogrammétrie. Le programme tourne donc principalement autour d'une fonction qui déplace le plateau à certains angles précis pour ensuite déclencher la prise de photo après un délai déterminé et du paramétrage de cette fonction.

Le programme dispose d'une série de configurations préétablies pour cette fonction afin d'être simple et rapide à utiliser. En outre, il offre à l'utilisateur la possibilité d'ajuster précisément la position du plateau de la table et de l'utiliser entièrement via une télécommande infrarouge.

La boucle principale va consister en un menu construit autour de la valeur du signal infrarouge décodé. Selon cette valeur, trois types d'actions vont être effectués :

- La modification des paramètres ou leur réinitialisation
- L'utilisation de la fonction *runSnap* avec différentes pré-configurations d'angles
- Le déplacement du plateau de la table

Ordinogramme du programme :



Description modulaire

Contrôle du moteur:

Accelstepper

La gestion du moteur pas-à-pas est principalement faite autour de la librairie Arduino *Accelstepper*. Arduino possède déjà sa propre librairie *Stepper* pour le contrôle de ces moteurs. Mais celle-ci ne convientque pour des utilisations très basiques du moteur. *Accelstepper* est plus performant et apporte nombres de possibilités supplémentaires.

Dans le cadre du projet, la librairie va se montrer particulièrement intéressante pour sa gestion de l'accélération des moteur pas-à-pas.

En effet, dans le processus de photogrammétrie, il est primordial que l'objet ne bouge pas entre les différentes prises de photos. Le démarrage et l'arrêt du plateau doivent donc se faire de manière très douce et sans à-coups.

MyStepper

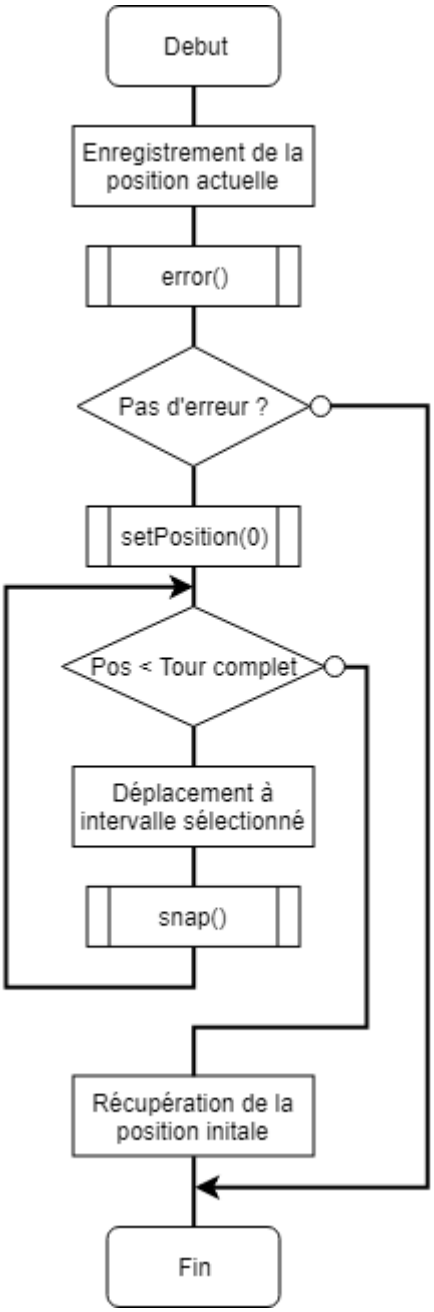
MyStepper, est une classe fille de la classe *Accelstepper*. Elle hérite donc de toutes les méthodes de la classe *Accelstepper* mais en possède, de nouvelles, définies dans le programme et qui sont spécifiques au projet de la table tournante. À savoir: *initPos*, *snap*, *runSnap*, *goTo*, *stepToAngle*, *angleToStep*, *set_mode*, *error*, *errorMessage*.

```
// ----- New class MyStepper, Accelstepper daugther -----
class MyStepper: public AccelStepper {
  public:
    MyStepper():AccelStepper(AccelStepper::DRIVER,STEP,DIR) {}

}; // End of MyStepper declaration
MyStepper turntable;
```

Rotation du plateau et déclenchement de l'appareil photo - Méthode runSnap et snap

Il s'agit ici de la fonctionnalité première de la table tournante, l'automatisation du processus de récupération de données pour la reconstitution d'image par photogramétrie. La table tourne à des intervalles angulaires précis et déclenche ensuite l'appareil photo jusqu'à ce qu'elle ait effectué un tour complet. Pour cela, deux méthodes de la classe *MyStepper* sont utilisées : *snap* et *runSnap*.

Ordinogramme de la méthode *runSnap*:



runSnap est appelée depuis la boucle principale et prend pour argument un entier de 32 bits qui correspond à l'intervalle angulaire choisi.

Lorsqu'elle est appelée, une vérification de l'état du système va être effectuée par la méthode *error* de la classe *MyStepper*. Si un 1 est renvoyé, c'est qu'une erreur est détecté,e et dans ce cas le programme retourne dans la boucle principale. Si c'est un 0 qui est renvoyé, il n'y a pas d'erreur et la méthode poursuit.

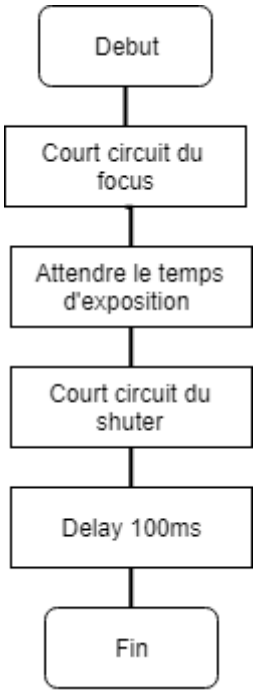
La position actuelle du plateau va être affectée à une variable globale (*memPos*) – Cette position du plateau est connue du système par un comptage des pas du moteur initialisé au démarrage – et ensuite remise à zéro par la méthode *setCurrentPosition* héritée de la classe *Accelstepper*.

Le plateau va tourner jusqu'à l'intervalle passé en paramètre de la méthode et une fois celui-ci atteint, la méthode *snap* sera appelée pour le déclenchement de l'appareil photo. Cette boucle se répétera ainsi jusqu'à ce que le système ait compté le nombre de pas nécessaire à une rotation complète, soit 80 000.

Lorsque la condition de fin de la boucle est atteinte, la position est récupérée en additionnant le nombre de pas retenu dans la variable *memPos* à ceux comptés depuis la réinitialisation.

```
// ----- function runSnap -----
void runSnap(uint32_t interval){
  long memPos=currentPosition();
  lastState = state;
  state = 6;
  if(error())return;
  setCurrentPosition(0);
  while(currentPosition()<FULL_ROT){
    move(angleToStep(interval%360));
    runToPosition();
    snap();
  lastState = state;
  state = 6;
  }
  setCurrentPosition((memPos+currentPosition())%FULL_ROT);
  return;
} // End of function RunSnapp
```

Ordinogramme de méthode *snap*:



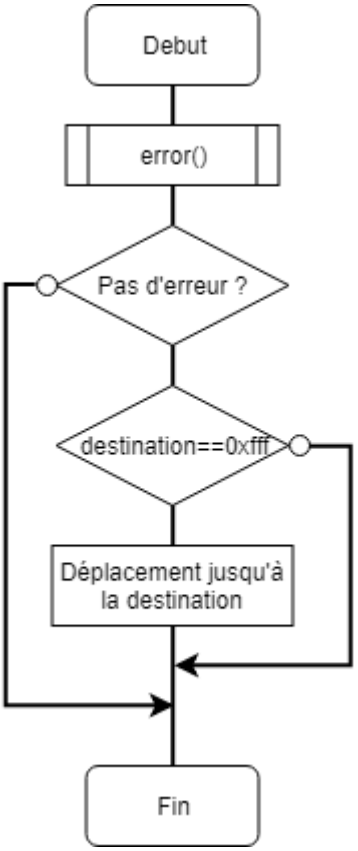
La méthode *snap*, est quant à elle, uniquement appelée depuis la méthode *runSnap*. Lorsque c'est le cas, la broche *focus* de l'appareil photo est mise à la masse. Le système attend ensuite le délai d'exposition[6] avant de mettre la broche *shutter*, également, à la masse. La photo prise, les broches sont remisent à l'état haut et le programme retourne dans le sous-programme *runSnap*.

```
// ----- Function snap -----
void snap() {
  lastState = state;
  state=7;
  digitalWrite (FOCUS, LOW);
  delay (conf.expTime*1000); // Conversion en millisecondes
  digitalWrite (SHUT, LOW);
  delay(100);
  digitalWrite (SHUT, HIGH);
  digitalWrite (FOCUS, HIGH);
  return;
} // End of function snapp
```

[6] Valeur du délai configuré dans la variable conf.expTime

Rotation du plateau à un angle précis - Méthode goTo

Ordinogramme de ma méthode goTo:



Le mode *goTo* permet de déplacer le plateau à une position précise. La méthode est appelée depuis la boucle principale et prend en paramètre l'angle de destination. La saisie de cette valeur se fait par la fonction *Acqnum* [7] et avec la télécommande infrarouge.

Comme pour la méthode *runSnap*, une vérification de l'état du moteur et des paramètres est effectuée. En cas d'erreur, ou s'il reçoit la valeur 0xFFF en paramètre, le programme retourne dans la boucle principale; sinon il poursuit.

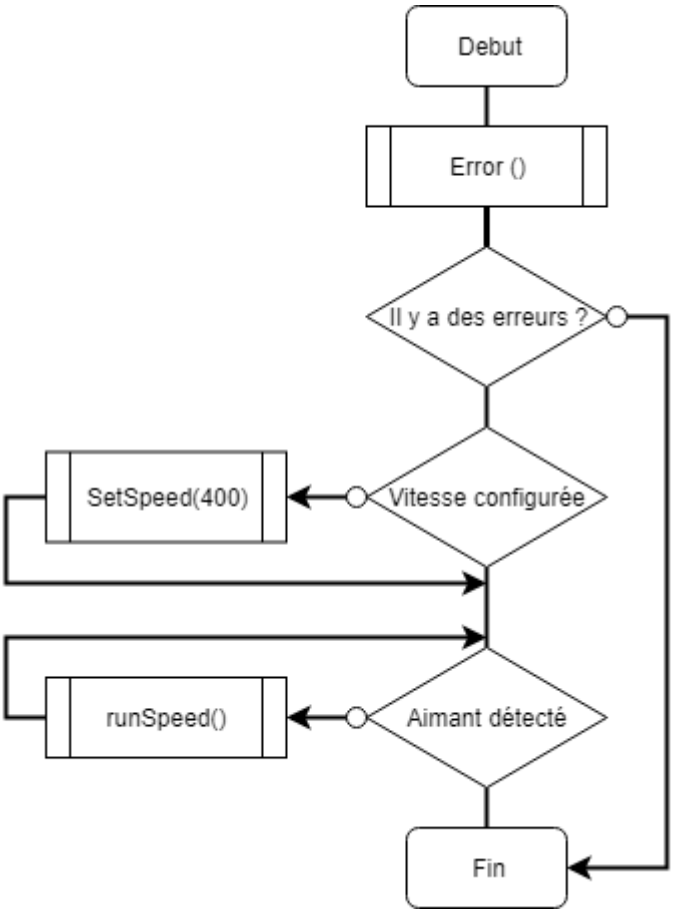
Elle utilise la méthode *moveTo*, héritée de la classe *Accelstepper*, avec en paramètre l'angle de destination converti en nombre de pas par la méthode *angleToStep*.

[7] La fonction *acqNum* permet d'entrer une valeur à plusieurs chiffres via la télécommande infrarouge. Elle sera expliquée plus en détail dans la partie *Fonctions diverses* de ce chapitre

```
// ----- Function goTo -----
  void goTo(uint32_t destination){
    lastState = state;
    state=5;
    if(error())return;
    if(destination==0xfff)return ;
    moveTo(angleToStep(destination%360));
    runToPosition();
    return;
  } // End of Function goTo
```

Remise à zéro - fonction initPos

La méthode *initPos* permet de repositionner le plateau de la table à un point de référence.



