Rapport TER

SKATE Electrique

Université Paul Sabatier
M1 SME

DEDJEH Ove-Anselme
MAZARGUIL Marlon

2021-2022

Introduction

Les vélos sont très encombrants (par leur taille et souvent volés), pratiques pour les longues distances et très rapides mais pas pour les petits trajets du quotidien.

Aujourd'hui les longboards (skateboard très long) sont beaucoup utilisés pour répondre à ce problème (plus léger, plus petit) mais peu pratique pour les monter et il est difficile d'aller vite.

C'est pour cette raison que le skate électrique a été créé. Mais le skate accessible au public est encore très basique. Nous souhaitons donc en créer avec de nouvelles fonctionnalités qui pourraient faciliter le quotidien des personnes et le rendre plus accessible au grand public, en gardant les sensations d'un longboard.

C'est pour cette raison qu'on à choisi le projet de fabriquer un skate électrique pour le TER.

Pour réaliser ce projet nous avons du 10/01/2022 au 30/04/2022.

Pour réaliser le projet nous devons dans un premier temps réaliser un prototype qui fonctionnerait avec une télécommande sans fil, une fois le prototype validé on ajoutera différents types de capteurs, par exemple :

- Un capteur de pression capteur pour faire avancer le skate si on met notre pour en avant (sans télécommande)
- Le pourcentage de batterie du skate
- Un capteur de vitesse
- Un capteur qui compte le nombre de tours de la roue pour connaître le nombre de km parcourue
- Un gyroscope pour connaître l'orientation du skate

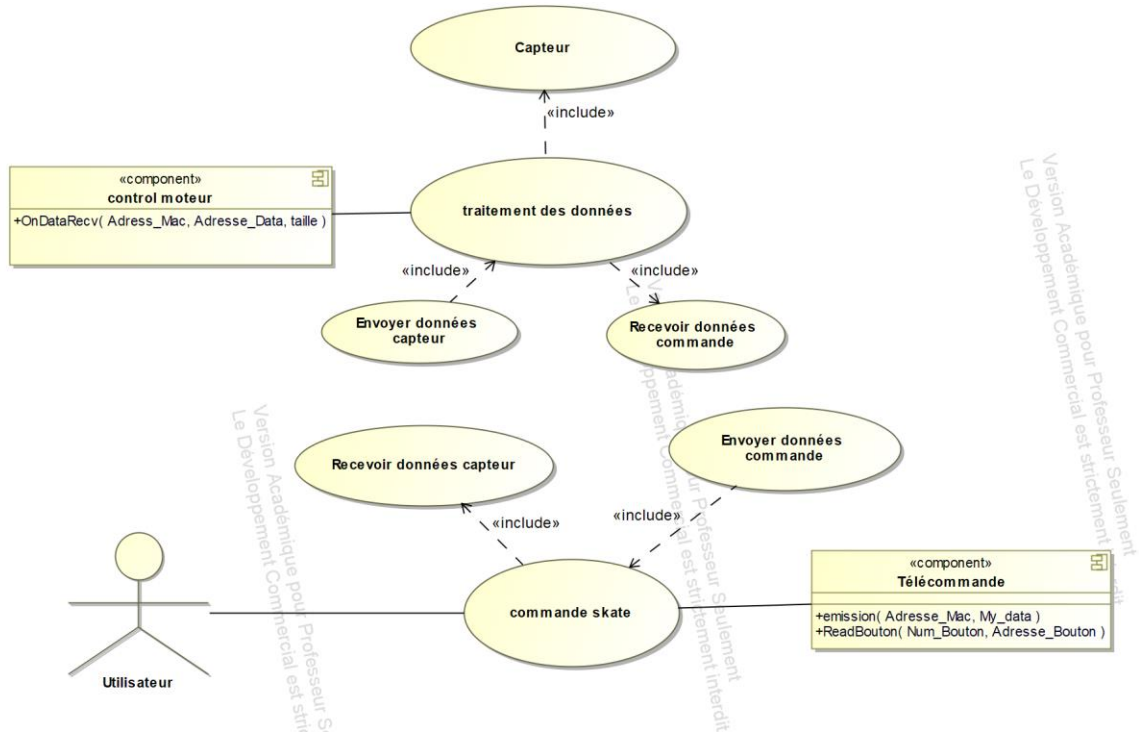
On réfléchira à différent type de de système pour apporter une valeur ajoutée au projet :

- ➔ Faire un "système moteur" qui s'enlève et se remette à volonté, cela permettrait d'avoir un skateboard normal ou électrique
- ➔ Faire un système roue libre
- ➔ Faire un système qui recharge la batterie quand on freine

