3AU

ISAT – EPHEC

2020-2021

Projet intégré
Kart automatisé

Fasquel Sacha, Hemeleers Emile, Alvarez Lopez Juan,
Grabenweger Olivier et Malewo Kanda

A. Table des matières

A. Cahier des charges

Notre projet consistera à la réalisation d'un véhicule électrique (usage non routier) doté de capteurs. Cela nécessitera une gestion électronique, de l'automatisation et de la régulation via des composants dédiés ainsi que par des microcontrôleurs. Une personne à bord du véhicule pourra le manœuvre dans le sens avant et arrière de la marche et pourra tourner grâce à un volant. Un tableau de bord sera écran intégré pour afficher vitesse, niveau de charges des batteries, voyant témoins des phares, voyant témoins d'introduction clé.

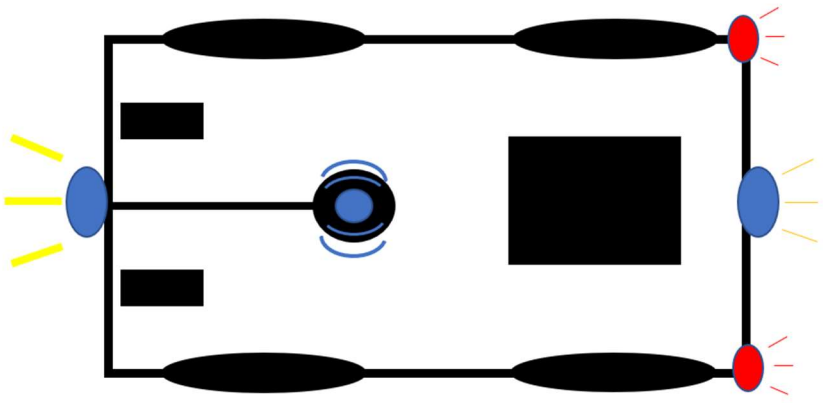
Objectifs :

Notre objectif, faire avancer le kart et établir la communication entre diffèrent élément de gestion

B. Schémas, plans et maquettes

Eclairage

Nous avons sur le kart un phare avant, un phare arrière et deux clignotants à l'arrière.
Les phares avant et arrière sont commandé via des transistors S8050 de deux façons différentes. Soit on peut les allumer/éteindre via un bouton sur le HMI soit si la luminosité devient basse, ils s'allument automatiquement. Et se rééteignent lorsque la luminosité redevient assez haute.
Les clignotants, piloté par des transistors, sont pour l'instant seulement activé lorsque la pédale de frein est actionnée.



Klaxon

Alimenté en 12v et piloté grâce à un transistor lui-même piloté par le microcontrôleur.

Electronique

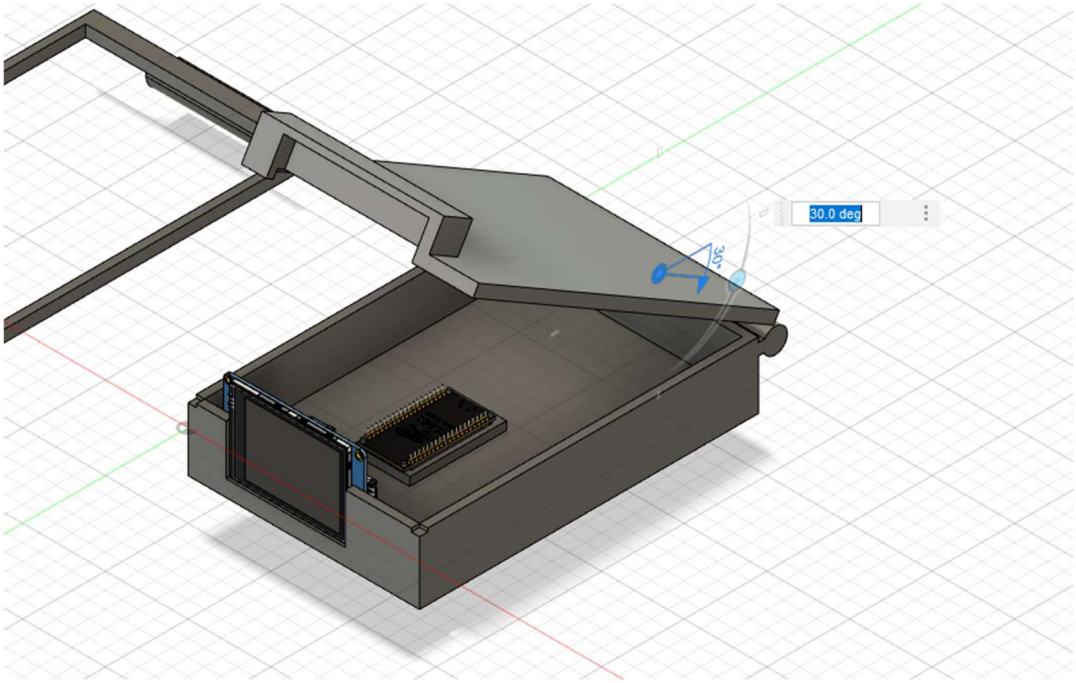
Cette partie se compose de deux ESP qui communiquent entre eux via le protocole CAN grâce à deux modules mcp2515 CANSPI.

Module 1 gestion tableau de bord (ESP32)

Situé dans le tableau de bord à l'avant du véhicule.

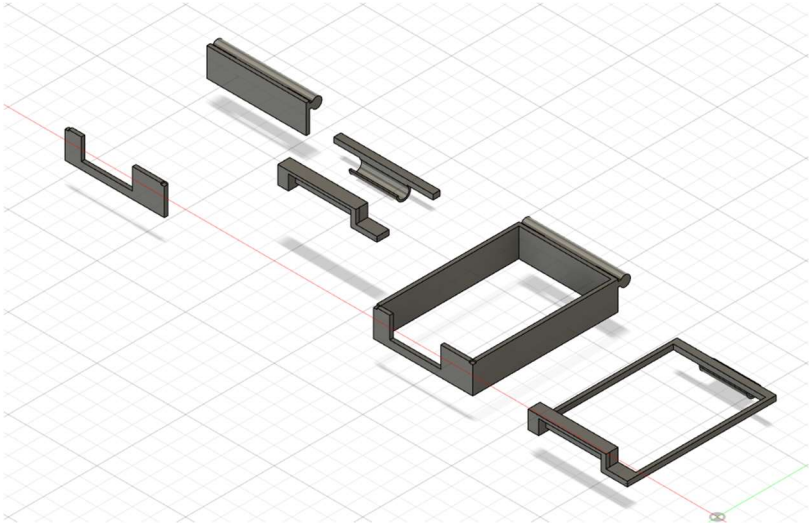
- Ce module sera relié avec un écran intelligent (Utilisation du Nextion 3.5 pouces)
- Il sera le module principal de tout notre kart.
- Capteur de vitesse (effet Hall), capteur de pression, clé RFID, boutons, PCB avec transistors pour l'éclairage, klaxon.
- Communication CAN avec le module 2
- …

Tout d'abord il convient de présenter notre « tableau de bord », en effet étant donné que le kart est susceptible d'être exposé à la pluie, il convenait de protéger les éléments électroniques. Pour ce faire nous avons imprimé en 3D le boitier suivant :



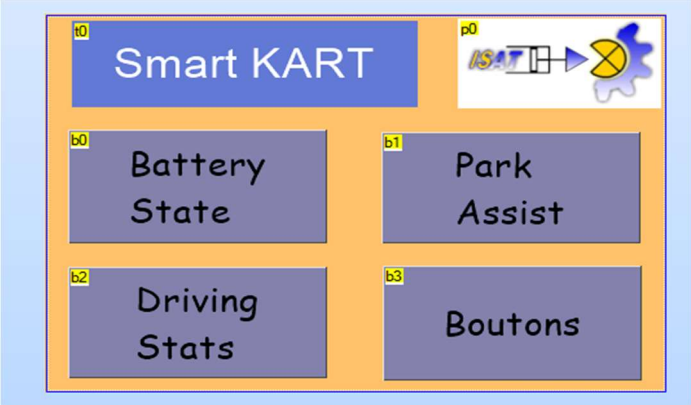
3AU

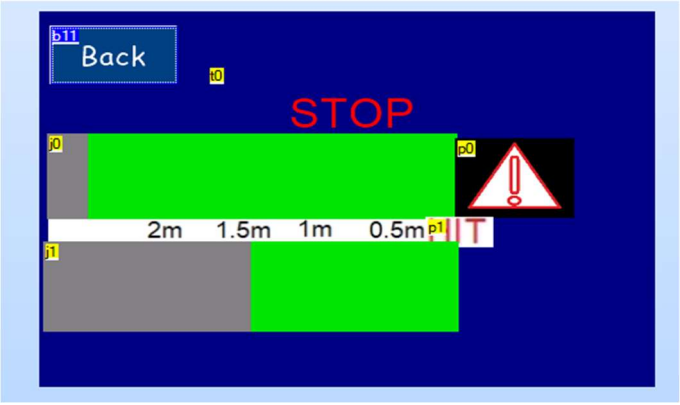
Dont voici, les différentes pièces.