Un capteur à effet hall se trouve dans le support de la table et un aimant est attaché sous le plateau. La méthode va faire tourner le plateau à vitesse constante jusqu'à ce que le

capteur détecte l'aimant.

```
// ----- Function initPos -----
    void initPos() {
      lastState = state;
      state=8;
      if(error())return;
      if(!speed())setSpeed(400);
      while(digitalRead(SHALL)) runSpeed();
      setCurrentPosition(0);
      return;
    }// End of function initPos
```

Paramétrage

Afin de convenir aux différents besoins, certaines caractéristiques de la table se doivent d'être configurables de manière simple et rapide par l'utilisateur. Parmi ces aspects paramétrables, nous retrouvons : le temps d'exposition de l'appareil photo, l'écart angulaire entre chaque prise de photo du mode *runSnap*, l'accélération du moteur et la vitesse maximale que celui ci peut atteindre. La valeur de ces paramètres sont affectées aux variables de la structure *conf*, de type configuration, déclarée en début de programme.

```
// ----- EEPROM Adress -----
const uint16_t ADRESS_CONF = 0x00 ;

// ----- Global  variables  -----
struct configuration{
  bool mem ; // Faux si une configuration est restée en mémoire
  uint16_t expTime ; // Temps d'attente avant le déclanchement de l'appareil
photo
  uint16_t interv ; // Distance angulaire entre chaque étape
  uint16_t accel ; // Accélération du moteur
  uint16_t max_Speed; // Vitesse maximale du moteur
};
configuration conf = {1,1,10,400,750};

void setup(){
  // ----- Init Config -----
  turntable.set_mode(MODE);
  turntable.setMaxSpeed(conf.max_Speed);
  turntable.setAcceleration(conf.accel);
  loadEEPROM();
} // End of setup
```

Temps d'exposition:
Le temps d'exposition s'exprime en secondes et désigne le délai d'attente entre l'activation du *focus* et du *shuter* lors du déclenchement de l'appareil photo. Il se configure en affectant une valeur à la variable *expTime* de la structure *conf* depuis la boucle principale. Cette variable sera ensuite directement utilisée dans la fonction *snap.*

```
void loop(){
  lastState=state;
  state=1;
  if(irrecv.decode(&resultsIR)){
    irrecv.resume(); // Receive the next value
    if(resultsIR.value == LG_INPUT){ // Bouton Imput
      lastState = state;
      state=2;
      delay(500);
      conf.expTime = acqNum(3,conf.expTime);
      if(turntable.error());
      saveEEPROM();
    } // End  else if
  } // End  if IRrecv
} // End of Main loop
```

Accélération:
L'accélération se configure en affectant une valeur allant de 1 à 999 à la variable *accel* de la structure *conf* depuis la boucle principale. Cette variable sera ensuite prise pour argument de la méthode *setAcceleration* de la classe *MyStepper*, héritée de la classe *Accelstepper*.

Vitesse maximale:
A l'instar de l'accélération, la vitesse maximale du moteur se configure en affectant une valeur allant de 1 à 9999 à la variable *max_Speed* de la structure *conf* depuis la boucle principale. Cette variable sera ensuite prise pour argument de la méthode *setMaxSpeed* de la classe *MyStepper*, héritée de la classe *Accelstepper*

Sauvegarde et chargement automatique de la configuration:
L'Arduino Nano qui sert de microcontrôleur au projet, est pourvu d'une mémoire EEPROM interne de 1024 bytes

Celle-ci est utilisée avec les fonctions *saveEEPROM* et *loadEEPROM* afin de sauvegarder chaque modifications des paramètres et les charger au démarrage du système.
Les variables de configuration sont déclarées dans une structure par soucis de clarté du code. Mais cela permet aussi de pouvoir définir de manière relative l'emplacement des

variables dans l'EEPROM de l'Arduino en indiquant uniquement l'emplacement de la structure *conf*. La première variable de cette structure, *mem*, est une variable booléenne qui a pour valeur *faux* si une structure type configuration été précédemment sauvé dans l'EEPROM et *vraie* sinon.

La fonction *saveEEPROM* sera appelée à chaque modification d'un paramètre dans la boucle principale et va sauvegarder dans l'EEPROM de l'Arduino les valeurs actuelles d'accélération, de vitesse maximale et de temps d'exposition. Pour cela, elle affecte la valeur *faux* à la variable *mem* et appelle la méthode *EEPROM.put* avec l'adresse 0x00 et la structure *conf* en arguments.

La fonction *loadEEPROM* sera appelée à l'initialisation du programme et va elle vérifier l'état de la variable *mem* stockée dans l'EEPROM. Si celle-ci est *vraie*, les variables de configuration prendront leur valeur prédéfinie lors de l'initialisation du programme. Si elle est *fausse*, les valeurs de la structure de type configuration sauvegardée seront chargées à l'aide de la méthode *EEPROM.get* et affectées à la structure *conf* dans le programme.

```
// ----- Function saveEEPROM -----
void saveEEPROM (){
  conf.mem = false ;
  EEPROM.put(ADRESS_CONF,conf);
  return;
} // End of function saveEEPROM

// ----- Function loadEEPROM -----
void loadEEPROM (){
  if(!EEPROM.get(ADRESS_CONF,conf.mem)){
    conf=EEPROM.get(ADRESS_CONF,conf);
    lcd.clear();
    lcd.print("conf Loaded");
    delay(500);
  } // End if
  return;
} // End of function loadEEPROM
```

Réinitialisation des paramètres d'origine:

Le rétablissement des paramètres originaux se fait par une affectation de leur valeur initiale à la structure de type configuration. Un EEPROM.get est ensuite effectué pour affecter la valeur *vraie* à la variable *mem* dans l'EEPROM de l'Arduino.

```
void loop(){
  lastState=state;
  state=1;
  if(irrecv.decode(&resultsIR)){
    irrecv.resume(); // Receive the next value
    if(resultsIR.value == LG_TEXT){ // Bouton Text
      lcd.clear();
      lcd.print("Reset Param");
      conf = {1,1,1,400,750};
      EEPROM.put(ADRESS_CONF,conf.mem);
      turntable.setAcceleration(conf.accel);
      turntable.setMaxSpeed(conf.max_Speed);
      delay(1000);
    } // End  else if
  } // End  if IRrecv
} // End of Main loop
```

Commande infrarouge

La table tournante se contrôle entièrement à l'aide de la télécommande infrarouge. Les valeurs décodées des signaux infrarouges de chaque bouton de la télécommande sont déclarées comme constantes en début de programme afin de rendre le code plus lisible et d'être modifier facilement pour l'adapter à d'autres télécommandes[8].

```
// ----- Remote Control -----
const uint32_t LG_NUM0  =  551487735;
const uint32_t LG_NUM1  =  551520375;
const uint32_t LG_NUM2  =  551504055;
const uint32_t LG_NUM3  =  551536695;
const uint32_t LG_NUM4  =  551495895;
const uint32_t LG_NUM5  =  551528535;
const uint32_t LG_NUM6  =  551512215;
const uint32_t LG_NUM7  =  551544855;
const uint32_t LG_NUM8  =  551491815;
const uint32_t LG_NUM9  =  551524455;
const uint32_t LG_RA    = 551486205;
const uint32_t LG_DA    = 551518845;
const uint32_t LG_OFF   = 551489775;
const uint32_t LG_INPUT = 551538735;
const uint32_t LG_MENU  = 551535165;
const uint32_t LG_INFO  = 551507370;
const uint32_t LG_TEXT  = 551509665;
const uint32_t LG_GUIDE = 551540010;
const uint32_t LG_HOME  = 551501505;
const uint32_t LG_VIEW  = 551508135;
const uint32_t LG_BACK  = 551490795;
const uint32_t LG_EXIT  = 551541285;
```

La réception ainsi que le décodage des trames infrarouges se font à partir des objets *decode_results* et *IRrecv* de la librairie *IRremote*.

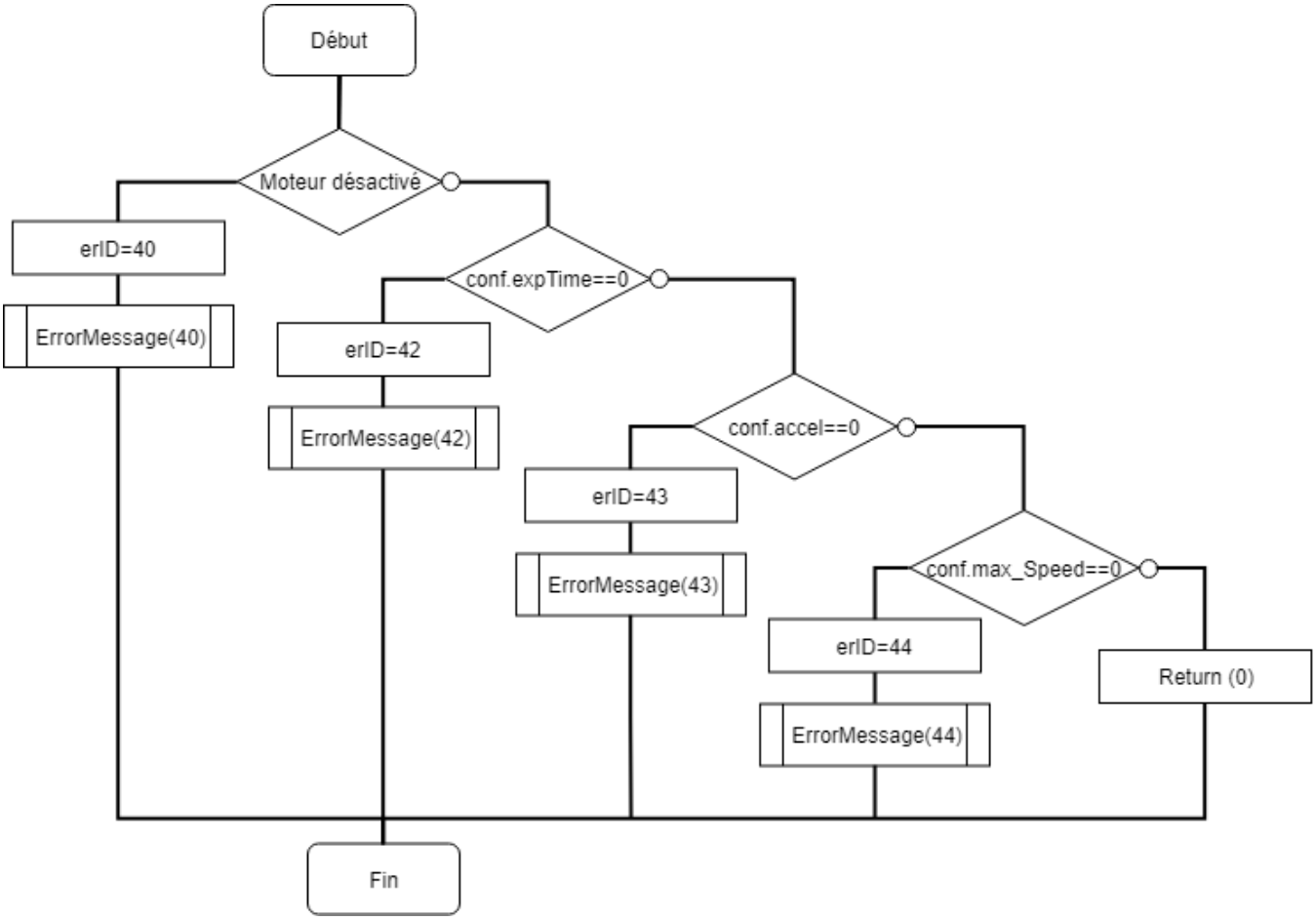
[8] La liste des codes infrarouges des différents boutons et des fonctionnalités associées, se trouve en annexe

Programmation défensive

La programmation défensive sert à éviter que le programme dysfonctionne à cause d'erreur de manipulation de la part de l'utilisateur. Par exemple, lancer le déplacement du plateau de la table alors qu'une valeur nulle est configurée comme vitesse maximale.