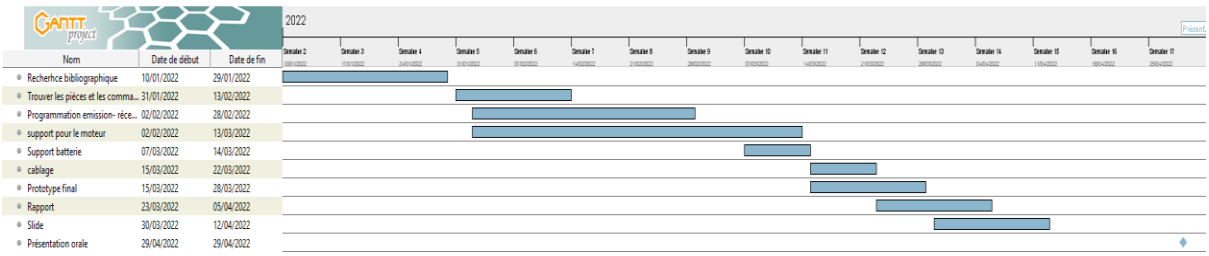
Et pour finaliser le projet on remplacera les pièces imprimées en 3D pour des pièces en métal.

Organisation

Cas d'utilisation

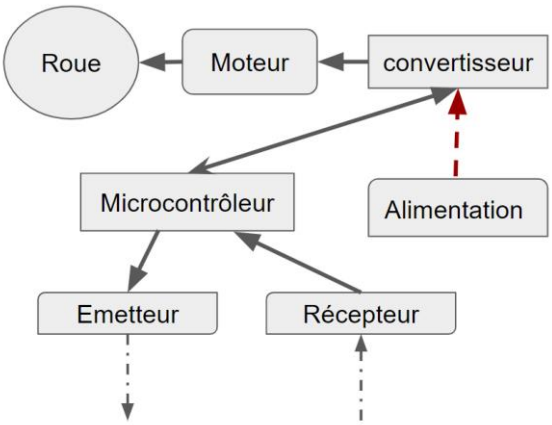


Organisation (GANTT)



Système moteur

Pour comprendre comment marche la partie système moteur on peut faire un graphique de fonctionnement :



Fixation du moteur au Skateboard

Pour fixer le moteur au skate nous avons 2 solutions :

- Prendre un moteur intégré dans la roue (hub motor)
- Faire un système qui relie le moteur à la roue avec une courroie (belt motor)

hub motor

belt motor

Nous allons comparer les deux choix pour savoir quel solution est la meilleur :

Système	**Hub Motor**	**Belt Motor**
Pièces	Hub Motor	Moteur
	Batterie	Batterie
	ESC brushless	ESC brushless

		Courrois
		Maintien moteur
		Lien roue-courrois (Wheel pulley)
		Lien moteur-courrois (Diver pulley)
Temps	Livraison (plusieur mois)	Livraison ou fabrication 3D (plusieur mois)

Le Hub Motor est un système fiable et robuste comparé au Belt Motor. Mais le temps de livraison était grand (plusieurs mois) pour espérer finir le projet avec la date limite.

Nous avons donc choisi le système Belt Motor, maintenant nous avions le choix entre commander les pièces où les fabriquer en 3D. Le temps de livraison était trop important et nous voulions faire un skate le moins cher possible.

La fabrication des pièces en 3D est presque gratuite, alors que le prix des pièces sur le marché est de 100 à 200 euros.

Nous avons donc choisi de faire les pièces en 3D pour faire un premier prototype du skate, puis dans un second temps pour faire les mêmes pièces en métal pour consolider le système.

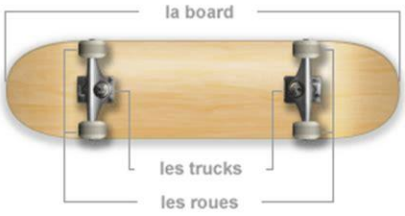
Choix des composants

Nous devons choisir les composants qui vont permettre la fabrication du skate

Choix du Skateboard

Il existe beaucoup de types de skateboard, cf photo ci-dessous.

Types de Skate



Composition d'un skate

Les critères pour choisir le skate :

- Stable (pour aller à 30km/h)
- Suffisamment haut pour mettre les composants (environ 5 cm)
- Possibilité de mettre des grandes roues (pour pas que la roue touche la board quand on tourne)

Le skateboard qui correspond à ces critères est le **longboard :**

Choix du Moteur

Nous devons avoir un moteur qui ait un rapport couple-vitesse suffisant pour monter une pente et démarrer et qu'il aille au moins 30 km/h
Nous avons choisi le moteur : MFO Prop Promenade 50-60 Série 380kV / 2665W

Choix de la Batterie

La batterie ne doit pas être trop grande pour qu'on puisse la mettre en dessous du skate, et elle doit permettre une autonomie de 30min à 1h à 30km/h.