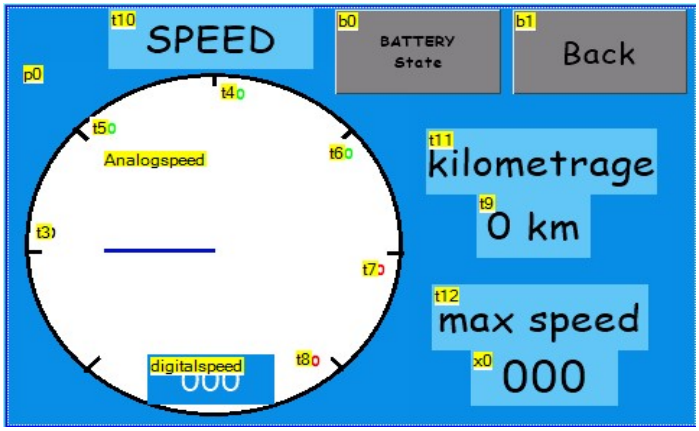


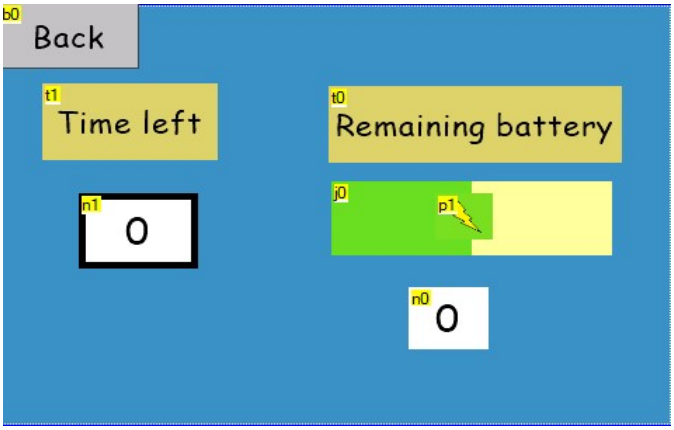
3AU

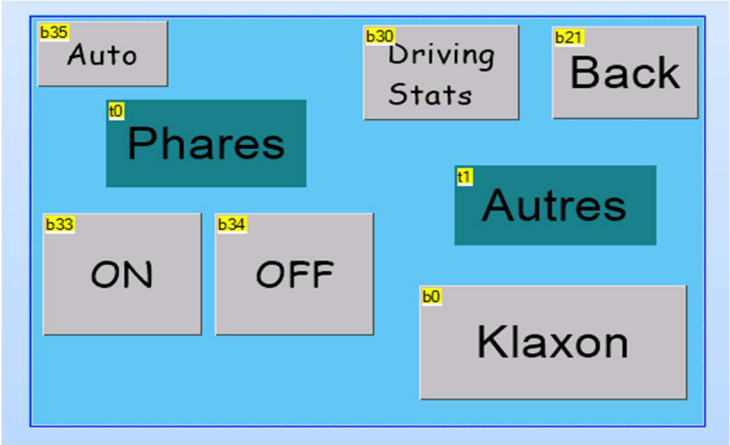
Voici quelques images de notre Nextion.











La communication avec le nextion est une « simple » communication série, en effet même si le protocole utilisé est relativement simple, l'interaction reste assez délicate. En effet, envoyer des données vers un objet non affiché cause des problèmes, nous avons dès lors du ruser pour structurer notre code se sorte que les données ne soient envoyées que quand la bonne page était affichée. Pour ce faire nous avons exploité la bibliothèque nextion.h mais les fonctions d'écriture de celle-ci étaient fort complexes alors que l'écriture de données est assez simple. Le code comporte donc des fonctions écrites par nous-mêmes pour l'écriture et une gestion d'évènements asynchrones pour la réception.

```
// Format: <type of object> <object name> = <type of object>(<page i
NexButton b0 = NexButton(4, 2, "b0");  // Button added
NexButton b2 = NexButton(0, 4, "b2");  // Button added
NexButton b1 = NexButton(0, 3, "b1");  // Button added
NexButton b25 = NexButton(3, 17, "b25");  // Button added
NexButton b10 = NexButton(3, 15, "b10");  // Button added
NexButton b16 = NexButton(3, 4, "b16");  // Button added
NexButton b12 = NexButton(1, 1, "b12");  // Button added
NexButton b11 = NexButton(2, 3, "b11");  // Button added
NexButton b21 = NexButton(4, 5, "b21");  // Button added
NexButton b30 = NexButton(4, 4, "b30");  // Button added
NexButton b33 = NexButton(4, 6, "b33");  // Button added
NexButton b34 = NexButton(4, 7, "b34");  // Button added
NexButton b35 = NexButton(4, 8, "b35");  // Button added
```

```
NexTouch *nex_listen_list[] =
{
  &b0,  // Button added
  &b1,  // Button added
  &b2,  // Button added
  &b25,  // Button added
  &b10,  // Button added
  &b16,  // Button added
  &b12,  // Button added
  &b11,  // Button added
  &b21,  // Button added
  &b30,  // Button added
  &b33,  // Button added
  &b34,  // Button added
  &b35,  // Button added
```

```
} // End of press event
void b0PopCallback(void *ptr)  // Press event for button b0…
void b2PushCallback(void *ptr)  // Press event for button b2…
void b21PushCallback(void *ptr)  // Press event for button b2…
void b30PushCallback(void *ptr)  // Press event for button b2…
void b25PushCallback(void *ptr)  // Press event for button b2…
void b10PushCallback(void *ptr)  // Press event for button b2…
void b16PushCallback(void *ptr)  // Press event for button b2…
void b12PushCallback(void *ptr)  // Press event for button b2…
void b11PushCallback(void *ptr)  // Press event for button b2
{
  messageCANEmission.data[1]=0;
  CurrentPage = 0;  // Set variable as 3 so from now on arduino knows
} // End of press event
void b33PushCallback(void *ptr)  // Release event for dual state butto
void b34PushCallback(void *ptr)  // Release event for dual state butto
void b35PushCallback(void *ptr)  // Release event for dual state butto
```

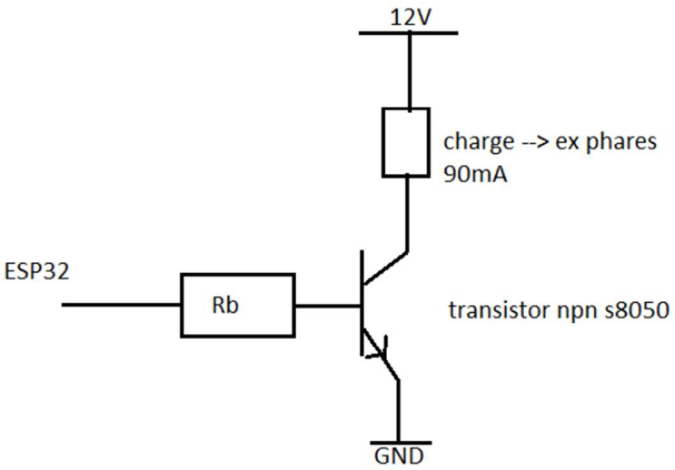
```
// Format for release events: <obj
b0.attachPush(b0PushCallback);  //
b0.attachPop(b0PopCallback);  // B
b1.attachPush(b1PushCallback);  //
b2.attachPush(b2PushCallback);  //
b12.attachPush(b12PushCallback);
b11.attachPush(b11PushCallback);
b25.attachPush(b25PushCallback);
b10.attachPush(b10PushCallback);
b16.attachPush(b16PushCallback);
b21.attachPush(b21PushCallback);
b30.attachPush(b30PushCallback);
b33.attachPush(b33PushCallback);
b34.attachPush(b34PushCallback);
b35.attachPush(b35PushCallback);
```

Comme vous pouvez le constater la gestion des évènements du nextion c'est avérée assez délicate alors que l'écriture se résume à la fonction suivante.

```
void hmiprint(char cmd[32]){
  Serial.print((String)cmd);
  Serial.write(0xff);
  Serial.write(0xff);
  Serial.write(0xff);
}
```

On peut également apercevoir sur une des images que l'envoi de données via le can s'effectue entre autres à travers les fonctions de callback associées aux différents événements du nextion.

Dimensionnement des transistor



Ib = Ic/Hf
Hf = 50
Ib = 0.09/50 = 0.0018

Urb = 3.3 - 0.7 = 2.6V

--> Ibmin = 0.0018

Ibsat = 2*Ibmin = 0.0036

Rb = Urb/Ib = 2.6/0.0018
= 720 ohm

Module 2 gestion des capteurs et actionneurs arrière (EPS32)

Sur ce module se trouvent tous les composants (capteurs, …) situés à l'arrière du kart, en effet parce qu'il était peu pratique de tirer des câbles pour chaque capteur nous avons mis en place une communication CAN.

Communication CAN

Le bus CAN (*Control Area Network*) est un moyen de communication série qui supporte des systèmes embarqués temps réel avec un haut niveau de fiabilité. Ses domaines d'application s'étendent des réseaux moyens débits aux réseaux de multiplexages faibles coûts. Il est avant tout à classer dans la catégorie des réseaux de terrain utilisé dans l'industrie. La structure du protocole du bus CAN possède implicitement les principales propriétés suivantes :

- Hiérarchisation des messages.
- Garantie des temps de latence.
- Souplesse de configuration.
- Détections et signalisations d'erreurs.
- Retransmission automatique des messages altérés dès que le bus est de nouveau au repos.

- Distinction d'erreurs : d'ordre temporaire ou de non-fonctionnalité permanente au niveau d'un nœud, déconnexion automatique des nœuds défectueux.

La transmission des données est effectuée sur une paire filaire différentielle. La ligne est donc constituée de deux fils :

- CAN L (CAN LOW),
- CAN H (CAN HIGH).

Le CAN est un bus de terrain, soumis à des parasites importants. La transmission en paire différentielle permet donc de limiter ces problèmes.

Nous n'utilisons pas plus de 8 données transmises entre les deux cartes nous avons donc simplifié au maximum les trames néanmoins nous sommes conscients qu'il peut être très intéressant de différencier les différentes trames émises où reçues. Nous détaillerons en premier le protocole mis en place pour ensuite discuter des améliorations potentielles.
Nous avons défini des bytes précises sur lesquels nous communiquons des données, par exemple de l'arrière vers l'avant le byte 1 contiendra toujours l'état de la LDR, la photorésistance, afin d'allumer les phares.
De la même manière toutes les autres bytes peuvent être utilisés. Notons que nous sommes ici limités à un maximum de 8 données envoyées et reçues.
Un Protocole plus complet serait d'attribuer un caractère de début à chaque donnée ainsi nous pouvons envoyer autant de données que nous voulons, par exemple envoyer la valeur de la LDR comme ceci : « <1 » → ici le « < » fait office de marqueur pour indiquer que c'est la LDR qui est transmise, et nous pouvons même imaginer dans le cas d'un très grand nombre de données différents des marqueurs de plusieurs bytes. Néanmoins vu la simplicité des données transmises nous avons décidé de rester simple non sans avoir exploré les possibilités que nous avions.