Méthode error:

La méthode *error*, de la classe *MyStepper*, va être appelée depuis la boucle principale après chaque modification de paramètres par l'utilisateur et avant chaque démarrage du moteur.

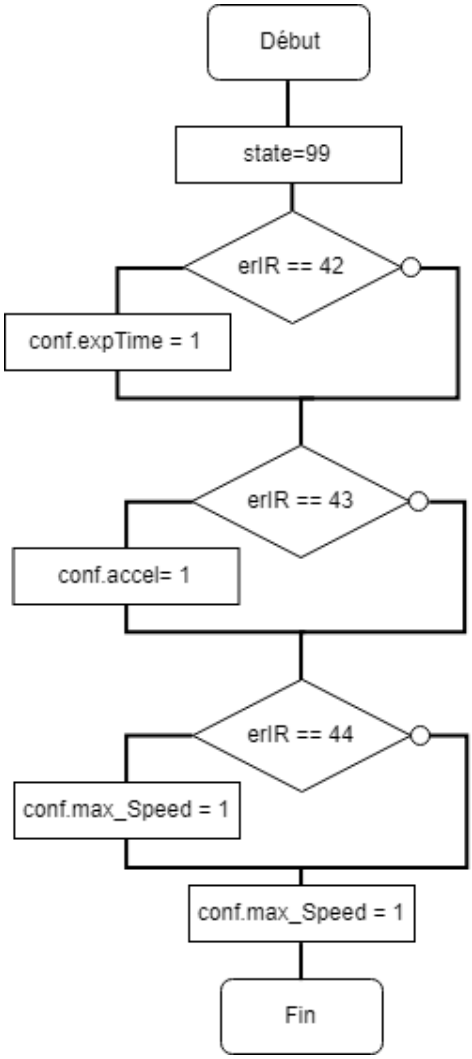


Elle va procéder à une série de vérification afin de déterminer si le système est en état d'accomplir ce qu'on lui demande ou non. Si il n'y pas d'erreurs, la méthode renvoie un 0, si il y en a, elle appelle la méthode *errorMessage* de la classe *MyStepper* avec, comme argument, un identifiant en fonction de l'erreur rencontrée.

```
// ----- Function Error -----
  uint8_t error (){
  lastState=state;
  state=98;
    if(enableMotor)errorMessage(erID=40); //Moteur désactivé
    else if(!conf.expTime)errorMessage(erID=42); //Temps d'exposition nul
    else if(!conf.accel)errorMessage(erID=43); //Accélération nulle
    else if(!conf.max_Speed)errorMessage(erID=44); //Vitesse maximale nulle
    else {
      state=lastState;
      return(0);
    }
  } // End of function checkError
```

Méthode errorMessage:

Cette méthode, de la classe *MyStepper*, appelée uniquement depuis la méthode *error*, prend pour argument un identifiant qui diffère selon l'erreur détectée. Elle va servir à informer l'utilisateur de celle-ci via un affichage qui dépendra de l'identifiant transmit.



```
// ----- Function errorMessage -----
  void errorMessage (uint8_t id){
  state=99;
  if(id == 42)conf.expTime=1;
  if(id == 43)setAcceleration(conf.accel=1);
  if(id == 44)setMaxSpeed(conf.max_Speed=1);
  delay(2000);
  } // End of function error
```

Machine d'état

Le programme fonctionne sur le principe des machines d'états. On compte ici 13 états correspondant aux différents modes ou états du système:

État 0 → Initialisation du système

État 1 → IDLE, boucle principale

État 2 → Configuration du temps d'exposition

État 3 → Configuration de l'accélération

État 4 → Configuration de la vitesse maximale

État 5 → Mode *goTo*, déplacement du plateau de la table à un angle déterminé

État 6 → Mode *runSnap*

État 7 → Déclenchement de l'appareil photo

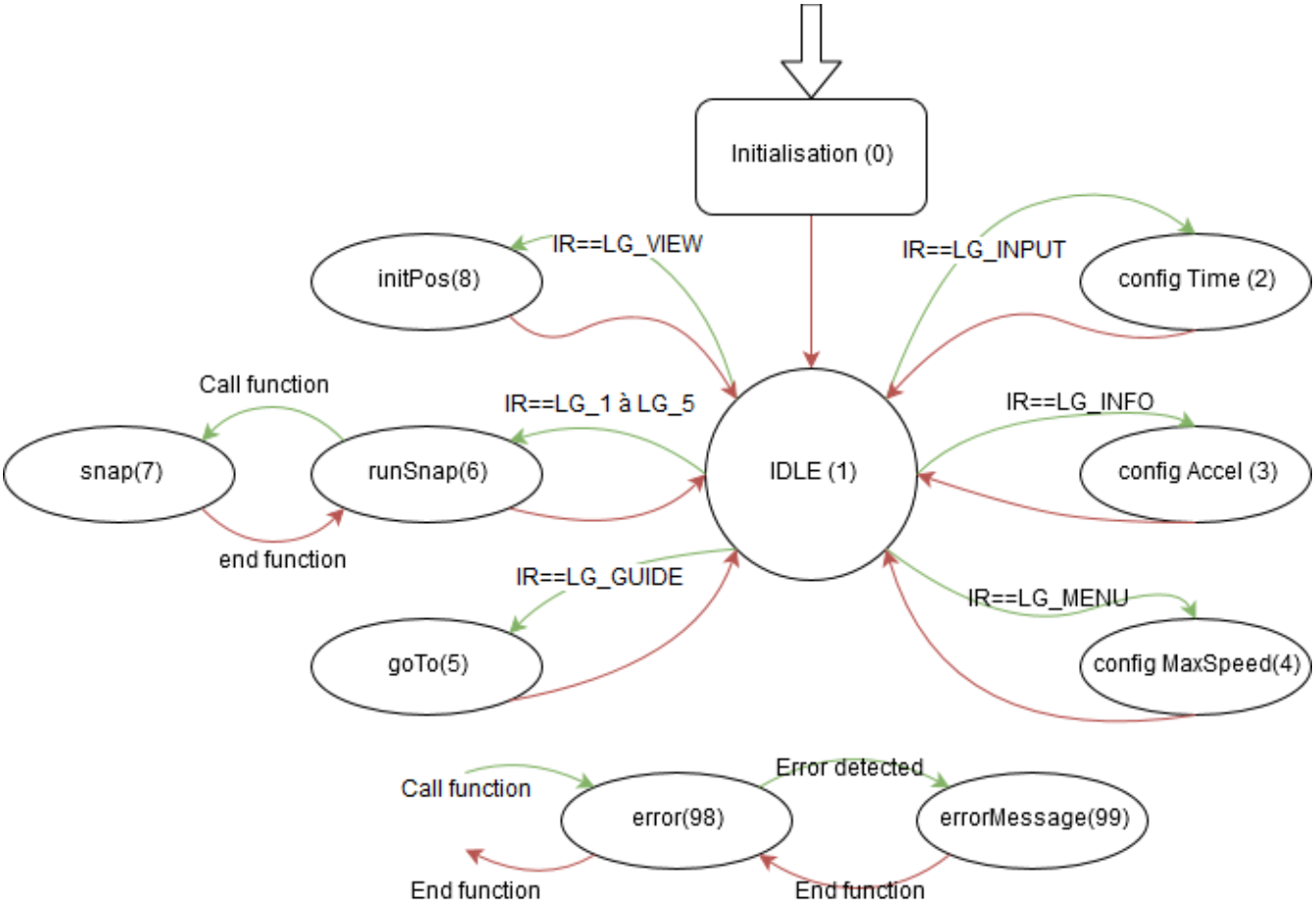
État 8 → Mode *initPos*, remise à zéro de la position du plateau

État 9 → Saisi de valeurs par la télécommande infrarouge

État 10 → Configuration du mode du moteur pas à pas

État 98 → Vérification d'erreurs

État 99 → Erreur détectée



Timer

On déclare le *timer1* de l'Arduino dans le bloc d'initialisation en début de programme. On initialise le timer avec une valeur de 500000; ce qui correspond à une durée d'une demi seconde. Lorsque le timer se finit, il déclenche une interruption qui, dans les états 5, 6, 7 et 8, correspondant aux modes *goTo*, *runSnap* et *initPos*, vérifie si le système a reçu un signal provenant du bouton d'arrêt de la télécommande infrarouge. Si c'est le cas, les variables sur lesquelles reposent les conditions de fin de la boucle où se situe le programme, sont incrémentées jusqu'à atteindre les dites conditions. On a donc une certaine latence entre l'appuie sur le bouton d'arrêt et la sortie du mode en cours d'exécution. Mais ceci était un compromis entre un ralentissement globale du système lorsqu'on diminue la valeur du timer et une plus grande latence lorsqu'on l'augmente.

```
void setup(){
  // ----- Init Timer -----
  Timer1.initialize(500000); // Initialise le Timer1  avec une période d'une
demi seconde
  Timer1.attachInterrupt(Tim1); // Attaches Tim1() à l'interruption lié au
debordement du Timer1
} // End of steup
```

Affichage

L'affichage se fait pratiquement entièrement par les interruptions du *timer1* et en fonction de la variable d'état du programme. Cela permet d'avoir un affichage presque en temps réel (l'affichage se rafraîchi toute les demis secondes) même lorsque le programme est dans une boucle.

```
// ----- Interrupts -----

void Tim1(){
  digitalWrite(LED_BUILTIN,!digitalRead(LED_BUILTIN));
  switch (state){
    case 0: // initalisation
      break;
    case 1: // IDLE
      digitalWrite (GLED, !enableMotor);
      lcd.clear();
      lcd.print("Position: ");
      lcd.print(turntable.stepToAngle(turntable.currentPosition()));
      lcd.setCursor(10,2);
      lcd.print(" St:");
      lcd.print(state);
      break;
    case 2: // Configuration ExpoTime
      lcd.clear();
      lcd.print("Time: ");
      lcd.print(conf.expTime);
      lcd.print(" second");
      break;
    case 3: // Configuration Acceleration
```

Fonctions et méthodes diverses

Méthodes angleToStep:
La méthode *angleToStep* de la classe *MyStepper*, permet de convertir un déplacement angulaire en nombre de pas pour le moteur pas-à-pas en prenant en compte son mode (le quart de pas) et le planétaire de 1:100.

Méthode stepToAngle:
La méthode *stepToAngle*, de la classe *MyStepper*, va effectuer l'opération inverse de la précédente méthode et va donc convertir un nombre de pas en un déplacement angulaire.