Nous avons choisi comme batterie : Multistar High Capacity 5200mAh 6S 10C Multi-Rotor Lipo

Cette batterie reste un peu grande pour se mettre en dessous du skate sans qu'elle ne touche le sol, comme elle contient 6 batteries collées les unes aux autres, on pourrait les décoller pour qu'on prenne moins d'espace

Choix du ESC Brushless

Nous avons choisi ESC Brushless : YEP 60A (2~6S) SBEC Brushless Speed Controller

Il faut acheter un autre composant pour configurer l'ESC Brushless.

Choix du Microcontrôleur

Nous avons choisi un ESP32 DevkitC V4 comme microcontrôleur pour contrôler le moteur.

Ce choix a été fait en fonction des contraintes de la communication (cf partie : Télécommande)

La contrainte que nous avions avec le microcontrôleur est dans la communication, nous pouvons pourquoi nous avons fait le choix de prendre un ESP

Fabrication des pièces

Pour fabriquer les pièces en 3D nous avons pris pour exemple les pièces sur le marché
Comparaison des logiciels 3D :

Logiciel	**Inventor**	**Openscad**	**Sketchup**
Maîtrise	Moyen	Moyen	Moyen
Type de modélisation	Traits / Calques	Contraintes	Programmations
Taille	2KB	Plus de 10GB	1KB
Utilisé	Architecture	Fabrication de pièces mécanique	Fabrication de pièces

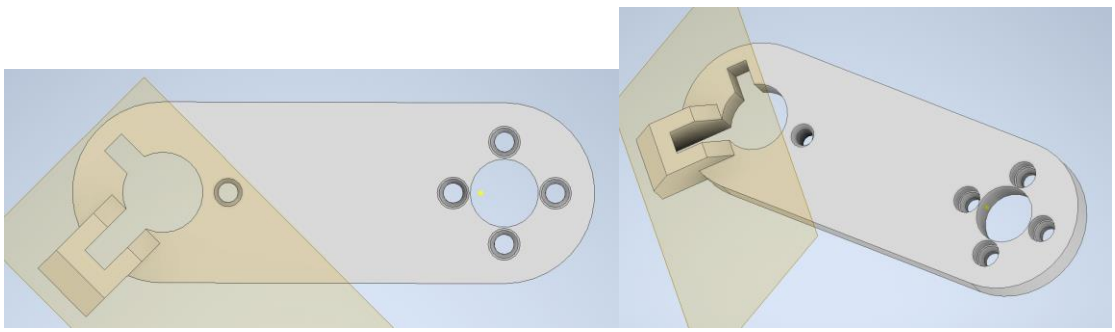
On a choisi le logiciel Inventor parce qu'on peut tester l'assemblage des pièces

Maintien moteur

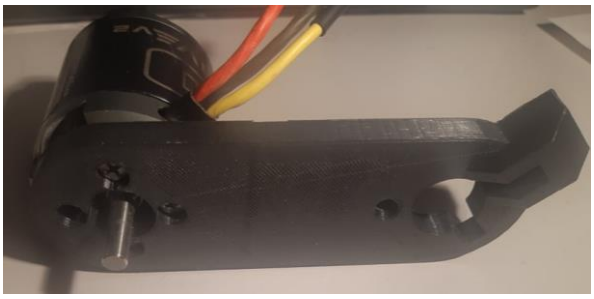
Pièce sur le marché

nom de la pièce : V7 63MM REVERSE MOTOR MOUNT ONLY

Pièce fabriqué en 3D



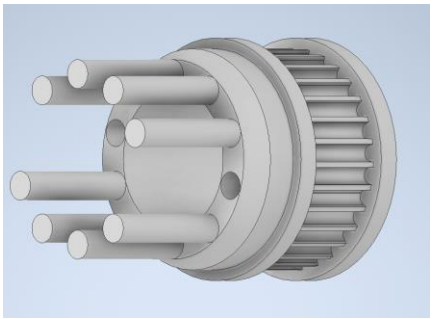
Impression 3D



Lien roue-courrois (Wheel pulley)