Gestion des différents capteurs

Avant tout il est nécessaire de préciser que deux ESP32 sont installé dans le kart, un à l'avant et un à l'arrière. Ils communiquent entre eux et gèrent chacun différents équipements qu'on a implémenté dans le projet.

ESP32 Avant	ESP32 Arrière
HMI	LDR
Capteur Ultrason	Capteur Ultrason
Capteur Effet hall	Capteur Effet hall
Phare	Phare
Klaxon	2 clignotants

Notre kart a été doté de différents capteurs afin d'ajouter une dimension automatique. Grâce à ceux-ci, diverses données sont récoltées et traitées en fonction de leur utilité. Certains capteurs serviront à afficher des données sur l'HMI et d'autres à automatiser certaines fonctionnalités.

Le kart sera donc équipé de deux capteurs ultrasons, un à l'avant, un à l'arrière qui auront la même fonction. Ils serviront à éviter de se cogner contre des obstacles quand on essaye des se garer. Pour activer les détecteurs il faut activer le mode « parking » sur le HMI. Ceci fait, on pourra observer sur l'écran à quelle distance se situent les obstacles s'il y en a.
Les capteurs ultrasons utilisés ici sont des HC-SR04, des capteurs pas des plus précis mais dont le coût et la distance de détection convenait parfaitement à notre projet.

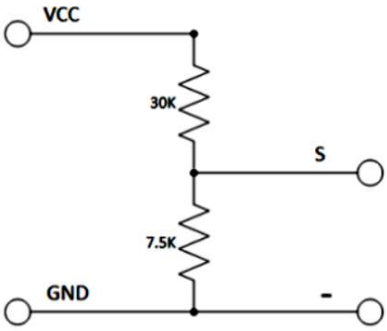
Electric Parameter

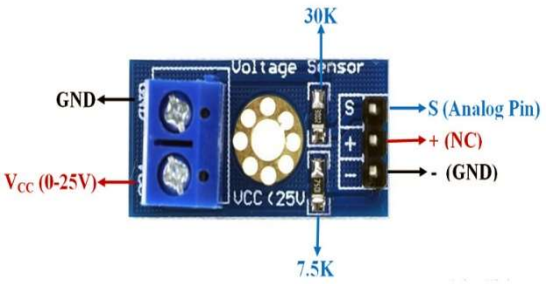
Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

Deux capteurs effet hall ont aussi été monté sur le véhicule et remplissent des objectifs bien différent. Le premier servira à détecter si la pédale de frein est enclenchée afin de transmettre un signal au microcontrôleur de l'arrière, ce qui allumera les clignoteurs arrière en continu sur l'intervalle de temps pour lequel le frein est actif. Cela sert à prévenir d'éventuel autres véhicules de se tenir à une certaine distance de notre kart afin de ne pas nous emboutir lors de la phase de freinage.
Le deuxième capteur effet hall sert à calculer la vitesse de notre véhicule. Ce capteur étant un TOR et se mettant à l'état haut uniquement lorsqu'on passe un aimant devant celui-ci, nous avons dû élaborer un calcul qui a directement été implémenté dans le programme de l'ESP32 situé à l'avant. Nous avons donc collé un aimant dans une des roues arrière et mis le capteur à une distance de 2 à 3 cm. La vitesse du kart sera dès lors relevé à partir d'un calcul qui se base sur la circonférence du cercle tracé par la course de l'aimant autour de l'axe de la roue et du temps qu'il met pour l'effectuer. La vitesse sera affichée sur le HMI.

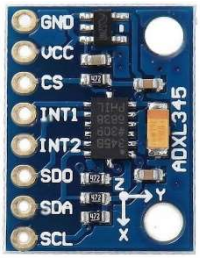
Comme mentionné un peu plus haut, une photorésistance va servir à allumer automatiquement les phares, à l'avant comme à l'arrière, lorsque le niveau de luminosité passe en dessous d'un certain seuil.

Il y a aussi un élément (qui ne peut pas s'appeler capteur) qui permet de mesurer la tension de la batterie. Il s'agit d'un pont diviseur de tension qui est sous la forme d'un module compose d'une résistance de 30kΩ et 7,5kΩ.





L'élément que nous avons rajouté est un accéléromètre 3 axes qui nous permet de calculer l'accélération et le déplacement du kart. Le tout sera affiché sur l'écran LCD.



Son fonctionnement est expliqué en détail dans la vidéo de présentation.

Code de l'ESP32 Arrière

```
#include <ACAN2515.h>
#include <HCSR04.h>

/*
 * Broches pour le chip select et l'interruption du MCP2515
 */

static const byte MCP2515_SCK  = 26 ; // SCK input of MCP2517
static const byte MCP2515_MOSI = 19 ; // SDI input of MCP2517
static const byte MCP2515_MISO = 18 ; // SDO output of MCP2517

static const byte MCP2515_CS  = 17; // CS input of MCP2515 (adapt to your design)
static const byte MCP2515_INT =  21 ; // INT output of MCP2515 (adapt to your design)

/*
 * L'objet pour piloter le MCP2515. SPI designe l'objet
 * utilise pour la connexion SPI car sur certaines cartes
 * notamment les Teensy, il peut y avoir plusieurs SPI.
 */
ACAN2515 controleurCAN(MCP2515_CS, SPI, MCP2515_INT);
```

```
21
22 /*
23  * La frequence du quartz du MCP2515 en hertz.
24  */
25 static const uint32_t FREQUENCE_DU_QUARTZ = 20UL * 1000UL * 1000UL ; // 20 MHz
26
27 /*
28  * La fréquence du bus CAN
29  */
30 static const uint32_t FREQUENCE_DU_BUS_CAN = 125ul * 1000ul;
31
32 /*
33  * Un objet pour le message CAN. Par defaut c'est un message
34  * standard avec l'identifiant 0 et aucun octet de donnees
35  */
36 CANMessage messageCANReception;
37 CANMessage messageCANEmission1;
38
39 // PIN des différente Capteurs/Actionneurs
40
41 const int LDR = 15;    //LDR ou photorésistance
42 const int hallPin = 22; //Capteur effet hall
43 const int pharearriere = 23;  //phare arriere
44 const int clignoteurG = 12; //clignoteur gauche
45 const int clignoteurD = 14;  //clignoteur droite
46 UltraSonicDistanceSensor distanceSensor(4, 5);    //PIN TRIGGER et ECHO du Ultrason
47
48 //Différentes variables
49 bool parking;
50 int lumi;
51 int sensorValue;
52 bool flag_sensor;
53 uint8_t stateLumi = 0;
54 bool oldstate;
55 int distance;
56
57 //Fonction pour allumer les clignoteurs si on freine (que le capteur ne detecte plus l'aimant)
58 void Halldetect()
59 {
60   if (sensorValue == LOW)
61   {
62     Serial.print("Halldetect");
63     digitalWrite(clignoteurG,sensorValue);
64     digitalWrite(clignoteurD,sensorValue);
65   }
66   else if (sensorValue == HIGH)
67   {
68     digitalWrite(clignoteurG,sensorValue);
69     digitalWrite(clignoteurD,sensorValue);
70   }
71 }
72
73 void setup()
74 {
75   //Initialise le Serial
76   Serial.begin(115200);
77   //Initialise le SPI
```

```
78   SPI.begin (MCP2515_SCK, MCP2515_MISO, MCP2515_MOSI) ;
79   Serial.println("Configuration du MCP2515");
80   //Fixe la vitesse du bus a 125 kbits/s
81   ACAN2515Settings reglages(FREQUENCE_DU_QUARTZ, FREQUENCE_DU_BUS_CAN);
82   // Demarre le CAN
83   const uint16_t codeErreur = controleurCAN.begin(reglages, [] { controleurCAN.isr(); } );
84   // Verifie que tout est ok
85   if (codeErreur == 0) {
86     Serial.println("Recepteur: configuration ok");
87   }
88   else {
89     Serial.println("Recepteur: Probleme de connexion");
90     while (1);
91   }
92
93
94   //Initialise les broche des LEDs
95   pinMode(pharearriere, OUTPUT);
96   digitalWrite(pharearriere,LOW);
97   pinMode(clignoteurD, OUTPUT);
98   digitalWrite(clignoteurD,LOW);
99   pinMode(clignoteurG, OUTPUT);
100  digitalWrite(clignoteurG,LOW);
101  //Configure la longueur de la trame d'envoi
102  messageCANEmission1.len = 3;
103  //met le flag à zero
104  stateLumi = 0;
105 }
106 //fonction loop
107 void loop()
108 {
109   //Reception de donnee du CAN
110   if (controleurCAN.receive(messageCANReception)) {
111     /* Un message CAN est arrive */
112     static uint32_t numero = 0;
113     controleurCAN.receive(messageCANReception) ;
114     Serial.println(messageCANReception.data[1]);
115     //on reçois l'etat de parking
116     parking = messageCANReception.data[1];
117   }
118   //lecture de l'etat du capteur effet hall
119   sensorValue = digitalRead(hallPin);
120   //fonction du capteur effet hall
121   Halldetect();
122   //SI parking passe à 1, on commence à envoyer des données de
123   //notre ultrason
124   if (parking == 1)
125   {
126     //envoi de la donnee de la distance entre nous et un obstacle
127     // via le capteur ultrasons
128     messageCANEmission1.data[1] = distance;
129     //vérification que le message a bien été envoyé
130     const bool ok = controleurCAN.tryToSend(messageCANEmission1);
131         if (ok)
132         {
133           Serial.print("message ");
134           Serial.print("Distance ");
```

```
135           Serial.println(" envoyee !");
136         }
137   }
138   //lecture de la valeur de la photoresistance
139   lumi = analogRead(LDR);
140   Serial.println(lumi);
141   /* Dans les lignes ci-dessous, on envoie notre message
142   seulement lorsqu'il y a un changement d'etat de notre
143   LDR, c'est à dire si la valeur de la luminosité passe
144   au dessus ou en dessous de 2000. Cela allege le
145   microcontrôleur qui ne dois donc pas envoyer des messages
146   en permanence    */
147   if (lumi < 2000)
148   {
149     stateLumi = 1;
150   }
151   else if (lumi >= 2000)
152   {
153     stateLumi = 0;
154   }
155   //si la luminosité est en dessous de 2000 et qu'elle
156   //était avant au dessus de 2000
157   if (stateLumi == 1 && oldstate == 0)
158   {
159     messageCANEmission1.data[0] = stateLumi;
160     const bool ok = controleurCAN.tryToSend(messageCANEmission1);
161       if (ok)
162       {
163         Serial.print("message ");
164         Serial.print("Allumer ");
165         Serial.println(" envoye !");
166       }
167     //on allume en meme temps les phare arriere
168     digitalWrite(pharearriere,HIGH);
169   }
170   //si la luminosité est au dessus de 2000 et
171   //qu'elle était avant en dessous de 2000
172   else if (stateLumi == 0 && oldstate == 1)
173   {
174     messageCANEmission1.data[0] = stateLumi;
175     const bool ok = controleurCAN.tryToSend(messageCANEmission1);
176       if (ok)
177       {
178         Serial.print("message ");
179         Serial.print("Eteindre ");
180         Serial.println(" envoye !");
181       }
182     //on eteind en meme temps les phare arriere
183     digitalWrite(pharearriere,LOW);
184   }
185   //on sauvegarde la le flag precedent
186   oldstate = stateLumi;
187   delay(1000);
188 }
```