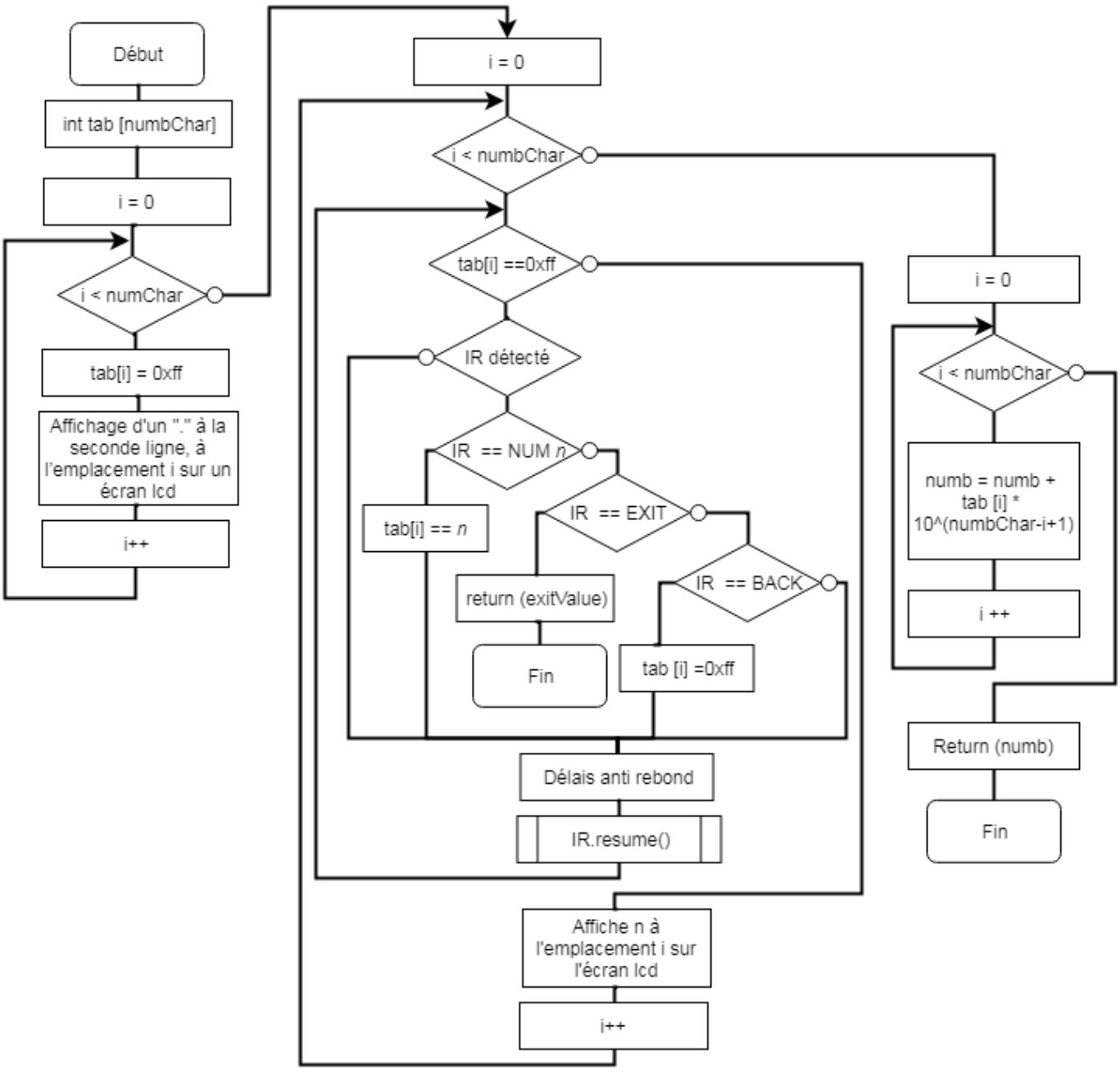
```
// ----- function stepToAngle -----
  float stepToAngle (long stepValue) { //Configuré sur un quart step +
reducteur 1:100
    float angleValue = (stepValue%FULL_ROT) *(1.8 /(ST_MOD*RP));
    return (angleValue);
  } // End of function stepToAngleErro

  // ----- function angleToStep -----
  long angleToStep (float angleValue){ // Configuré sur un quart step +
reducteur 1:100
    long stepValue = angleValue*(ST_MOD*RP)/1.8;
    return (stepValue);
  } // End of function angleToStep
```

Fonction acqNum:
acqNum, est une fonction qui va permettre de saisir un nombre comprenant plusieurs chiffres à l'aide de la télécommande infrarouge. La fonction prend en paramètre le nombre de chiffres à saisir, fixant ainsi la limite de valeur maximale, et la valeur à retourner en cas de sortie du sous-programme. La fonction ne permet que de concaténer des chiffres de 1 à 0 afin de former un entier. Elle ne permet pas d'entrer d'autres caractères, de nombres décimaux, ou de négatifs, on empêche ainsi l'utilisateur d'entrer des valeurs hors format.

La fonction construit un tableau *tab* de la taille du nombre du paramètre *numChar* et le remplir de 0xff.

Ensuite la fonction va entrer dans une boucle *POUR* qui a *i* comme variable de teste et la valeur du paramètre *numChar* comme condition de fin. Dedans, tant que la valeur stockée dans le tableau à l'indice *i* vaut 0xff, la fonction va scruter en boucle si un code

infrarouge correspondant à un chiffre est détecté. Lorsque c'est le cas, la valeur du chiffre est placée dans le tableau à l'emplacement de l'indice *i* à la place du 0xff et l'indice *i* s'incrémente. L'opération est répétée jusqu'à ce que le tableau ne contienne plus de 0xff. Après être sorti de la boucle, Les chiffre sont concaténés par quelques simples opérations arithmétiques.



```
// ----- function acqNum -----
int acqNum(int numbChar,int ExitValue){
  lastState= state;
  state=9;
  int tab[numbChar];
  int numb =0;
  int i;
  for(i=0;i<numbChar;i++){
    tab[i] = 0xff ;
    lcd.setCursor(i,2);
    lcd.print(".");
  }
  delay(200); //anti rebond
  for(i=0;i<numbChar;i++){
    while(tab[i]==0xff){
      if(irrecv.decode(&resultsIR)){
        if (resultsIR.value == LG_NUM1)       tab[i] = 1;
        else if (resultsIR.value == LG_NUM2) tab[i] = 2;
        else if (resultsIR.value == LG_NUM3) tab[i] = 3;
        else if (resultsIR.value == LG_NUM4) tab[i] = 4;
        else if (resultsIR.value == LG_NUM5) tab[i] = 5;
        else if (resultsIR.value == LG_NUM6) tab[i] = 6;
        else if (resultsIR.value == LG_NUM7) tab[i] = 7;
        else if (resultsIR.value == LG_NUM8) tab[i] = 8;
        else if (resultsIR.value == LG_NUM9) tab[i] = 9;
        else if (resultsIR.value == LG_NUM0) tab[i] = 0;
        else if (resultsIR.value == LG_EXIT) return(ExitValue); // Exit
        else if (resultsIR.value == LG_BACK) {    // Back
          i=max(i-1,0); // Empêche l'indice d'être négatif en cas de spam
          tab[i]=0xff;
          lcd.setCursor(i,2);
          lcd.print(".");
        }
        delay(100); // Anti rebond
        irrecv.resume();
      } // End  if
    } // End  while
    lcd.setCursor(i,2);
    lcd.print(tab[i]);
  } // End for
  for(i=0;i<numbChar;i++) numb = numb + tab[i]*ipow(10,numbChar-(i+1));
  return (numb);
} // End of function acqNum
```

Description des outils et méthodes de développement

Arduino

Arduino est une marque qui développe des cartes à microcontrôleur et une série de modules directement dont les plans et schémas son publiés en licence libre. Les cartes disposent généralement d'un microcontrôleur d'architecture *Atmel AVR* ou *ARM*

Arduino à l'avantage de disposer d'une grande gamme de cartes différentes et d'une commuté très active ayant développé de nombreuses librairies.

Il dispose aussi de son propre environnement de développement, *Arduino IDE*, dans lequel est pré-intégré un certain nombre de librairies et de programmes démo. L'Arduino se programme en *C* ou *C++*; les possibilités d'allouer de la mémoire dynamiquement restent tout de même limité.

Git et Github

Git est un outil de gestion des versions développé par Linus Torvalds, créateur du noyau *Linux*.

Git va permettre de créer un dossier git. Chaque modification enregistrée d'un fichier à l'intérieur du dossier git va être indexée dans une base de données locale pour créer une nouvelle version du projet.

Cela permet de travailler sans risquer de "casser" son code vu qu'il est toujours possible de revenir à une version du code antérieure aux changements apportés. Il est en outre impossible de modifier le contenu d'un fichier sans que Git ne le détecte. Pour cela, Git utilise un mécanisme appelé *empreinte Sha-1*. Le principe est de générer une chaîne de 40 caractères hexadécimaux en fonction du contenu des fichiers et de la structure du dossier. Le moindre bit modifié change la chaîne de caractères. Git assure ainsi l'intégrité des fichiers qui ont été validés [9]

Git permet aussi de créer des branches afin de travailler en parallèle sur le code. Ainsi, il est possible de développer les différentes fonctionnalités séparément sur des copies du projet en cours. Lorsqu'elles fonctionnent correctement on les implémente en fusionnant la branche de tests vers la branche de développement. Une fois assuré que les nouvelles fonctionnalités n'entrent pas en conflit avec d'autres dans la branche de développement,

[9] *commit* selon le terme technique

on peut fusionner celle-ci à la branche principale pour créer une nouvelle version du projet. Si les modifications apportées sur une branche ne conviennent pas, il suffit de la supprimer et aucune des validations faites sur celle-ci ne sera déportée sur la branche “mère".

(*Image provenant du site : https://github.com/webSebLabbe/mini-help-git*)

GitHub est, quand à lui, une plate-forme en ligne de "dossiers git distants" sur laquelle il est possible de publier son dossier git local. Il devient ainsi un dossier distant hébergé sur les serveurs de Github. Dès lors, son contenu pourra être téléchargé (cloné) afin de synchroniser les versions et d’assurer un contrôle visuel des changements en ligne.

DesignSpark

DesignSpark est un logiciel gratuit de conception assistée par ordinateur (CAO) développé par la compagnie *RS Components*. Il a été utilisé pour le dessin des schémas électroniques et des circuits imprimés.

SolidWorks

SolidWorks, développé par *Dassault Systèmes*, est un logiciel de conception assistée par ordinateur qui permet la modélisation tridimensionnelle d’éléments mécaniques et simuler leur assemblage. Il a été utilisé pour la création d’un visuel du projet ainsi que pour la mise en plan des différents éléments de la structure mécanique.

Fritzing

Fritzing est un petit logiciel libre, toujours en phase de test, développé par interaction Design Lab Potsdam. Le logiciel dispose d'une grande base de données de composants électroniques et offre la possibilité de simuler une platine d'essais. A partir de cette platine d'essais, le programme génère un schéma électronique. Le programme dispose d'autres fonctionnalités, comme le dessin de circuits imprimés et l'autoroutage des pistes. Mais celles-ci ne sont encore forts limitées.

Conclusions

La table a été conçue, fabriquée, programmée et l'ensemble est fonctionnel.

Malheureusement le cahier des charges n'a pas entièrement été respecté: il restait trop peu de temps pour finaliser convenablement le déclencheur infrarouge. La solution d'un déclencheur relié par câble à la table a donc été convenue lors des derniers jours. Cette solution convient tout autant, mais est moins élaborée.

Certain défauts sont toujours présents comme l'erreur *backlash* créée par les engrenages du planétaire et amplifiée par la grandeur du rayon du plateau. Le problème étant lié à la conception même du projet, il n'était plus possible de la corriger sans modifier toute la conception. Une solution aurait été de déporter le moteur au lieu de fixer celui-ci directement au plateau et d'utiliser une courroie crantée. Une autre, aurait été d'utiliser un moteur à courant continu avec un encodeur rotatif et une régulation de type PI. Heureusement, le poids conséquent du plateau s'oppose au recul de celui-ci et lors d'une utilisation "normale" de la table, cette erreur ne s'observe pas.

Comme autre défaut, on peut noter des imperfections de précision dans l'ouvrage de la table. Mais ils sont toute foi à relativiser si l'on considère les moyens mis à disposition pour la réalisation du projet.

Pour finir, je dirais que si le projet de la table tournante peut sembler léger à première vue, il n'en est rien. A travers ce projet, on retrouve nombre d'éléments vu, en profondeur ou non, lors du cursus: contrôle de moteur, programmation orienté objet, interfaçage de capteur, machine d'état, interruptions, timers, mécanique, masques binaires, etc ...

De plus, la longueur du programme couplée aux changements récurrent du cahier des charges, force une certaine rigueur dans l'écriture du code et la méthode de développement.