Nom de la pièce : 36T 15MM KEGEL DRIVE WHEEL PULLEY ONLY

Pièce sur le marché



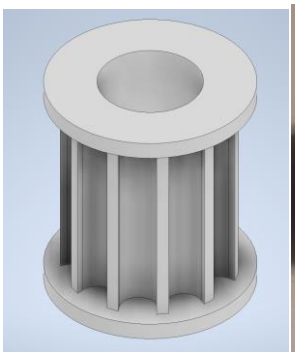
Pièce fabriqué en 3D

Impression 3D

Lien moteur-courrois (Diver pulley)

Nom de la pièce : 16T HTD5 12MM MOTOR PULLEY

Pièce sur le marché



Pièce fabriqué en 3D

Impression 3D

Partie fonctionnelle

Dans cette partie nous verrons tout ce qu'il y a en rapport avec le fonctionnement du système intégré au skate.

Fonctionnement du système (UML)

Les données transmises à la télécommande peuvent varier en fonction des capteurs, on peut envoyer :

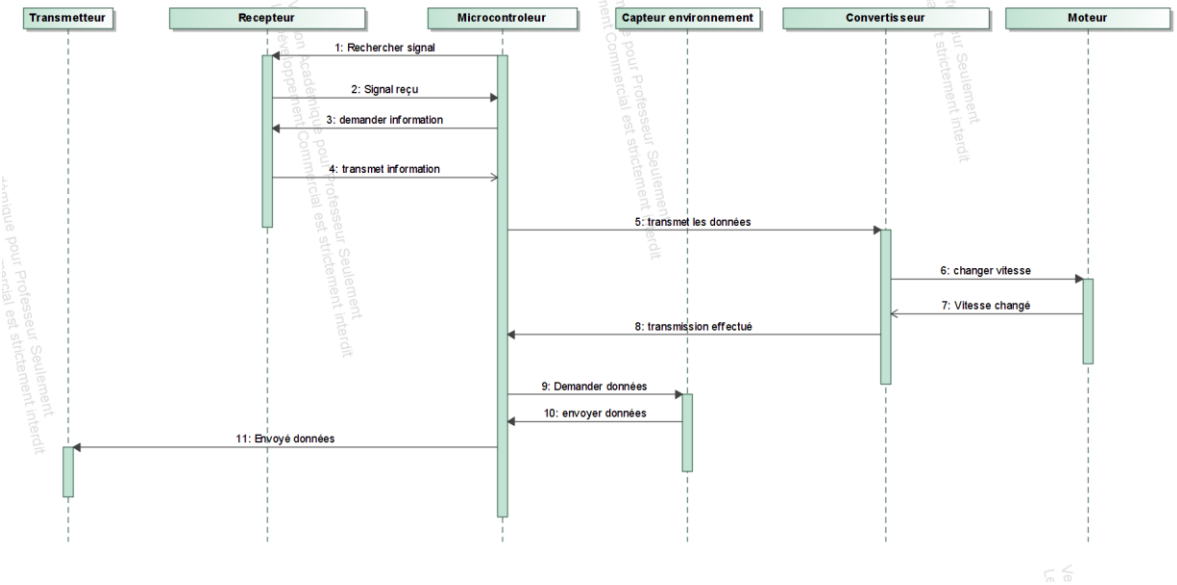
- Le pourcentage de batterie du skate
- la vitesse
- l'état des capteurs
- le nombre de km parcourue
- L'orientation du skate (avec un gyroscope)

On peut avoir 2 types de fonctionnement du moteur :

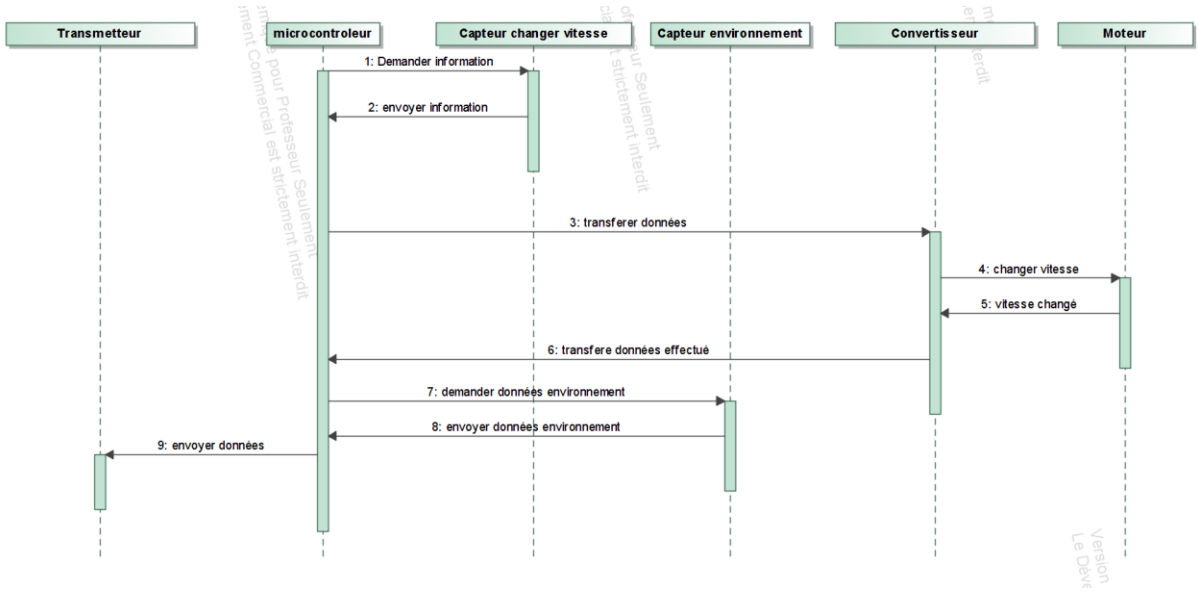
1. Soit il est piloté avec une télécommande
2. Soit il est piloté avec des capteurs (exemple : capteur de pression à l'avant et l'arrière du skate)