**Pour la communication CAN entre les deux ESP32, nous utilisons la bibliothèques ACAN de Pierre Molinaro.*

Dans le code ci-dessus, il y a plusieurs points intéressants à soulever.
D'abord, pour déclarer les PIN dédier au mcp2515 nous avons dû un peu chipoter car bien qu'il y ait des PIN dédiées sur l'ESP32 pour le SPI, nous avons rencontré quelques soucis qui empêchais le bon fonctionnement du programme. Nous avons donc modifié ces PIN par défaut pour des PIN qui ne posait pas de problème.

Code ESP avant

```
1  // CODE CARTE AV ==> HMI + CPT
2
3  #include <HCSR04.h>
4  #include <ACAN2515.h>
5  #include <Arduino.h>
6  #include <Nextion.h>
7  #include <SPI.h>
8
9  #define Front_light 4
10 #define HALL_PIN 23
11 #define Klaxon 16
12 #define ultraS_pin 25
13 #define ANALOG_PIN 34
14
15 UltraSonicDistanceSensor distanceSensor(32, 33);  // Initialize
16 sensor that uses digital pins 13 and 12.
17
18
19 static const byte MCP2515_SCK  = 26 ; // SCK input of MCP2517
20 static const byte MCP2515_MOSI = 19 ; // SDI input of MCP2517
21 static const byte MCP2515_MISO = 18 ; // SDO output of MCP2517
22
23 static const byte MCP2515_CS  = 17 ; // CS input of MCP2515
24 (adapt to your design)
25 static const byte MCP2515_INT =  21 ; // INT output of MCP2515
26 (adapt to your design)
27
28 ACAN2515 controleurCAN(MCP2515_CS, SPI, MCP2515_INT);
29 static const uint32_t FREQUENCE_DU_QUARTZ = 20ul * 1000ul *
30 1000ul;
31 static const uint32_t FREQUENCE_DU_BUS_CAN = 125ul * 1000ul;
32
33 CANMessage messageCANEmission;
34 CANMessage messageCANReception1;
35
36
37 // Declare objects that we are going to read from the display.
38 This includes buttons, sliders, text boxes, etc:
39 // Format: <type of object> <object name> = <type of
40 object>(<page id>, <object id>, "<object name>");
41 NexButton b0 = NexButton(4, 2, "b0");  // Button added
42 NexButton b2 = NexButton(0, 4, "b2");  // Button added
```

```
43 NexButton b1 = NexButton(0, 3, "b1");  // Button added
44 NexButton b25 = NexButton(3, 17, "b25");  // Button added
45 NexButton b10 = NexButton(3, 15, "b10");  // Button added
46 NexButton b16 = NexButton(3, 4, "b16");  // Button added
47 NexButton b12 = NexButton(1, 1, "b12");  // Button added
48 NexButton b11 = NexButton(2, 3, "b11");  // Button added
49 NexButton b21 = NexButton(4, 5, "b21");  // Button added
50 NexButton b30 = NexButton(4, 4, "b30");  // Button added
51 NexButton b33 = NexButton(4, 6, "b33");  // Button added
52 NexButton b34 = NexButton(4, 7, "b34");  // Button added
53 NexButton b35 = NexButton(4, 8, "b35");  // Button added
54
55
56 char buffer[100] = {0};
57 int CurrentPage = 0;  // Create a variable to store which page
58 is currently loaded
59
60 NexTouch *nex_listen_list[] =
61 {
62   &b0,  // Button added
63   &b1,  // Button added
64   &b2,  // Button added
65   &b25,  // Button added
66   &b10,  // Button added
67   &b16,  // Button added
68   &b12,  // Button added
69   &b11,  // Button added
70   &b21,  // Button added
71   &b30,  // Button added
72   &b33,  // Button added
73   &b34,  // Button added
74   &b35,  // Button added
75
76   NULL  // String terminated
77 };  // End of touch event list
78
79 long tm1,tm2,tm3,tm_v;
80 int vit;
81 bool etat=false;
82 bool mvt;
83
84 //int index;
85 char command_line[32];
86 bool isCommandReady;
87
88 int vitesse,max_speed;
89 char serialbuffer[32];
90 char serialbuffer2[32];
91 char recptbuffer[8];
92 int cst,cst1,cst2;
93 int phareState, distance2;
94
95 int distance;
96
97 int sensorValue;
```

```
 98 bool flag_button,autoLight;
 99
100 void calcul_v(){
101   // lecture du capteur a Effet Hall
102   sensorValue = digitalRead( HALL_PIN );
103   // senseurValue = HIGH sans aimant
104   // senseurValue = LOW  quand POLE SUD aimant
105   sensorValue = not( sensorValue );
106   if (sensorValue == HIGH)
107   {
108     flag_button = 1;
109   }
110   if (flag_button == 1 && sensorValue == LOW)
111   {
112     flag_button = 0;
113     tm2 = millis();
114     tm_v = tm2 - tm3;
115     vit =(0.0007536 / tm_v)*3600000;   // avec 12cm de rayon
116     tm3 = tm2;
117   }
118 }
119
120 void b0PushCallback(void *ptr)  // Press event for button b0
121 {
122   digitalWrite(Front_light,HIGH);
123   //Serial.print("klaxon");
124   //Serial.print("/n");
125 }  // End of press event
126 void b1PushCallback(void *ptr)  // Press event for button b0
127 {
128   CurrentPage = 2;
129   messageCANEmission.data[1]=1;
130 }  // End of press event
131 void b0PopCallback(void *ptr)  // Press event for button b0
132 {
133   digitalWrite(Front_light,LOW);
134   //Serial.print("klaxon");
135   //Serial.print("/n");
136 }  // End of press event
137 void b2PushCallback(void *ptr)  // Press event for button b2
138 {
139   CurrentPage = 3;  // Set variable as 3 so from now on arduino
140 knows page 3 is loaded on the display
141 }  // End of press event
142 void b21PushCallback(void *ptr)  // Press event for button b2
143 {
144   CurrentPage = 0;  // Set variable as 3 so from now on arduino
145 knows page 3 is loaded on the display
146 }  // End of press event
147 void b30PushCallback(void *ptr)  // Press event for button b2
148 {
149   CurrentPage = 3;  // Set variable as 3 so from now on arduino
150 knows page 3 is loaded on the display
151 }  // End of press event
152 void b25PushCallback(void *ptr)  // Press event for button b2
```

```
153 {
154   CurrentPage = 4;  // Set variable as 3 so from now on arduino
155 knows page 3 is loaded on the display
156 }  // End of press event
157 void b10PushCallback(void *ptr)  // Press event for button b2
158 {
159   CurrentPage = 0;  // Set variable as 3 so from now on arduino
160 knows page 3 is loaded on the display
161 }  // End of press event
162 void b16PushCallback(void *ptr)  // Press event for button b2
163 {
164   CurrentPage = 1;  // Set variable as 3 so from now on arduino
165 knows page 3 is loaded on the display
166 }  // End of press event
167 void b12PushCallback(void *ptr)  // Press event for button b2