Annexe

Code du programme

```
#include <AccelStepper.h>
#include <LiquidCrystal.h>
#include <EEPROM.h>
#include <IRremote.h>
#include <TimerOne.h>

// ----- Arduino Nano Pin -----
#define RECV  2   //PORTD2
#define EN    3   //PORTD3
#define M0    4   //PORTD4
#define M1    5   //PORTD5
#define M2    6   //PORTD6
#define STEP  7   //PORTD7
#define DIR   8   //PORTB0
#define FOCUS 9   //PORTB1
#define SHUT  10  //PORTB2
#define SHALL 11  //PORTB3
#define GLED  12  //PORTB4
#define LCD_EN 14 //PORTC0
#define LCD_RS 15 //PORTC1
#define LCD_D4 16 //PORTC2
#define LCD_D5 17 //PORTC3
#define LCD_D6 18 //PORTC4
#define LCD_D7 19 //PORTC5

// ----- Stepper -----
const uint8_t ST_MOD = 4; // Stepping mode
const uint8_t RP = 100;   // Réducteur planétaire
const uint8_t MODE = 3;   // Sélection du mode dasns le switch case
const uint32_t FULL_ROT = 80000; // Rotation complète: 80000step

// ----- Remote Control -----
const uint32_t LG_NUM0  =  551487735;
const uint32_t LG_NUM1  =  551520375;
const uint32_t LG_NUM2  =  551504055;
const uint32_t LG_NUM3  =  551536695;
const uint32_t LG_NUM4  =  551495895;
const uint32_t LG_NUM5  =  551528535;
const uint32_t LG_NUM6  =  551512215;
const uint32_t LG_NUM7  =  551544855;
const uint32_t LG_NUM8  =  551491815;
const uint32_t LG_NUM9  =  551524455;
const uint32_t LG_RA    = 551486205;
const uint32_t LG_DA    = 551518845;
const uint32_t LG_OFF   = 551489775;
const uint32_t LG_INPUT = 551538735;
const uint32_t LG_MENU  = 551535165;
const uint32_t LG_INFO  = 551507370;
const uint32_t LG_TEXT  = 551509665;
const uint32_t LG_GUIDE = 551540010;
const uint32_t LG_HOME  = 551501505;
const uint32_t LG_VIEW  = 551508135;
const uint32_t LG_BACK  = 551490795;
const uint32_t LG_EXIT  = 551541285;

// ----- EEPROM Adress -----
```

```
const uint16_t ADRESS_CONF = 0x00 ;

// ----- Global  variables  -----
bool enableMotor = LOW;
uint8_t state = 0; // Machine d'état
uint8_t lastState = 0; // Etat précédent
uint8_t erID = 0;
struct configuration{
  bool mem ; // Faux si une configuration est restée en mémoire
  uint16_t expTime ; // Temps d'attente avant le déclanchement de
l'appareil photo
  uint16_t interv ; // Distance angulaire entre chaque étape
  uint16_t accel ; // Accélération du moteur
  uint16_t max_Speed; // Vitesse maximale du moteur
};
configuration conf = {1,1,10,400,750};

// ----- Objects -----
LiquidCrystal lcd(LCD_RS, LCD_EN,LCD_D4,LCD_D5,LCD_D6,LCD_D7); //
(rs,en,d4,d5,d6,d7)
decode_results resultsIR;
IRrecv irrecv(RECV);

// ----- New class MyStepper, Accelstepper daugther -----
class MyStepper: public AccelStepper {
  public:
    MyStepper():AccelStepper(AccelStepper::DRIVER,STEP,DIR) {}

  // ----- Function initPos -----
    void initPos() {
      lastState = state;
      state=8;
      if(error())return;
      if(!speed())setSpeed(400);
      while(digitalRead(SHALL)) runSpeed();
      setCurrentPosition(0);
      return;
    }// End of function initPos

  // ----- Function snap -----
  void snap() {
    lastState = state;
    state=7;
    digitalWrite (FOCUS, LOW);
    delay (conf.expTime*1000); // Conversion en millisecondes
    digitalWrite (SHUT, LOW);
    delay(100);
    digitalWrite (SHUT, HIGH);
    digitalWrite (FOCUS, HIGH);
    return;
  } // End of function snapp

  // ----- function runSnap -----
  void runSnap(uint32_t interval){
    long memPos=currentPosition();
    lastState = state;
    state = 6;
    if(error())return;
    setCurrentPosition(0);
    while(currentPosition()<FULL_ROT){
      move(angleToStep(interval%360));
```

```
      runToPosition();
      snap();
    lastState = state;
    state = 6;
    }
    setCurrentPosition((memPos+currentPosition())%FULL_ROT);
    return;
  } // End of function RunSnapp

  // ----- Function goTo -----
  void goTo(uint32_t destination){
    lastState = state;
    state=5;
    if(error())return;
    if(destination==0xfff)return ;
    moveTo(angleToStep(destination%360));
    runToPosition();
    return;
  } // End of Function goTo

  // ----- function stepToAngle -----
  float stepToAngle (long stepValue) { //Configuré sur un quart step +
reducteur 1:100
    float angleValue = (stepValue%FULL_ROT) *(1.8 /(ST_MOD*RP));
    return (angleValue);
  } // End of function stepToAngleErro

  // ----- function angleToStep -----
  long angleToStep (float angleValue){ // Configuré sur un quart step +
reducteur 1:100
    long stepValue = angleValue*(ST_MOD*RP)/1.8;
    return (stepValue);
  } // End of function angleToStep

  // ----- Function set_mode -----
  uint8_t set_mode(uint8_t mode){
    lastState = state;
    state=10;
    switch (mode){
      case 1: //Full step
        PORTD = PORTD & 0b10001111;
        return(1);
     case 2: // 1/2 step
        PORTD = PORTD & 0b10001111;
        PORTD = PORTD | 0b00010000;
        return(2);
    case 3: //  1/4 step
        PORTD = PORTD & 0b10001111;
        PORTD = PORTD | 0b00100000;
        return(4);
    case 4: //  1/8 step
        PORTD = PORTD & 0b10001111;
        PORTD = PORTD | 0b01100000;
        return(8);
    case 5: //  1/16 step
        PORTD = PORTD & 0b10001111;
        PORTD = PORTD | 0b01000000;
        return(16);
    case 6: //  1/32 step
        PORTD = PORTD | 0b01110000;
        return(32);
```

```
    default: //  1 step
        PORTD = PORTD & 0b10001111;
        return(1);
    } // End of switch
  } //End of function set_mode

 // ----- Function Error -----
  uint8_t error (){
  lastState=state;
  state=98;
    if(enableMotor)errorMessage(erID=40); //Moteur désactivé
    else if(!conf.expTime)errorMessage(erID=42); //Temps d'exposition nul
    else if(!conf.accel)errorMessage(erID=43); //Accélération nulle
    else if(!conf.max_Speed)errorMessage(erID=44); //Vitesse maximale nulle
    else {
      state=lastState;
      return(0);
    }
  } // End of function checkError

 // ----- Function errorMessage -----
  void errorMessage (uint8_t id){
  state=99;
  if(id == 42)conf.expTime=1;
  if(id == 43)setAcceleration(conf.accel=1);
  if(id == 44)setMaxSpeed(conf.max_Speed=1);
  delay(2000);
  } // End of function error

}; // End of MyStepper declaration
MyStepper turntable;

void setup(){
  lastState = 0;
  state=0;
  // ----- Init I/O -----
  pinMode(EN, OUTPUT);
  pinMode( M0 , OUTPUT );
  pinMode( M1 , OUTPUT );
  pinMode( M2 , OUTPUT );
  pinMode( GLED , OUTPUT );
  pinMode( FOCUS, OUTPUT );
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode( SHUT , OUTPUT );
  pinMode (SHALL , INPUT);
  digitalWrite(EN, LOW);
  digitalWrite(FOCUS, HIGH);
  digitalWrite(SHUT , HIGH);
  // ----- Init Serial -----
  Serial.begin(115200);
  // ----- Init LCD -----
  lcd.begin(16, 2);
  lcd.print("Initializes");
  irrecv.enableIRIn();
  delay(500);
  // ----- Init Config -----
  turntable.set_mode(MODE);
  turntable.setMaxSpeed(conf.max_Speed);
  turntable.setAcceleration(conf.accel);
  loadEEPROM();
  // ----- Init Timer -----
```

```
  Timer1.initialize(500000); // Initialise le Timer1  avec une période
d'une demi seconde
  Timer1.attachInterrupt(Tim1); // Attaches Tim1() à l'interruption lié au
debordement du Timer1
} // End of steup


void loop(){
  lastState=state;
  state=1;
  if(irrecv.decode(&resultsIR)){
    irrecv.resume(); // Receive the next value

    if (resultsIR.value == LG_OFF){ //Bouton off
      enableMotor = !enableMotor ;
      digitalWrite(EN,enableMotor);
    }
    else if(resultsIR.value == LG_INPUT){ // Bouton Imput
      lastState = state;
      state=2;
      delay(500);
      conf.expTime = acqNum(3,conf.expTime);
      if(turntable.error());
      saveEEPROM();
    }
    else if(resultsIR.value == LG_MENU){ // Bouton Q.Menu
      lastState = state;
      state=3;
      delay(500);
      turntable.setAcceleration(conf.accel=acqNum(3,conf.accel));
      if(turntable.error());
      saveEEPROM();
    }
    else if(resultsIR.value == LG_INFO){ // Bouton Info
      lastState = state;
      state=4;
      delay(500);
      turntable.setMaxSpeed(conf.max_Speed=acqNum(4,conf.max_Speed));
      if(turntable.error());
      saveEEPROM();
    }
    else if(resultsIR.value == LG_TEXT){ // Bouton Text
      lcd.clear();
      lcd.print("Reset Param");
      conf = {1,1,1,400,750};
      EEPROM.put(ADRESS_CONF,conf.mem);
      turntable.setAcceleration(conf.accel);
      turntable.setMaxSpeed(conf.max_Speed);
      delay(1000);
    }
    else if(resultsIR.value == LG_NUM1){ // Bouton 1
      turntable.runSnap(conf.interv=10);
    }
    else if(resultsIR.value == LG_NUM2){ // Bouton 2
      turntable.runSnap(conf.interv=15);
    }
    else if(resultsIR.value == LG_NUM3){ // Bouton 3
      turntable.runSnap(conf.interv=20);
    }
    else if(resultsIR.value == LG_NUM4){ // Bouton 4
      turntable.runSnap(conf.interv=30);
```

```
    }
    else if(resultsIR.value == LG_NUM5){ // Bouton 5
      turntable.runSnap(conf.interv=90);
    }
    else if(resultsIR.value == LG_NUM0){ // Bouton 0
      lcd.clear();
      lcd.print("Mise a 0");
      turntable.setCurrentPosition(0);
      delay(1000);
    }
    else if(resultsIR.value == LG_VIEW){ // Bouton Q.View
      turntable.initPos();
    }
    else if (resultsIR.value == LG_GUIDE){ // Bouton Guide.
      turntable.goTo(acqNum(3,0xfff));
    }
    else if(resultsIR.value == LG_RA){ // Bouton flèche Haut
      if(!turntable.error()){
        turntable.move(turntable.angleToStep(1));
        while (turntable.distanceToGo()!=0)turntable.run();
      } // End if error();
    }
    else if(resultsIR.value == LG_DA){ // Bouton flèche Bas
      if(!turntable.error()){
        turntable.move(-turntable.angleToStep(1));
        while (turntable.distanceToGo()!=0)turntable.run();
      } // End if error();
    } // End  else if
  } // End  if IRrecv
} // End of Main loop

// ----- Interrupts -----

void Tim1(){
  digitalWrite(LED_BUILTIN,!digitalRead(LED_BUILTIN));
  switch (state){
    case 0: // initalisation
      break;
    case 1: // IDLE
      digitalWrite (GLED, !enableMotor);
      lcd.clear();
      lcd.print("Position: ");
      lcd.print(turntable.stepToAngle(turntable.currentPosition()));
      lcd.setCursor(10,2);
      lcd.print(" St:");
      lcd.print(state);
      break;
    case 2: // Configuration ExpoTime
      lcd.clear();
      lcd.print("Time: ");
      lcd.print(conf.expTime);
      lcd.print(" second");
      break;
    case 3: // Configuration Acceleration
      lcd.clear();
      lcd.print("Accel: ");
      lcd.print(conf.accel);
      break;
    case 4: // Configuration MaxSpeed
      lcd.clear();
      lcd.print("Max Speed: ");
```