On va montrer le diagramme de séquence pour les deux cas :

1) Le principe de fonctionnement pour contrôler le moteur avec une télécommande :



2) Le principe de fonctionnement pour contrôler un moteur avec des capteur :

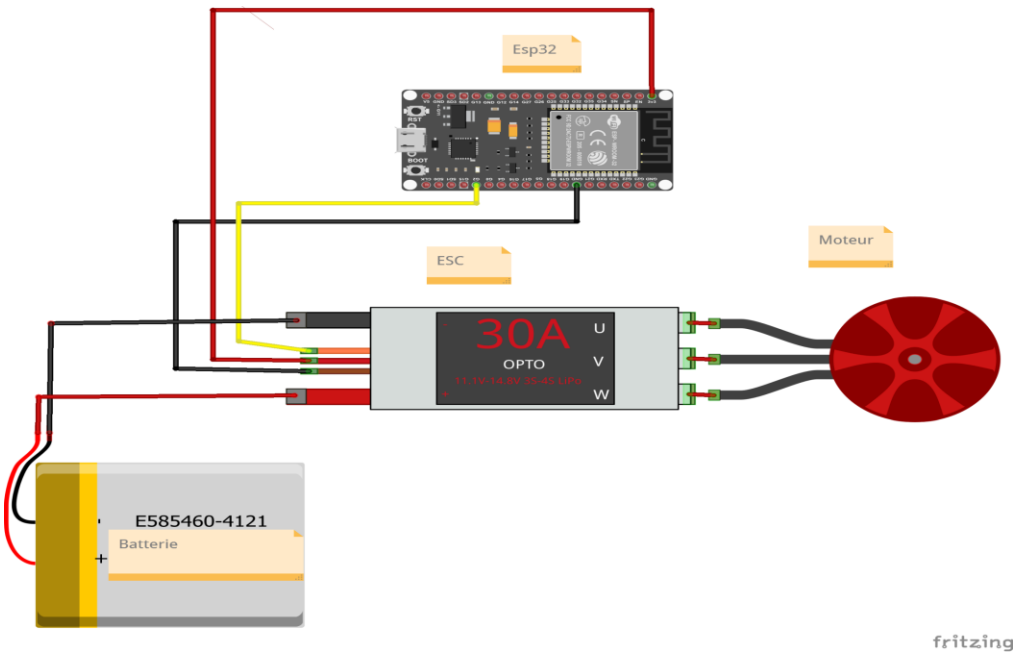


Processus de mise en œuvre

Pour réaliser ce système nous avons fait en parallèle le système moteur et le système de communication.

Pour la partie système moteur nous avons commencé par simuler la réception d'un signal pour la valeur d'un potentiomètre, puis par des boutons. Une fois qu'on a vu que tout fonctionnait, on a pu ajouter le système de communication.

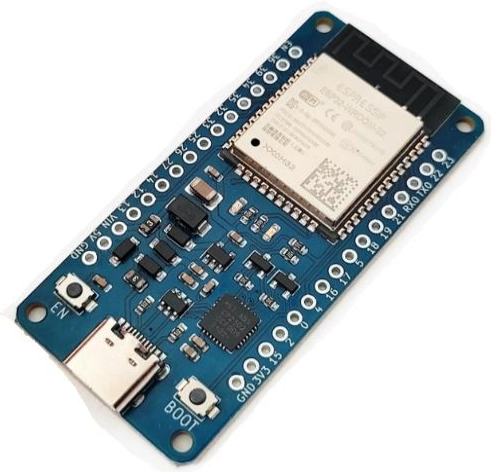
Ci-dessous le schéma électrique du système moteur :



Système de communication

Télécommande

Dans le cadre de réalisation du skate électrique, nous avons opté pour une télécommande wifi basée sur une carte de développement esp32 DevkitC V4.