168 {
169   CurrentPage = 0;  // Set variable as 3 so from now on arduino
170 knows page 3 is loaded on the display
171 }  // End of press event
172 void b11PushCallback(void *ptr)  // Press event for button b2
173 {
174   messageCANEmission.data[1]=0;
175   CurrentPage = 0;  // Set variable as 3 so from now on arduino
176 knows page 3 is loaded on the display
177 }  // End of press event
178 void b33PushCallback(void *ptr)  // Release event for dual state
179 button bt0
180 {
181   digitalWrite(Front_light,HIGH);
182
183   //Serial.print("Front LIGHT");
184   //Serial.print("/n");
185 }  // End of release event
186 void b34PushCallback(void *ptr)  // Release event for dual state
187 button bt0
188 {
189   digitalWrite(Front_light,LOW);
190
191   //Serial.print("Back LIGHT");
192   //Serial.print("/n");
193 }  // End of relea
194 void b35PushCallback(void *ptr)  // Release event for dual state
195 button bt0
196 {
197   autoLight = 1;
198   //Serial.print("Back LIGHT");
199   //Serial.print("/n");
200 }  // End of relea
201
202 ////////////////////////// End of touch events
203
204 void hmiprint(char cmd[32]){
205   Serial.print((String)cmd);
206   Serial.write(0xff);
207   Serial.write(0xff);
```

```
208   Serial.write(0xff);
209 }
210
211 void setup() {
212     // put your setup code here, to run once:
213   tm1=millis();
214   max_speed=0;
215   pinMode(Front_light,OUTPUT);
216   pinMode(Klaxon,OUTPUT);
217   Serial.begin(9600);
218
219   /* Demarre le SPI */
220   SPI.begin (MCP2515_SCK, MCP2515_MISO, MCP2515_MOSI) ;
221   /* Configure le MCP2515 */
222   Serial.println("Configuration du MCP2515");
223   /* Fixe la vitesse du bus a 125 kbits/s */
224   ACAN2515Settings reglages(FREQUENCE_DU_QUARTZ,
225 FREQUENCE_DU_BUS_CAN);
226   /* Demarre le CAN */
227   const uint16_t codeErreur = controleurCAN.begin(reglages, [] {
228 controleurCAN.isr(); } );
229   /* Verifie que tout est ok */
230   if (codeErreur == 0) {
231     Serial.println("Configuration ok");
232   }
233   else {
234     Serial.println("Probleme de connexion");
235     //while (1);
236   }
237
238   messageCANEmission.len = 2;
239   //messageCANReception1.len = 1;
240
241   // Register the event callback functions of each touch event:
242   // You need to register press events and release events
243 seperatly.
244   // Format for press events: <object name>.attachPush(<object
245 name>PushCallback);
246   // Format for release events: <object name>.attachPop(<object
247 name>PopCallback);
248   b0.attachPush(b0PushCallback);  // Button press
249   b0.attachPop(b0PopCallback);  // Button press
250   b1.attachPush(b1PushCallback);  // Button press
251   b2.attachPush(b2PushCallback);  // Button press
252   b12.attachPush(b12PushCallback);
253   b11.attachPush(b11PushCallback);
254   b25.attachPush(b25PushCallback);
255   b10.attachPush(b10PushCallback);
256   b16.attachPush(b16PushCallback);
257   b21.attachPush(b21PushCallback);
258   b30.attachPush(b30PushCallback);
259   b33.attachPush(b33PushCallback);  // Dual state button bt0
260 release
261   b34.attachPush(b34PushCallback);  // Dual state button bt0
262 release
```

```
263   b35.attachPush(b35PushCallback);  // Dual state button bt0
264 release
265
266   // End of registering the event callback functions
267 }
268
269 void loop() {
270   nexLoop(nex_listen_list);  // Check for any touch event
271   if(CurrentPage == 0){  // If the display is on page 0, do the
272 following:
273
274   }
275   if(CurrentPage == 1){  // If the display is on page 1, do the
276 following:
277
278   }
279   if(CurrentPage == 2){  // If the display is on page 2, do the
280 following:
281     distance = 100-distanceSensor.measureDistanceCm();
282     memset(serialbuffer,0,sizeof(serialbuffer));
283     sprintf(serialbuffer,"j0.val=%d",distance);
284     hmiprint(serialbuffer);
285     memset(serialbuffer,0,sizeof(serialbuffer));
286     sprintf(serialbuffer,"j1.val=%d",distance2);
287     hmiprint(serialbuffer);
288     delay(500);
289   }
290   if(CurrentPage == 3){  // If the display is on page 2, do the
291 following:
292     if(tm1<millis()){
293
294       vitesse = analogRead(ANALOG_PIN);
295       cst = map(vitesse,0,4095,0,270);
296       cst1 = map(vitesse,0,4095,0,60);
297       memset(serialbuffer,0,sizeof(serialbuffer));
298       sprintf(serialbuffer,"digitalspeed.val=%d",cst1);
299       hmiprint(serialbuffer);
300
301       memset(serialbuffer,0,sizeof(serialbuffer));
302       sprintf(serialbuffer,"Analogspeed.val=%d",cst);
303       hmiprint(serialbuffer);
304       if(vitesse>max_speed){
305         max_speed=vitesse;
306         cst2 = map(vitesse,0,4095,0,60);
307       }
308       memset(serialbuffer,0,sizeof(serialbuffer));
309       sprintf(serialbuffer,"x0.val=%d",cst2);
310       hmiprint(serialbuffer);
311       tm1 = millis() + 500;
312
313
314     /*calcul_v();
315     cst = map(vit,0,60,0,270);
316
317     memset(serialbuffer,0,sizeof(serialbuffer));
```

```
318     sprintf(serialbuffer,"digitalspeed.val=%d",vit);
319     hmiprint(serialbuffer);

        memset(serialbuffer,0,sizeof(serialbuffer));
        sprintf(serialbuffer,"Analogspeed.val=%d",cst);
        hmiprint(serialbuffer);
        if(vit>max_speed){
          max_speed=vit;
        }
        memset(serialbuffer,0,sizeof(serialbuffer));
        sprintf(serialbuffer,"x0.val=%d",max_speed);
        hmiprint(serialbuffer);
        tm1 = millis() + 500;*/
        }
      }

      if(CurrentPage == 4){  // If the display is on page 4, do the
    following:

      }

      if (controleurCAN.receive(messageCANReception1)) {
        //Un message CAN est arrive
        controleurCAN.receive(messageCANReception1) ;
        phareState= messageCANReception1.data[0];
        distance2 = messageCANReception1.data[1];
        distance2 = 100-distance2;
        // l'etat de la LED correspondante est change
        if(autoLight){
          digitalWrite(Front_light,phareState);
        }
      }
    }
```