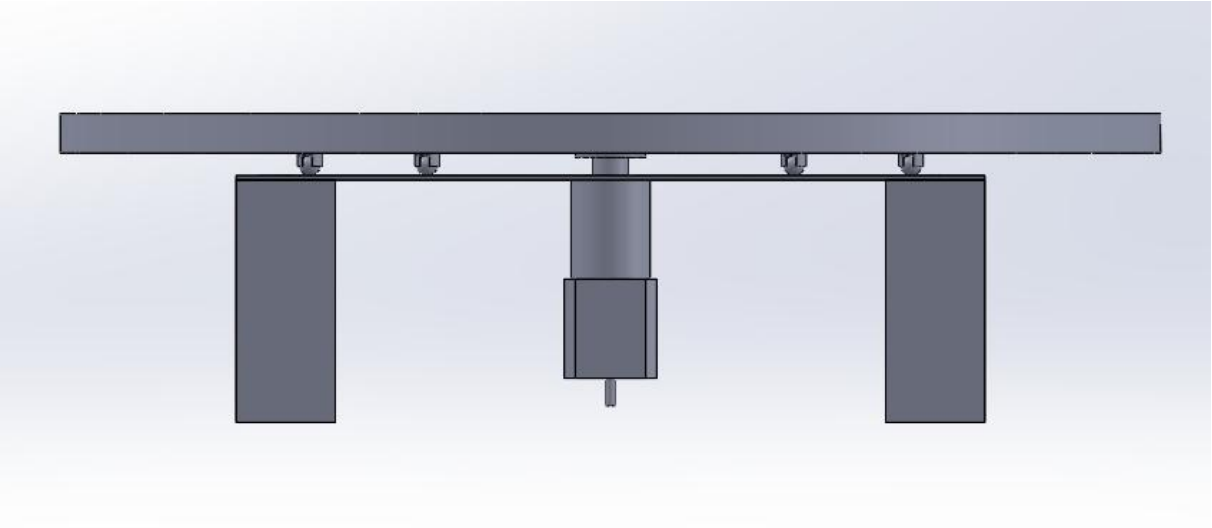
```
      lcd.print(conf.max_Speed);
      break;
    case 5: // Goto
      if(irrecv.decode(&resultsIR)){
        irrecv.resume();
        if (resultsIR.value == LG_EXIT){ //Bouton EXIT
          Serial.println("EXIT");
          turntable.setCurrentPosition(turntable.distanceToGo());
        }// End if result
      }// End if decode
      lcd.clear();
      lcd.print("Position: ");
      lcd.print(turntable.stepToAngle(turntable.currentPosition()));
      lcd.setCursor(0,2);
      lcd.print("Go to: ");
      lcd.print(turntable.stepToAngle(turntable.targetPosition()));
      lcd.setCursor(10,2);
      lcd.print(" St:");
      lcd.print(state);
      break;

    case 6: // RunSnap
      if(irrecv.decode(&resultsIR)){
        irrecv.resume();
        if (resultsIR.value == LG_EXIT){ //Bouton EXIT
          Serial.println("EXIT");

turntable.setCurrentPosition(turntable.currentPosition()+FULL_ROT);
        }// End if result
      }// End if decode
      lcd.clear();
      lcd.print("Progress: ");
      if (turntable.currentPosition()<FULL_ROT)
        lcd.print(turntable.currentPosition()/(FULL_ROT/100));
      else lcd.print("100");
      lcd.print(" %");
      lcd.setCursor(0,2);
      lcd.print(conf.interv);
      lcd.setCursor(4,2);
      lcd.print(conf.expTime);
      lcd.print("sec");
      lcd.setCursor(10,2);
      lcd.print(" St:");
      lcd.print(state);
      break;

    case 7: // Snap
      if(irrecv.decode(&resultsIR)){
        irrecv.resume();
        if (resultsIR.value == LG_EXIT){ //Bouton EXIT
          Serial.println("EXIT");
          turntable.setCurrentPosition(turntable.currentPosition()+
FULL_ROT);
        } // End if result
      } // End if decode
      lcd.setCursor(10,2);
      lcd.print(" St:");
      lcd.print(state);
      break;

    case 8: // InitPosition
```

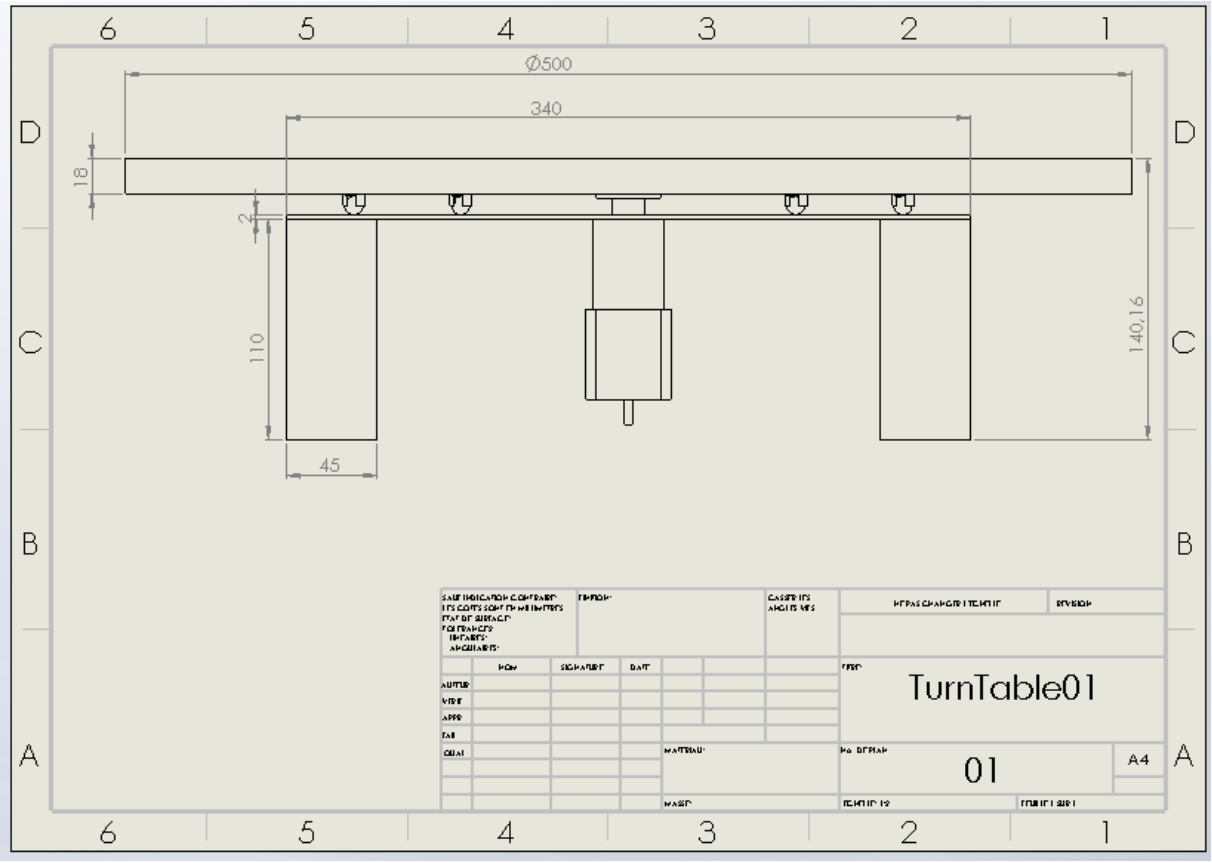
```
    if(irrecv.decode(&resultsIR)){
      irrecv.resume();
      if (resultsIR.value == LG_EXIT){ //Bouton EXIT
        Serial.println("EXIT");
        turntable.setCurrentPosition(turntable.distanceToGo());
      } // End if result
    } // End if decode
    lcd.clear();
    lcd.print("Reset Position");
    lcd.setCursor(10,2);
    lcd.print(" St:");
    lcd.print(state);
    break;

  case 9: // AcquNum
    lcd.setCursor(10,2);
    lcd.print(" St:");
    lcd.print(state);
    break;

  case 98: // error
    lcd.setCursor(10,2);
    lcd.print(" St:");
    lcd.print(state);
    break;

  case 99: // messageError
    if(erID==40){
      lcd.clear();
      lcd.print("Motor is Disable");
      lcd.setCursor(0,2);
      lcd.print("Error ");
      lcd.print(erID);
    }
    else if(erID==42){
      lcd.clear();
      lcd.print("Time value is 0");
      lcd.setCursor(0,2);
      lcd.print("Error ");
      lcd.print(erID);
    }
    else if(erID==43){
      lcd.clear();
      lcd.print("Accel value is 0");
      lcd.setCursor(0,2);
      lcd.print("Error ");
      lcd.print(erID);
    }
    else if(erID==44){
      lcd.clear();
      lcd.print("Max speed is 0");
      lcd.setCursor(0,2);
      lcd.print("Error ");
      lcd.print(erID);
    }
    lcd.setCursor(10,2);
    lcd.print(" St:");
    lcd.print(state);
    break;
  default:
    break;
```

```
  } //End of switch
} // End of Tim1


// ----- function acqNum -----
int acqNum(int numbChar,int ExitValue){
  lastState= state;
  state=9;
  int tab[numbChar];
  int numb =0;
  int i;
  for(i=0;i<numbChar;i++){
    tab[i] = 0xff ;
    lcd.setCursor(i,2);
    lcd.print(".");
  }
  delay(200); //anti rebond
  for(i=0;i<numbChar;i++){
    while(tab[i]==0xff){
      if(irrecv.decode(&resultsIR)){
        if (resultsIR.value == LG_NUM1)       tab[i] = 1;
        else if (resultsIR.value == LG_NUM2) tab[i] = 2;
        else if (resultsIR.value == LG_NUM3) tab[i] = 3;
        else if (resultsIR.value == LG_NUM4) tab[i] = 4;
        else if (resultsIR.value == LG_NUM5) tab[i] = 5;
        else if (resultsIR.value == LG_NUM6) tab[i] = 6;
        else if (resultsIR.value == LG_NUM7) tab[i] = 7;
        else if (resultsIR.value == LG_NUM8) tab[i] = 8;
        else if (resultsIR.value == LG_NUM9) tab[i] = 9;
        else if (resultsIR.value == LG_NUM0) tab[i] = 0;
        else if (resultsIR.value == LG_EXIT) return(ExitValue); // Exit
        else if (resultsIR.value == LG_BACK) {    // Back
          i=max(i-1,0); // Empêche l'indice d'être négatif en cas de spam
          tab[i]=0xff;
          lcd.setCursor(i,2);
          lcd.print(".");
        }
        delay(100); // Anti rebond
        irrecv.resume();
      } // End  if
    } // End  while
    lcd.setCursor(i,2);
    lcd.print(tab[i]);
  } // End for
  for(i=0;i<numbChar;i++) numb = numb + tab[i]*ipow(10,numbChar-(i+1));
  return (numb);
} // End of function acqNum

// ----- Function ipow -----
int ipow(int base, int expo){
  int result=1;
  for(expo;expo>0;expo--) result= result*base;
  return(result);
}// End of function ipow

// ----- Function saveEEPROM -----
void saveEEPROM (){
  conf.mem = false ;
  EEPROM.put(ADRESS_CONF,conf);
  return;
} // End of function saveEEPROM
```