La télécommande doit être la plus ergonomique possible afin de permettre une prise en main agréable au skateur.

Il a donc fallu trouver les composants adéquats afin qu'elle soit assez petite pour respecter cette contrainte de dimension, nous avons le choix entre un Arduino nano et un Esp32. Mais nous avons finalement opté pour l'esp 32 car il intègre nativement la gestion du Bluetooth et du Wifi, contrairement à l'Arduino nano qui nécessite un Shield pour accomplir cette tâche.

De plus, la télécommande a été pensée pour avoir une assez grande autonomie et une liaison rapide et robuste avec le skate afin d'avoir une conduite agréable. Afin de respecter ces critères nous avons le choix du protocole de communication native de l'esp32 appelé Esp-Now.

Le protocole de communication

Esp-Now est un protocole de communication sans fil basé sur la couche de liaison de données définies par Expressif. La grande différence avec le Wifi et le BLE est que les données n'ont pas besoin d'être transmises via la couche réseau, la couche de transport, la couche de session, la couche de présentation et la couche d'application.

Avec Esp-Now, les cinq couches du modèle OSI sont donc réduites à une seule, ce qui facilite la communication et réduit la consommation d'énergie.

De plus, ce protocole occupe moins de ressources CPU et flash que les protocoles de connexion traditionnels, alors qu'il coexiste avec WI-FI et Bluetooth LE.

Il permet donc aux appareils jumelés de communiquer directement entre eux via la couche de liaison de données. L'ensemble du processus de couplage ne nécessitant pas de connexion Wi-Fi ou d'appareil tiers tel qu'un téléphone mobile.

Esp-Now est donc la solution pour établir une communication stable et rapide avec le skateboard. A présent, comment assurer une bonne autonomie à la télécommande.

Alimentation de la télécommande

Plusieurs options existent pour alimenter l'esp32 avec différents types de batteries. Nous avons opté pour une batterie rechargeable LiFePO4. L'alimentation d'un esp32 se fait sur sa broche de 3.3V ce qui fait que pour plusieurs types de batteries, il faut un régulateur de tension.

Mais avec un LiFePO4, on n'a pas besoin d'un régulateur de tension supplémentaire. La capacité de ces batteries allant jusqu'à 6000mAh, il confère à notre télécommande une longue durée de vie.

La batterie lithium fer phosphate (batterie LiFePO4) a une tension nominale de 3,2 V et une tension maximale de 3,65 V. Le principal avantage d'une batterie LiFePO4 est la courbe de décharge très plate de sorte que la tension chute très lentement pendant le processus de décharge.

Étant donné que la tension maximale de la batterie au lithium fer phosphate n'est avec 3,65 V que légèrement supérieure à la tension de fonctionnement maximale de l'ESP32 avec 3,6 V, vous pouvez connecter ce type de batterie directement avec la broche 3,3 V du microcontrôleur.

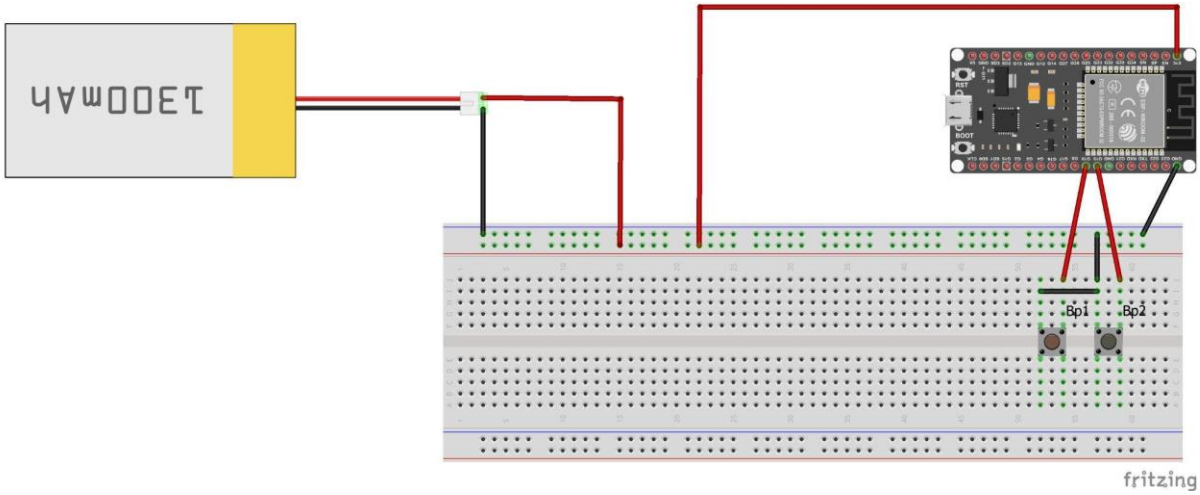
Critères de la batterie LiFePO4	**spécifications**
Tension de décharge minimale	2,5V
Tension de travail	3,0 V à 3,2 V
Tension de charge maximale	3,65V
Nombre de recharges	5000
Densité d'énergie	90 Wh / kg… 160 Wh / kg