PCB

Pour la réalisation du PCB nous sommes passé par plusieurs schéma en fonction de nos décisions. Au début, nous avions émis l'idée d'utiliser des ESP8266. Mais nos idées

d'améliorations nécessitant de plus en plus de PIN, nous fument obliger de nous tourner vers une autre solution, l'ESP32, qui aura en plus comme avantage d'être plus puissant et permettant même le multitâche.

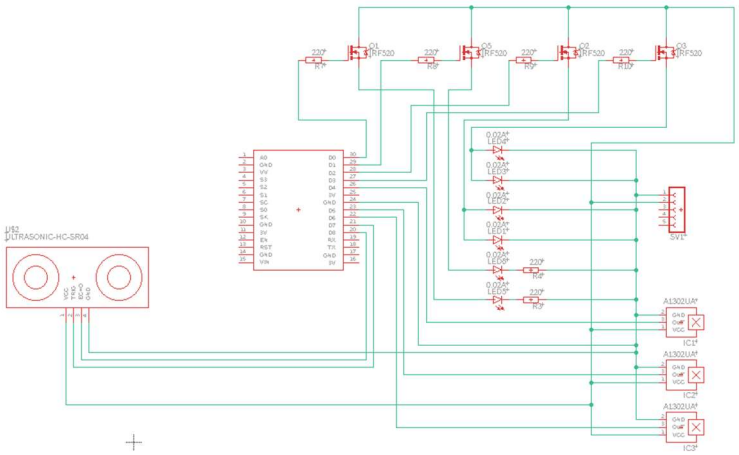


Figure 1: Premier JET avec l'ESP8266

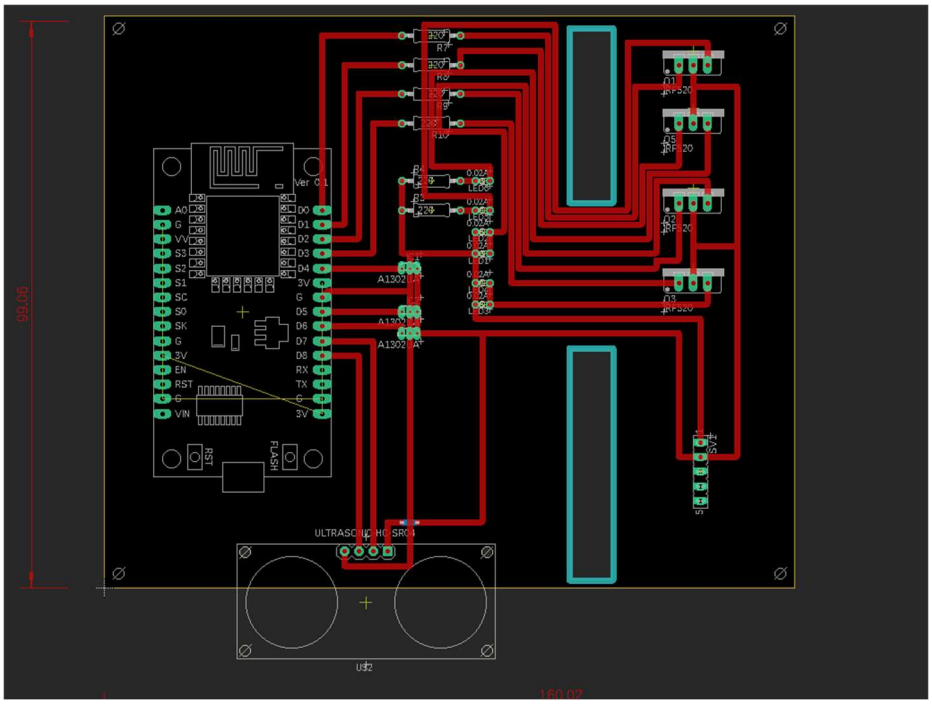


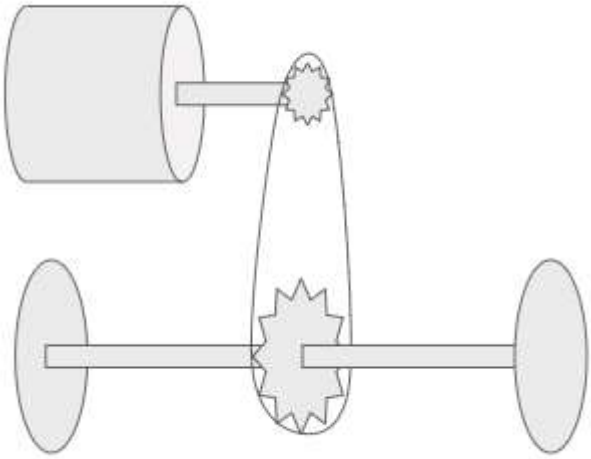
Figure 2: Board du premier JET

Déplacement mécanique

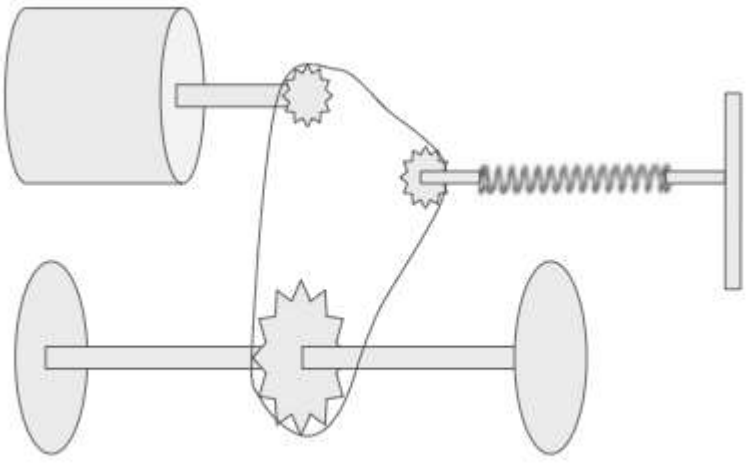
Mouvement :

3AU

Notre kart se meut à l'aide d'un moteur électrique. Ce moteur est un moteur à courant continu avec un tension maximale de 36 Volts et une puissance de 1000 Watts. La transmission entre l'axe du moteur et l'axe des roues arrière est établie par une chaîne. Voici un schéma de principe :



Nous nous sommes très vite rendu compte qu'il faudrait tendre la chaîne. Nous pensions la réduire de taille jusqu'à ce qu'elle soit tendue mais cette solution n'est pas réaliste pour 2 raisons. La première, est le fait qu'un maillon fait varier de 2 ou 3 cm la taille totale de la chaîne. Il est dès lors impossible d'ajuster la chaîne correctement car nous ne tombions pas sur une taille acceptable. La deuxième raison est le fait que tout notre système de transmission n'est pas totalement fixe : l'accélération induit un stress qui déforme temporairement les supports, les axes, (…), nous avons donc mis un tendeur de chaîne qui s'adapte en fonction du stress imposé. Voici le schéma de principe définitif de la transmission :



Le diamètre du pignon de l'axe des roues est plus grand que celui de l'axe du moteur. Ce choix est justifié par le fait que notre moteur DC à une vitesse de rotation qui, dans un monde idéal, donnerait 89 km/h (Le calcul a été fait à partir de la vitesse de rotation maximale du moteur et de la taille des roues du kart). Cette vitesse n'était pas dans notre cahier de charge, nous avons choisi les plateaux en fonction de cela. Malgré tout, quelques choix ont été empiriques : la conception mécanique n'étant pas dispensée dans notre cursus, nous avons fait des tests et nous avons choisi les composants qui fonctionnaient le mieux.

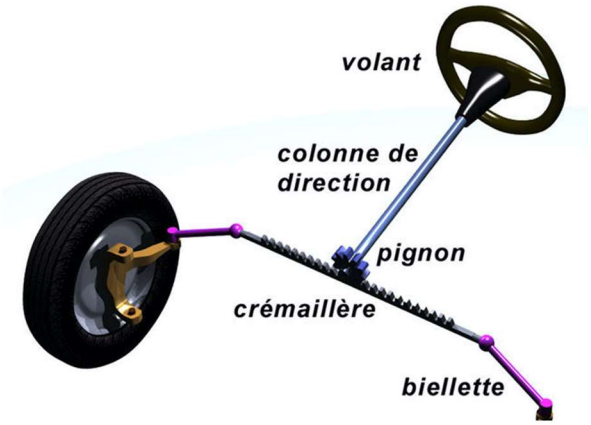
Notre Kart manque légèrement de couple. Nous avons sous-estimé le poids final : le kart, l'utilisateur, le moteur et tous les composants se rapprochent des 100 kilos… Il en résulte que le démarrage à l'arrêt est peu dynamique. Le démarrage en côte est lui impossible. Nous aurions dû mieux dimensionner les pignons mais nous avons récupéré de vieux vélos et n'avions pas tellement d'autres choix. De plus, la partie mécanique nous a pris beaucoup plus de temps que nous le pensions : nous avons fait le choix de continuer le projet en l'état car la partie mécanique n'est pas celle qui a motivé le choix de faire ce projet. Il ne faut pas oublier que nous sommes en automatique et pas en électromécanique.

Freinage :

Le freinage est fait de manière la plus classique qu'il soit : nous avons mis un câble qui va de la pédale de frein à l'arrière du véhicule. Lorsque l'utilisateur freine, le câble se serre. Ce câble est relié à un frein à disque qui freine l'axe arrière des roues. Lorsque l'utilisateur relâche le frein, un système de ressort, intégré dans le frein à disque repousse le câble dans sa position initiale. La vitesse du kart n'étant pas trop élevée, ce système de frein est suffisant pour l'arrêter en quelques secondes.

Direction :

Nous avons utilisé une crémaillère de direction pour faire tourner le kart. Le volant est fixé à un arbre de transmission. Au bout de cet arbre, se trouve une roue dentée. La roue dentée est sur la crémaillère, ce qui permet de transformer un mouvement circulaire en un mouvement rectiligne.
Le schéma ci-dessus permet de bien visualiser le principe de fonctionnement du type de crémaillère de direction que nous avons utilisé. Ce type de crémaillère est le plus simple à mettre en place. Beaucoup d'autres crémaillères se basent le même principe mais intègrent de l'électronique et/ou



l'hydraulique (Comme les directions assistées des voitures modernes).

Gestion de l'alimentation et sécurité

Le contrôleur moteur n'est pas directement lié aux batteries directement.
Nous avons mis en place un élément intermédiaire capable d'augmenter la tension des batteries (en parallèles) jusqu'à 36v. Cet élévateur a été scrupuleusement sélectionné pour admettre un courant jusque 27 ampères. Ce circuit de puissance est relié aux bornes bien solides avec des connecteurs spécifiques ainsi que du câblage au $6mm^2$. Cet élévateur de tension est composé d'un fusible de 40 ampères.

Composants

- MicroControleurs 2X
- Capteur ultrason HC04
- Capteur effet hall A3144
- Ecran Nextion 3.5 pouces
- Relais 5v
- Boutons
- DHT11
- RFID-RC522
- Photorésistance
- Élévateur de tension 30A 10-60V
- Élévateur de tension 3A
- Klaxon 12v
- Cables,ressort,...

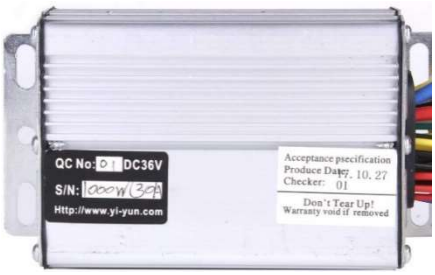
C. Electronique de puissance gestion moteur

L'objectif de l'électronique de puissance et de nous permettre de contrôler, alimenter et réguler le moteur à courant continue via plusieurs modules diffèrent qui vont être expliqué, avec un peu plus de détail, dans les paragraphes suivants.

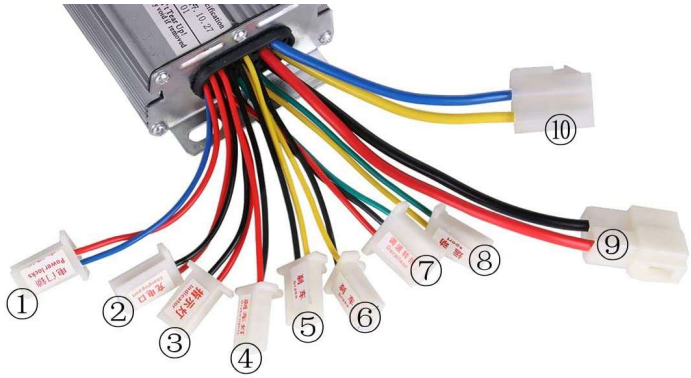
Contrôleur moteur DC

Le contrôleur de moteur à balais est un appareil électronique qui permet la régulation de la vitesse de rotation d'un moteur à balais par l'application d'une tension de 36VDC à l'appareil. Cet appareil s'alimente avec une tension de 36VDC pour des courants comprises entre 30 à 32A max et est compatible pour des moteurs d'une puissance de 1000W. Nous pouvons y brancher des batteries de voiture avec des charges 12 à 17AH.

3AU



L'appareil et composé de plusieurs ports d'entrées et sorties :



① Red & Blue (Small Cable): Key Switch(Power Lock): If there is no switch please connect red to blue.