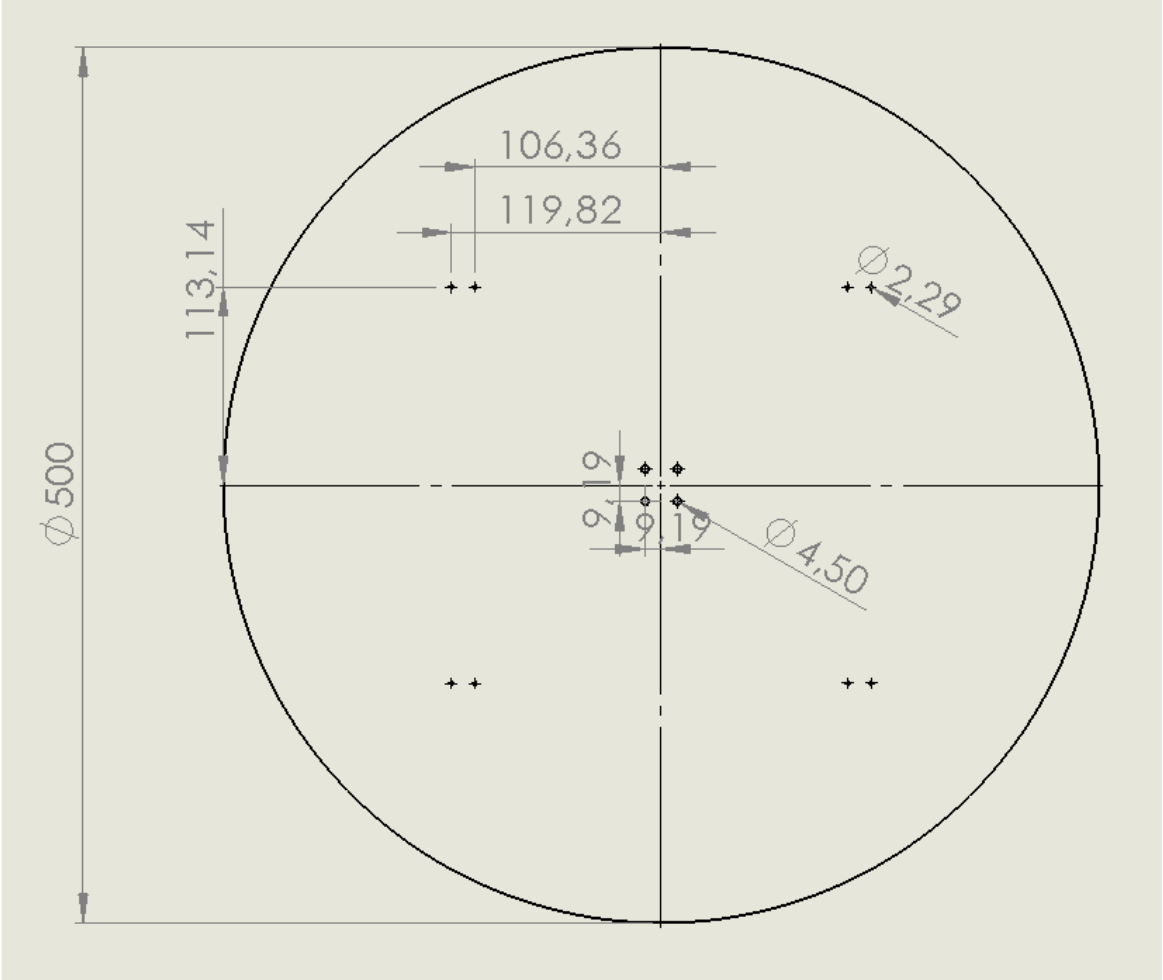
```
// ----- Function loadEEPROM -----
void loadEEPROM (){
  if(!EEPROM.get(ADRESS_CONF,conf.mem)){
    conf=EEPROM.get(ADRESS_CONF,conf);
    lcd.clear();
    lcd.print("conf Loaded");
    delay(500);
  } // End if
  return;
} // End of function loadEEPROM
```

Plans

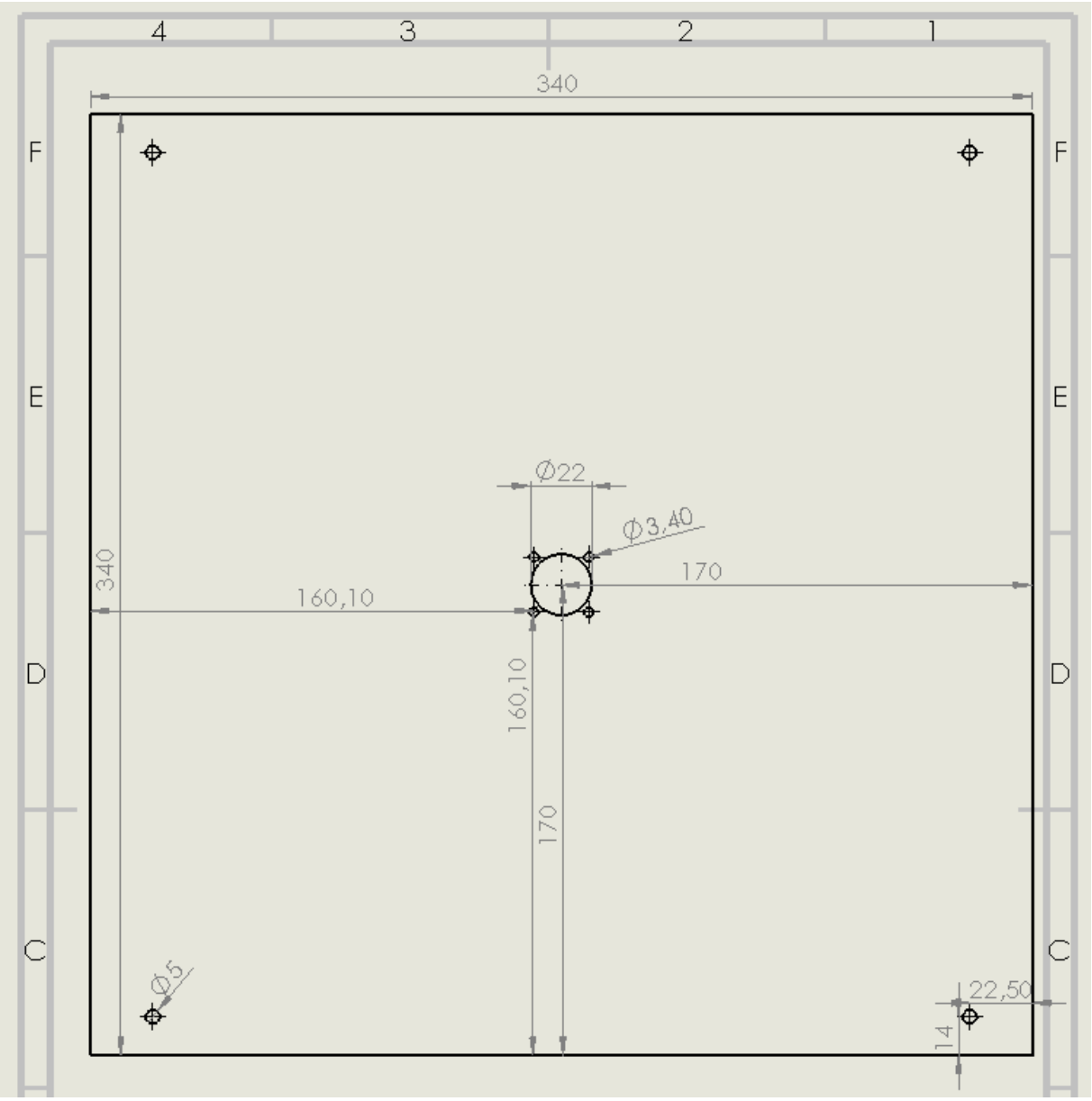




Plateau supérieur en MDF :



Plaque de support en aluminium:



Guide d'utilisation de la table

On/Off :

Le bouton *on/off* de la télécommande permet d'activer ou désactiver le moteur afin que le plateau de la table puisse être tournée à la main. Un voyant vert indique l'état du moteur.

Input :

Le bouton *imput* permet de configurer le délai entre l'activation du focus et celui du *shutter* qui déclenche la prise de photo lors du mode *Run*. Les valeurs acceptées vont de 1 à 999 et doivent être en seconde. La valeur doit être entrée sur trois chiffres. Le bouton *Back* permet de corriger, le bouton *Exit* de sortir du menu

Menu :

Le bouton *Menu* permet de configurer l'accélération de la table. Les valeurs acceptées vont de 1 à 999. La valeur doit être entrée sur trois chiffres. Le bouton *Back* permet de corriger, le bouton *exit* de sortir du menu.

Info :

Le bouton *Info* permet de configurer la vitesse maximale de la table. Les valeurs acceptées vont de 1 à 9999. La valeur doit être entrée sur quatre chiffres. Le bouton *Back* permet de corriger, le bouton *exit* de sortir du menu.

Texte :

Le bouton *Texte* permet de remettre les paramètres d'origine. à savoir :

- Délais d'exposition à 1 seconde.

- Accélération à 400

- Vitesse maximale à 850

Pavé numérique :

Les boutons du pavé numérique de 1 à 5 vont lancer le mode *Run* avec des écarts angulaires variant selon le bouton pressé. Le bouton *exit* permet de sortir du mode à tout instant.

- 1 : Prise de photo tous les 10°

- 2 : Prise de photo tous les 15°

- 3 : Prise de photo tous les 20°

- 4 : Prise de photo tous les 30°

- 5 : Prise de photo tous les 90°

- 0 : le bouton 0 permet de définir la position 0° à la position actuelle du plateau de la table.

Guide :

Le bouton *Guide* permet de lancer le mode *Go To.* Le mode *Go to* permet de déplacer le plateau de la table à un angle précis. La valeur est en degré et doit être entrée sur trois chiffres. Le bouton *Back* permet de corriger, le bouton *exit* de sortir du menu ou du mode.

View :

Le bouton *View* permet de lancer la remise à zéro de la position du plateau. Le plateau va tourner jusqu'à détecter l'aimant. (Cette action peut prendre du temps)

Flèche haut :

Le bouton *flèche vers le haut* permet de faire avancer le plateau d'un degré dans le sens horlogique

Flèche bas :

Le bouton *flèche vers le haut* permet de faire reculer le plateau d'un degré dans le sens anti horlogique

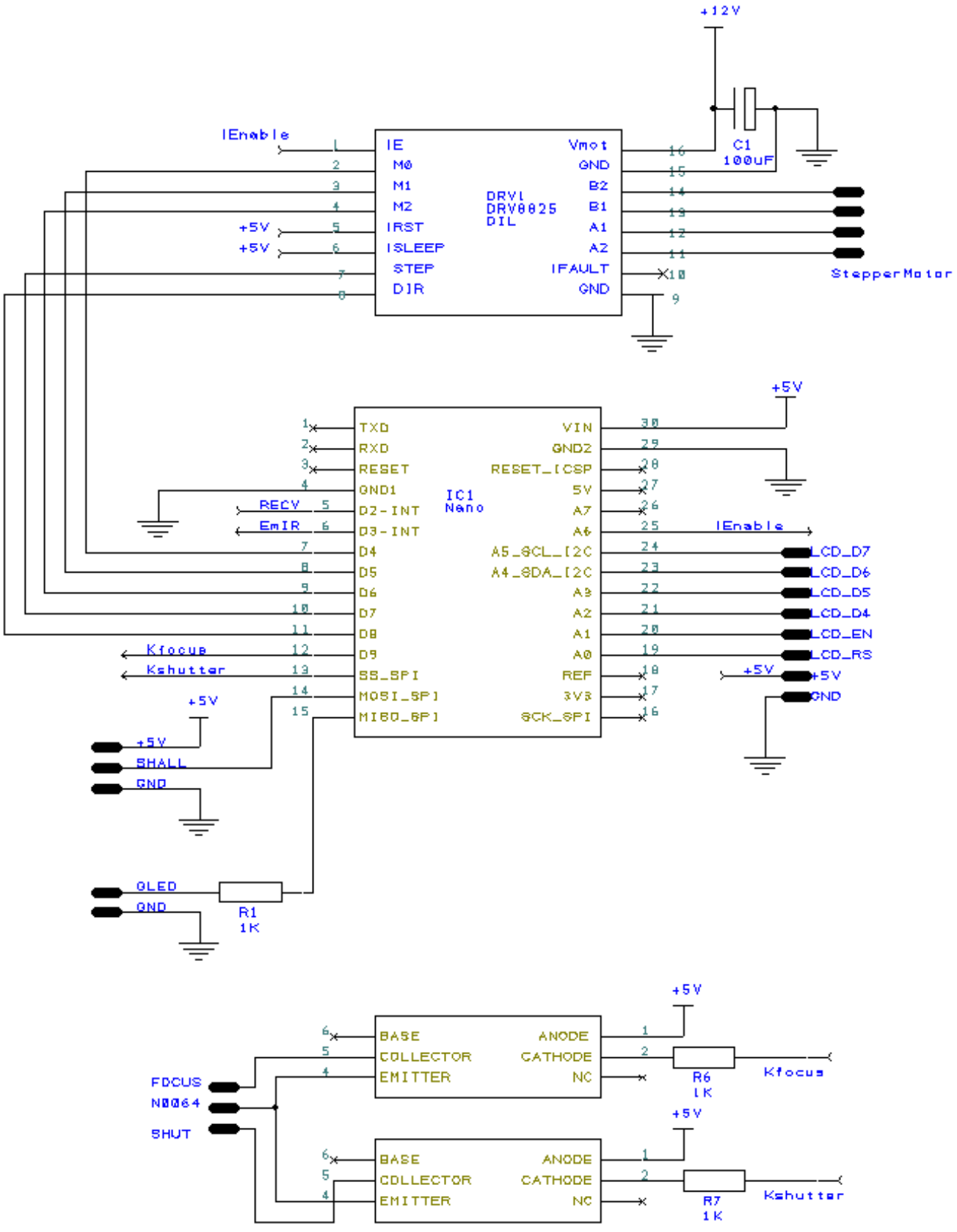
Back et Exit :