En résumé, une batterie LiFePO4 convient très bien à l'ESP32 dans le cadre de notre application. L'inconvénient est qu'il est très compliqué de charger la batterie pendant son utilisation. Actuellement, on n'a pas de solution à ce problème. La solution la plus simple serait d'avoir deux batteries LiFePO4 que vous pouvez changer rapidement et un chargeur de batterie externe.

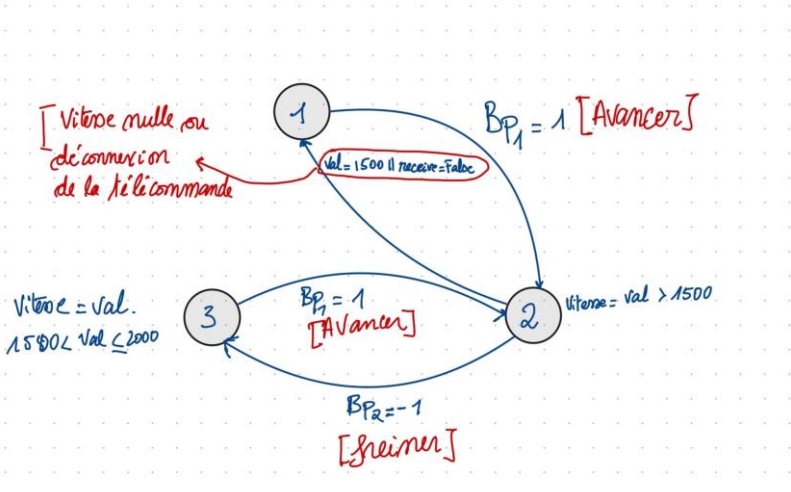
Nous n'avons pas pu implémenter cette solution. Pour les simulations et la présentation, nous alimenterons l'esp en 5 volt à l'aide d'une batterie portable.

Schéma de câblage et fonctionnement

Maintenant que nous avons défini, et trouvé les composants qui nous permettront d'avoir une télécommande respectant notre cahier des charges, nous sommes passés à la réalisation d'un schéma de câblage de la télécommande.

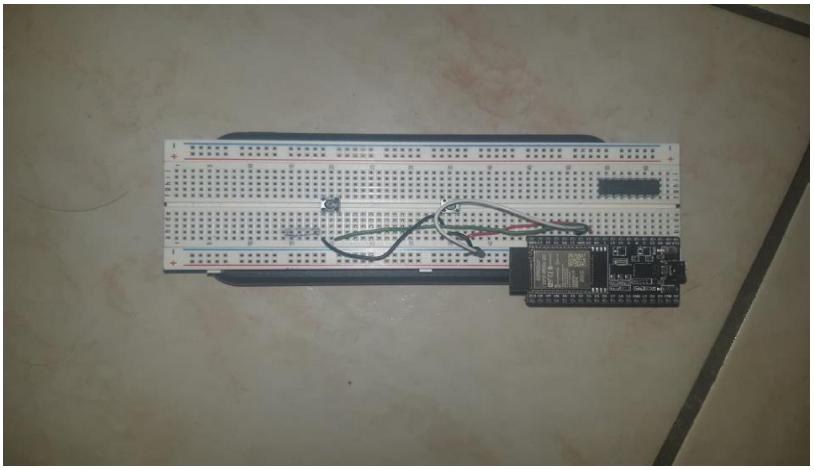


Cette télécommande lit les valeurs des pine 18 et 19, puis transmet ces valeurs à l'esp32 situé sur le skate. Les valeurs reçues sont ensuite implémentées dans une machine à état pour assurer la commande effective du skate.