② Red & Black (Small Cable): Charging Port, Red to +, Black to -

③ Red & Black (Small Cable): Indicator

④ Red & Black (Small Cable): Brake Light

⑤ ⑥ Yellow & Black (Small Cable): Brake, to front/ rear brake switch on brake levers

⑦ Red, Black & Green(Small Cable): To Speed Regulator, 1-5v Throttle. Red:+5v, Black: -, Green: Signal

⑧ Green & Yellow(Small Cable): Sports Model Switch

⑨ Red & Black (Thick Cable): To Battery, Red: +, Black: -

⑩ Blue & Yellow (Thick Cable): To Motor, Blue: +, Yellow: -

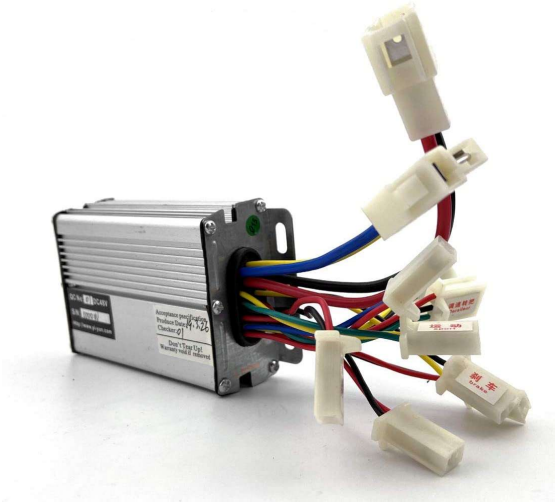
1. Port pour la connexion d'un interrupteur à clé (Power Lock)
2. Port de charge des batteries
    a. Rouge (+), Noir (-) ;
3. Indicator
4. Port pour les phares des freins
5. 1er Port de connexion pour le levier de frein
6. 2nd Port de connexion pour le levier de frein
7. Régulateur de vitesse, port de connexion pour pédale d'accélération
    a. Sortie 5V ;
    b. Rouge (+5V), Noir (-), Vert (Signal) ;
8. Sport Model switch
9. Port de connexion batterie 36VDC
10. Port de connexion moteur à Balais
    a. Bleu (+), Jaune (-) ;

Par la description de l'appareil, la connexion de différent module devient très simple et plus aisée. Nous y connecterons essentiellement une pédale d'accélération, des batteries en séries et un moteur de 1000W.

Pourquoi avoir choisi un contrôleur de moteur à balais ?

A la base, nous étions partis sur la conception d'un hacheur 4 quadrants composé de mosfet, pour des vitesses de commutation rapide et pour supporter des grands courants, contrôlé par un signal PWM d'un Arduino. Nous avons dû, hélas, comprendre, par le billet de l'un de nos professeurs, que cette idée n'était pas faisable à notre niveau dans le cursus scolaire. En effet, à partir d'un certain courant, les composants électroniques ne réagiraient pas comme nous le voudrions et que le hacheur ne se limiterait pas qu'à de simple mosfet mis en série et parallèle. Il aurait alors fallu penser à un moyen de réguler la tension, éviter les retours de courant, de moyen pour dissiper la chaleur et le courant, atténuer les bruits qui dans le circuit et un tas d'autres problèmes auxquels nous ne voulions certainement pas avoir à faire en vue du court temps que nous possédions pour développer ce type de solution pour le contrôle d'un moteur électrique. Il nous aurait fallu plus que 3 mois. Pour couronner le tout, le dimensionnement et la conception de ce type de module n'entre pas vraiment dans le cadre du cursus d'un automaticien.

Puis, nous avions cherché d'autre moyen de pour pouvoir contrôler notre moteur avec un module permettant de contrôler un moteur de 36V, 28A et 1000W. Il ne va sans dire que la recherche a été décevante surtout pour les 2 derniers paramètres où il nous a été impossible de trouver un module et peu coûteux pouvant supporter 28A et 1000W. Alors, nous nous sommes tournées vers une solution toute simple et qui était déjà disponible dès le départ et toute faite, le contrôleur de moteur à balais.



Au départ, elle devait juste nous servir à faire des tests pour faire avancer notre kart et voir comment nous allions disposer nos batteries pour développer le plus de couple possible au niveau du moteur et voir s'il nous faudrait rajouter des batteries supplémentaires. L'idée était, vraiment, d'en concevoir un nous-même.

Au final et suite aux problèmes énoncé, nous avons choisi de l'utiliser dans notre projet et il y a beaucoup d'avantage avec cet appareil. Toute la régulation électronique se fait grâce à cette petite boit très légèreté. Plus besoin de penser aux nombreux problèmes énumérés avant, l'appareil les gèrent. Il est doté d'une protection de la batterie, et si la tension baisse, l'unité de commande s'éteint, ne coute pas très chère et, en plus, l'installation des modules est simplifiée grâce aux différentes connexions.

Les caractéristiques du contrôleur en ont fait un choix de première puisque celui-ci permet parfaitement de réguler notre moteur électrique (qui sera développer dans le paragraphe suivant). Il accepte des tensions de 36V pour de grande charge allant de 30 à 32A et est compatible avec des moteurs développant des puissances de 1000W. Ce dernier, est adapté pour les quadriporteurs électriques, les scooters, les scooters 3 roues de ville, les scooters jouets, partout où on veut des performances élevées donc excellent pour notre kart.

Moteur DC

Le moteur DC est un moteur électrique à balais pour scooter électrique développant une puissance de 1000W avec 3000Tr/min pour une tension à ses bornes de 36VDC et un courant continu de 28A.

Il est l'un des acteurs principaux de notre projet intégré. Grace au moteur, nous pouvons mouvoir le véhicule à notre guise.

Le moteur étant le premier appareil sur lequel nous nous sommes penchés, c'est à partir de ce dernier que nous avons pu faire le choix des batteries, modules de contrôle et rapports de transmission avec les roues (un pignon était déjà compris avec le moteur).

Le branchement est très simple, nous branchant les 2 files du moteur au contrôler de moteur électrique et le tour est joué. Mais, il ne suffit pas de seulement brancher le moteur au contrôleur, il faut aussi pouvoir appliquer les bonnes tensions et les bons courants afin d'être sûr de pouvoir développer le puissant nécessaire pour pouvoir faire avancer le kart.

Au début du projet, nous avions utilisé 3 batteries 12V de 7ah chacune. Avec cette configuration, le moteur tournait vite mais manquait en couple car il avait du mal à tirer tout le poids du véhicule avec lui. Bien-sûr ceci n'était pas entièrement la faute des batteries mais

aussi de la transmission qui y joue beaucoup puisque nous l'avions modifiées par des chaines et roues dentée de vélo avec des rapport de transmission très peu élever. Bien-sûr, et nous l'avions compris, l'Idéal aurais été d'avoir des batteries qui puissent délivrer le courant nécessaire à notre moteur pour augmenter ce couple.

Afin d'augmenter le couple, il nous des batteries avec pouvant développer des courants plus importants. Grace aux formule suivante, nous comprenant l'utilité d'augmenter le courant :

- La force contre-électromotrice : $E = k' * B_s * \Omega$

  où Ω correspond à la vitesse de rotation, ou pulsation, en rad/s reliée à la fréquence (nombre de tours par seconde) par la relation : $f = \frac{\Omega}{2\pi}$

  B_s est le champ statorique

  Dans cette formule, on voit que la tension est proportionnelle à la vitesse de rotation du rotor donc ça veut dire qu'au plus on augmente la tension au plus on gagne en vitesse de rotation. Ce qui n'est pas requis dans ce cas pour faire avancer un kart avec du poids.

- Le couple électromécanique (moteur ou résistant) : $T = k' * B_s * I_i$

  Ici, on voit très bien que le courant lui est proportionnelle au couple électromécanique ce qui veut dire que pour pouvoir développer un couple suffisamment bon pour déplacer le kart, il nous faut augmenter le courant.

Batteries

Les batteries sont 3 batteries au plomb de 12V et ont une chacune charge de 7Ah.

Elles serviront d'alimentation principale au contrôleur de moteur électrique et seront disposer en série à l'arrière du kart.

Pédale d'accélérateur

La pédale d'accélérateur est une pédale électrique alimenté en 5V par la borne « Throttle » du contrôleur de moteur avec un signal de contrôle positif, 0--5V.

C’est par le billet de cette pédale que nous allons pouvoir réguler à notre guise la vitesse de rotation du moteur électrique via un 3ème fil « signal » qui permet de lire des tensions comprises entre 0 et 5V par le contrôleur pour moduler la tension de sortie au moteur (0-36V).

Le choix pour cette pédale était simple, elle acceptait les tensions de 5V que le contrôleur délivrait et dont on avait besoin pour le brancher.