Les boutons *back* et *exit* permettent de corriger ou sortir du menu lors de la saisie de valeurs. Le bouton *exit* permet en outre, de stopper les modes *Go to* et *Run*

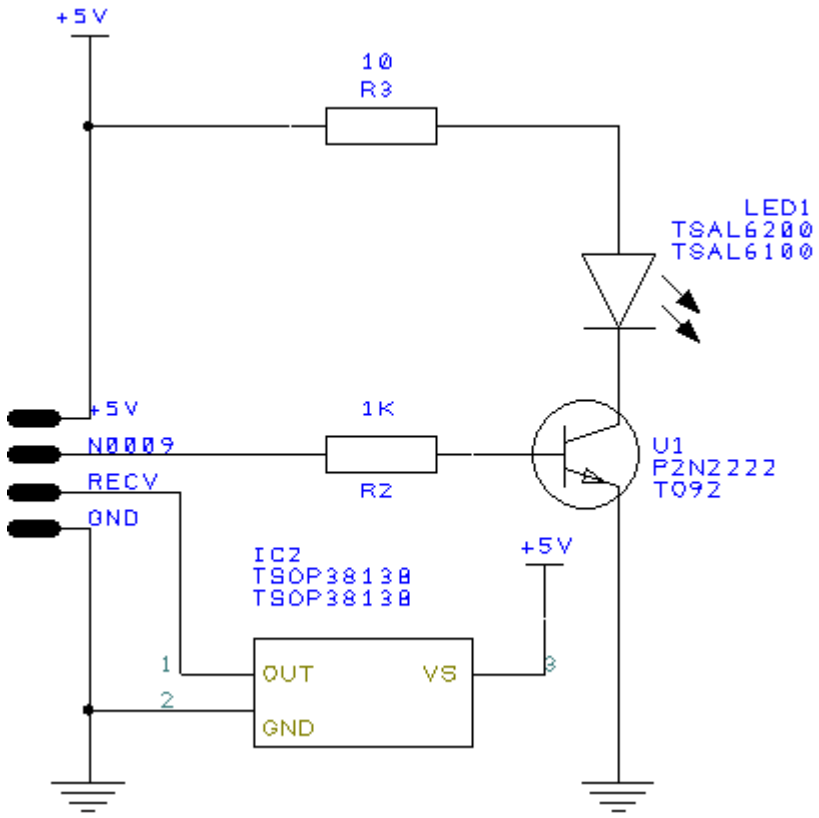
Table de correspondance des signaux infrarouge

Télécommande LG				
Bouton	**Code Hex**	**Code Dec**		**Fonction**
1	20DF8877	551520375		Nombre/ runSnap(10)
2	20DF48B7	551504055		Nombre/ runSnap(15)
3	20DFC837	551536695		Nombre/ runSnap(20)
4	20DF28D7	551495895		Nombre/ runSnap(30)
5	20DFA857	551528535		Nombre/ runSnap(90)
6	20DF6897	551512215		Nombre
7	20DFE817	551544855		Nombre
8	20DF18E7	551491815		Nombre
9	20DF9867	551524455		Nombre
0	20DF08F7	551487735		Nombre/ 0 relatif
Guide		551540010		Go to (position)
Q.view		551508135		initPos ()
info		551507370		Etat
input		551538735		réglage expTime
menu		551535165		réglage Accelération
text		551509665		
Vol +	20DF40BF	551502015		Augmenter la vitesse
Vol-	20DFC03F	551534655		Diminuer la vitesse
Prog +	20DF00FF	551485695		
Prog-	20DF807F	551518335		
↑	20DF02FD	551486205		Avancer de 1°
↓	20DF827D	551518845		Reculer de 1°
←	20DFE01F	551542815		
→	20DF609F	551510175		
Ok	20DF22DD	551494365		valider
Exit		551541285		sortir du menu
OFF		551489775		En/disable Moteur
Play		551489010		
Pause		551509410		Pause
Stop		551521650		Reset
Back		551490795		Retour arrière
Home		551501505		

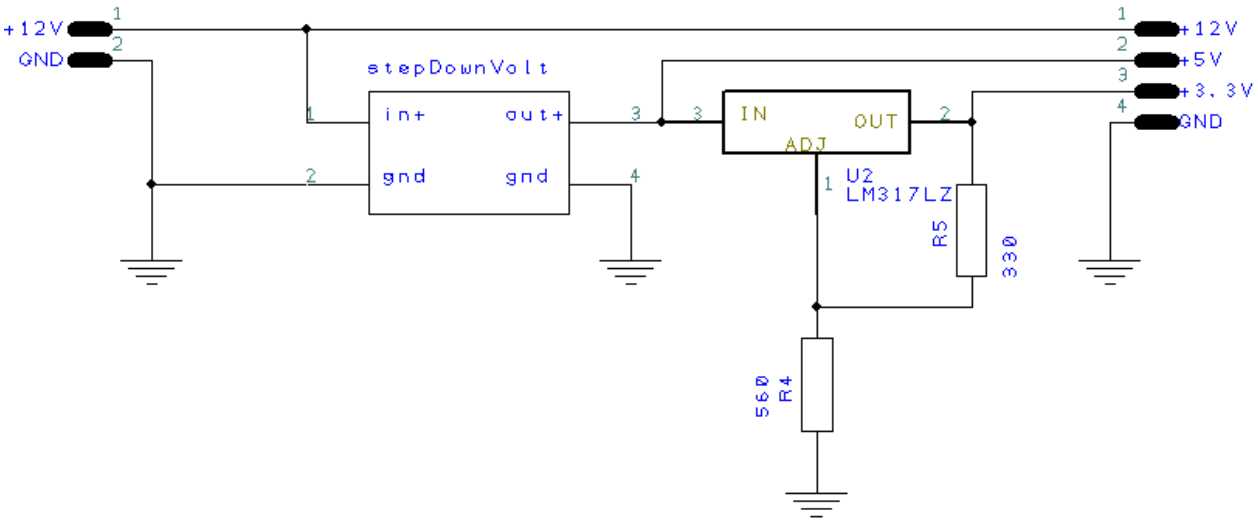
Schéma électronique:



Module infrarouge:



Alimentation:



Bibliographie

Ouvrage:
Le Langage C Norme ANSI Edition Dunod -Kernighan, Ritchie
Consulté le 2018-08-10

La Programmation Orienté Object - Bersini
Consulté le 2018-03-19

Site internet:

Arduino - EEPROMWrite [en ligne]
Disponible sur le World Wide Web : https://www.arduino.cc/en/Tutorial/EEPROMWrite
Consulté le 2018-04-04 16:46:21.

Arduino - PortManipulation [en ligne]
Disponible sur le World Wide Web :
https://www.arduino.cc/en/Reference/PortManipulation Consulté le 2018-04-09 18:14:06

Arduino Playground - Timer1 [en ligne]
Disponible sur le World Wide Web: https://playground.arduino.cc/Code/Timer1
Consulté le 2018-04-10 14:59:35.

Arduino Reference [en ligne].
Disponible sur le World Wide Web :
https://www.arduino.cc/reference/en/language/functions/math/max/
Consulté le 2018-03-20 18:02:45

Arduino Timer Interrupts [en ligne]
Disponible sur le World Wide Web : http://www.instructables.com/id/Arduino-Timer-Interrupts/
Consulté le 2018-04-10 15:16:47

Arduino Nano As Attiny 85 Programmer and 5 LED POV [en ligne]
Disponible sur le World Wide Web : http://www.instructables.com/id/Arduino-Nano-as-Attiny-85-programmer-and-5-LED-POV/
Consulté le 2018-04-16 10:43:06

Attiny 85 : Programmation avec un Arduino [en ligne]
Disponible sur le World Wide Web :
https://phmarduino.wordpress.com/2015/12/11/programmer-un-attiny-85/
Consulté le 2018-04-16 11:48:49

Exploration des "sleep mode" du attiny85 [en ligne]
Disponible sur le World Wide Web: https://enavarro.me/exploration-des-sleep-mode-du-attiny85.html
Consulté le 2018-04-26 15:39:03

Utiliser l'infrarouge avec une carte Arduino | WikiGeaks [en ligne].
Disponible sur le World Wide Web : http://notepad.xavierdetourbet.com/arduino-infrarouge/
Consulté le 2018-04-16 14:18:58

Driver de moteur pas-à-pas DRV8825 2133 [en ligne]
Disponible sur le World Wide Web : https://www.gotronic.fr/art-driver-de-moteur-pas-a-pas-drv8825-2133-22273.htm
Consulté le 2018-08-29 19:32:01

Moteur 42STH38-100 [en ligne]
Disponible sur le World Wide Web : https://www.gotronic.fr/art-moteur-42sth38-100-18839.htm
Consulté le 2018-03-19 21:36:33

Introduction aux algorigrammes [en ligne]
Disponible sur le World Wide Web: http://troumad.developpez.com/C/algorigrammes/
Consulté le 2018-05-14 12:29:11

Organigramme de programmation [en ligne].
Disponible sur le World Wide Web :
https://fr.wikipedia.org/w/index.php?title=Organigramme_de_programmation&oldid=148010080
Consulté le 2018-05-14 12:29:25

Arduino Nano [en ligne]
Disponible sur le World Wide Web : https://www.gotronic.fr/art-driver-de-moteur-pas-a-pas-drv8825-2133-22273.htm
Consulté le 2018-08-29 19:32:01

Git - Rudiments de Git [en ligne]
Disponible sur le World Wide Web : https://git-scm.com/book/fr/v1/D%C3%A9marrage-rapide-Rudiments-de-Git
Consulté le 2018-06-04 22:17:17

Git, les principes de base pour le prendre en main. adopteungit [en ligne]
Disponible sur le World Wide Web : http://adopteungit.fr/methodologie/2017/07/02/git-les-principes-de-base-pour-le-prendre-en-main.html
Mis à jours le 02 juillet 20167- Consulté le 04/06/2018.

AccelStepper: AccelStepper library for Arduino [en ligne]
Disponible sur le World Wide Web:
http://www.airspayce.com/mikem/arduino/AccelStepper/
Consulté le 2018-07-05 16:22:38

Comment éliminer le Backlash? [en ligne]
Disponible sur le World Wide Web: http://ohmyfab.fr/comment-eliminer-le-backlash/
Consulté le 2018-04-06 11:24:52

Comment déclencher un appareil photo automatiquement [en ligne]
Disponible sur le World Wide Web: http://www.yoctopuce.com/FR/article/comment-declencher-un-appareil-photo-automatiquement
Consulté le 2018-03-20 03:05:04

Articles:

La photogrammétrie numérique combinée avec la modélisation 3D: applications aux sciences forensiques - Lanzi Lorenzo[en ligne]
Disponible sur le World Wide Web :
https://www.unil.ch/files/live/sites/esc/files/shared/These_Lanzi.pdf
Consulté le 2018-05-14