Schéma de câblage

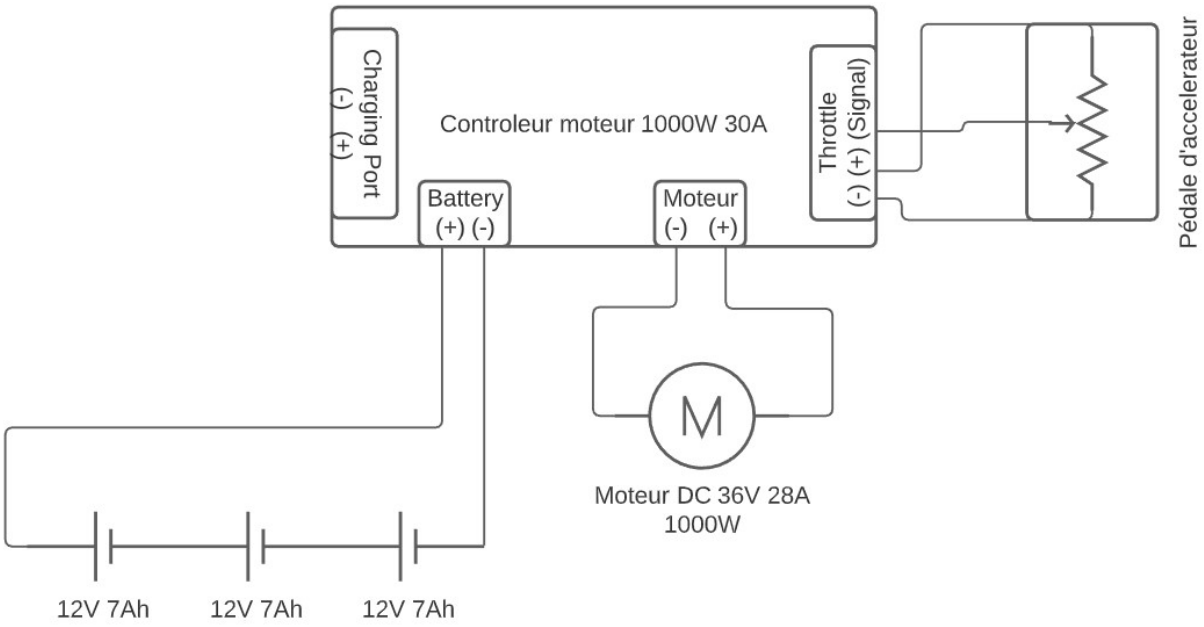
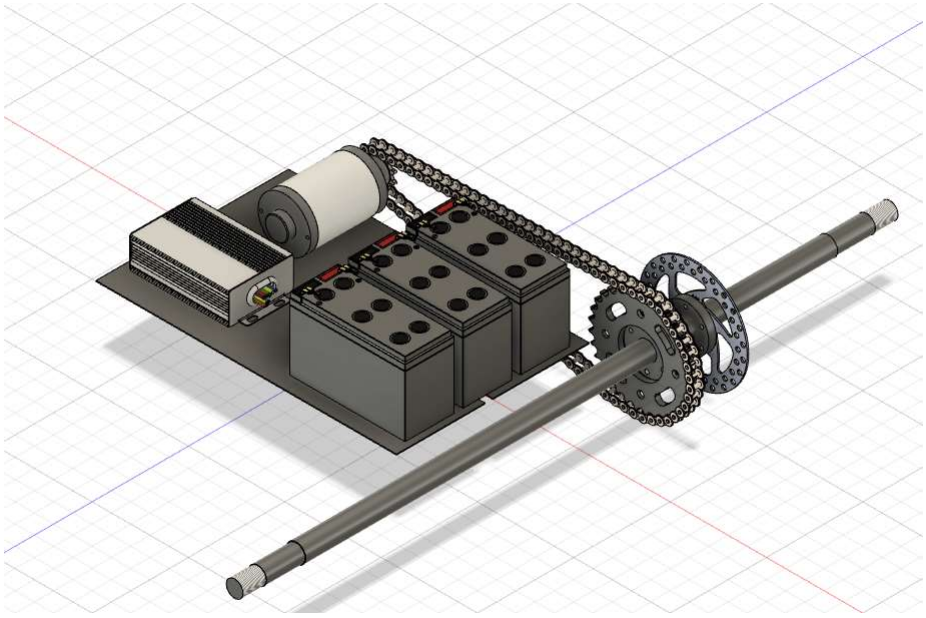
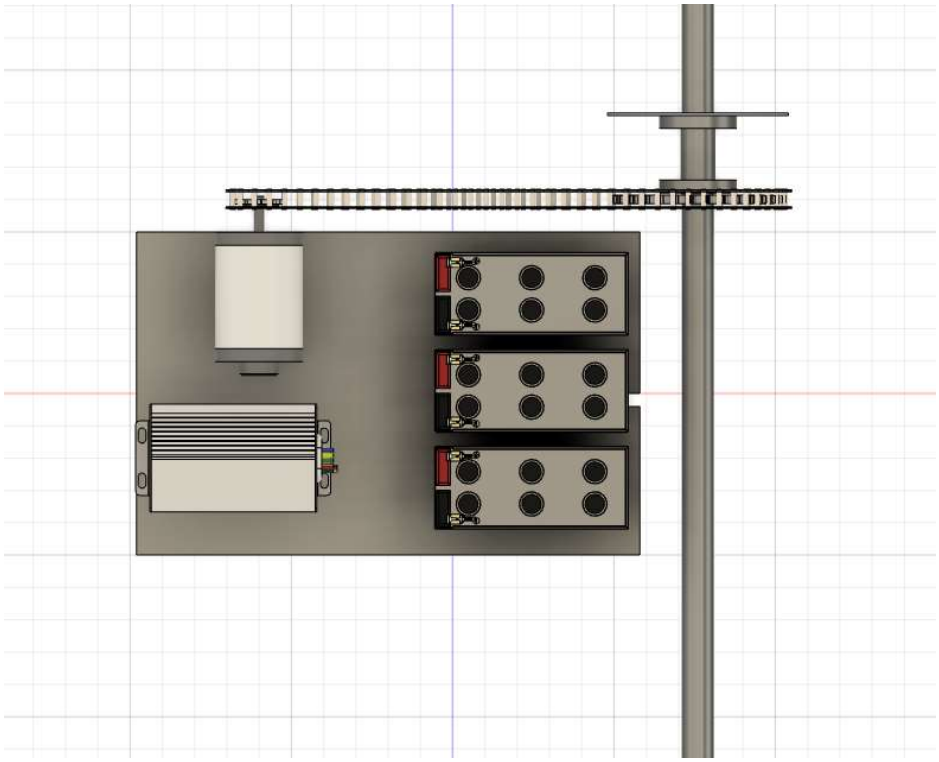
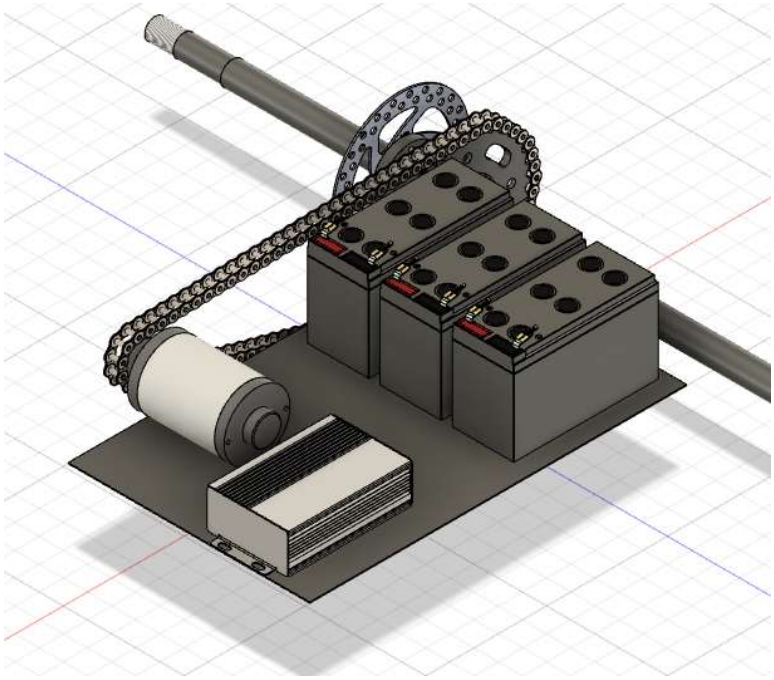


Schéma 3D

3AU







D. Répartition des tâches

Fasquel Sacha : Sécurité et chargement
Hemeleers Emile : Mouvement
Alvarez Lopez Juan : Sécurité et chargement, Options, Eclairage, Communication
Grabenweger Olivier: Options, PCB, Eclairage, Communication
Malewo Kanda : Mouvement

E. Difficultés rencontrées

L'une des plus grandes difficultés que nous avons rencontrées fus la transmission par chaines. Au commencement du projet, nous avions pris comme référence un pignon déjà compris avec le moteur DC pour le dimensionnent de la chaine de transmission et du plateau (2nd roue dentée) à poser sur l'axe des roues arrière. Nous avions trouvé une chaine qui correspondais parfaitement au pignon du moteur et il nous manquait seulement un plateau que nous avions trouvé sur internet et dont les dimensions correspondaient elles aussi à la chaine. Vint le moment de tester notre chaine et le plateau que nous avions remarqués que

celles-ci ne correspondaient pas et n'allaient pas ensemble. Suit à cela, nous avions décidé d'en commander un autre avec des dimensions qui pourrait correspondre avec notre chaine. Là aussi, le même résultat que pour le précédent plateau, elle ne correspond pas à la chaine. Par manque de temps (à cause des livraison tardives) et peur de se retrouver avec toujours le même problème nous avions essayé des solutions en plastique dur par impression 3D qui furent elles aussi infructueuses.


C'est alors qu'on nous conseilla d'utiliser des chaines et roues dentées de vélo pour palier à notre problème de transmission ce qui fut un véritable succès.

F. Conclusion

Nous avons appris à travers notre projet combien il était compliqué développer un kart électrique avec toutes les idées de base que nous avions prévu au départ, que celui-ci requérait de notre part plus de temps que prévu donc un beaucoup plus grand investissement en temps. En effet, pour la partie mécanique et électronique, nous avions dû faire beaucoup de compromis qui nous ont fait perdre un temps fou durant la conception du projet.

Malgré la faite que nous comprenions nos limites dans ce projet, nous sommes arrivés à aboutir notre projet même si le résultat n'est pas celui escompté. Nous avons pu revisiter nos cours d'automaticien et sortir un peu hors de notre zone de confort pour explorer un peu d'autre domaine lors de ce projet qui nous fut bénéfique et dont on retiendra quel sont les approches à avoir avec ce genre de projet, quel difficulté il faut appréhender au préalable et où mettre la barre en début de projet.

G. Sources

CAN SPI:

- https://github.com/pierremolinaro/acan2515
- https://www.locoduino.org/spip.php?article268

ESP32:

- https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

Contrôleur moteur :

- https://usefulldata.com/1000w-dc-brusch-motor-controller/

Moteur à courant continu :

- https://fr.wikipedia.org/wiki/Machine_%C3%A0_courant_continu

HC-SR04 datasheet:

- https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

3144 hall sensors:

- https://www.digchip.com/datasheets/parts/datasheet/029/3144-pdf.php

Crémaillaire de direction:
http://www.fiches-auto.fr/articles-auto/fonctionnement-d-une-auto/s-1199-la-direction-assistee.php