Prototype de la télécommande

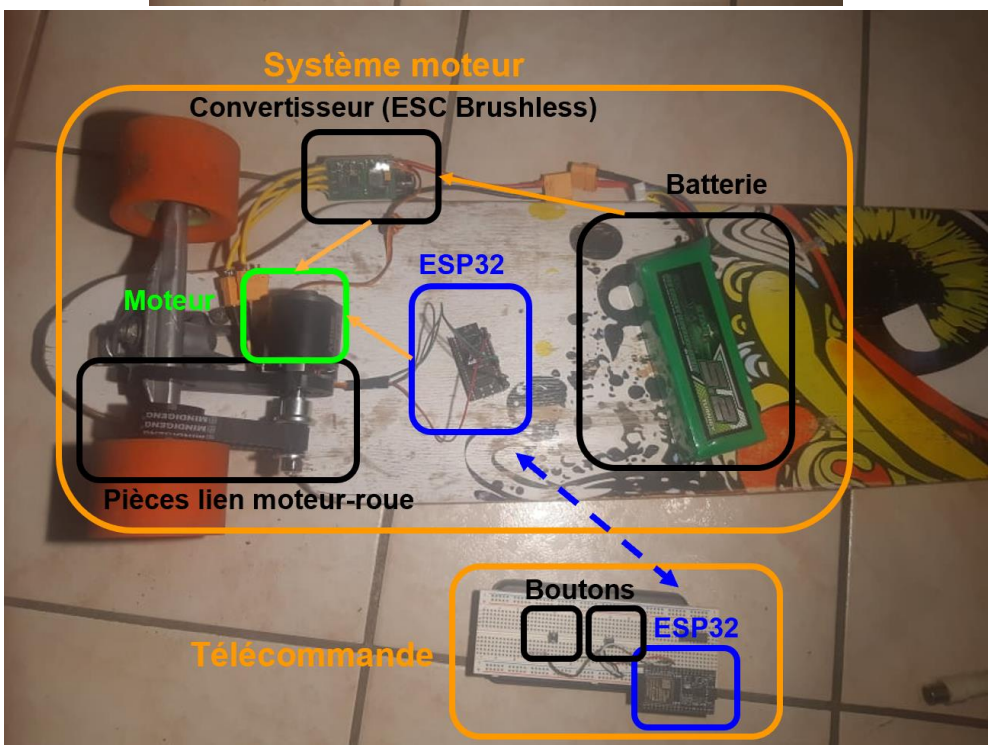
En raison d'un grand retard de nos commandes, nous n'avons pas eu la possibilité de désigner votre télécommande comme nous l'aurions voulu. Cependant à la date butoir de ce projet, nous sommes en mesure de fournir un prototype totalement fonctionnel de la télécommande.



Réalisation du projet

Nous avons pu réaliser un premier prototype fonctionnel, mais pas opérationnel.

Tous les composants fonctionnent correctement les uns avec les autres, mais dans la partie système moteur, les pièces qui font le lien entre le moteur et la roue ne sont pas encore “fixe” c’est-à-dire que si le skate est en mouvement les pièces bougerons et il faut créer la pièce support batterie, qui permet de maintenir la batterie et tous les composants électroniques.



Conclusion

L'objectif de ce projet TER était de mettre au point un skate électrique avec différents capteurs, qui pourrait permettre soit de commander le skate soit d'avoir des informations (état de la batterie, kilomètre parcouru, vitesse). Notre travail dans le cadre de ce projet TER nous a amené à réaliser un prototype de skate électrique avec les fonctions de base.

Certaine partie du projet on prit plus de temps que nous avions prévu, c'est le cas pour le temps de livraisons des composants et le temps de fabrication des pièces 3D.

Bien que le skate soit fonctionnel, il n'est pas encore opérationnel, pour qu'il le soit il faudrait améliorer les pièces 3D et faire le support des composants.

Une fois le skate opérationnel nous pourrions choisir/ajouter les capteurs au skate électrique et créer les systèmes de recharge de batterie, de mode roue libre … et finir par la métallisation des pièces.

En ce qui concerne la télécommande, il faudrait faire boîtier 3D intégrant une pile LiFePo4 et le microcontrôleur et les boutons.

Par la suite nous pourrions réaliser d'un PCB afin de réduire l'espace occupé par l'ESP en ne faisant un câblage qu'avec nos PIN d'intérêt dans la réalisation de la télécommande.

Annexes

Code récepteur

```
#include
<Arduino.h
>
              #include <WiFi.h>
              #include <esp_now.h>
              #include <ESP32Servo.h>

              //uint8_t Adresse_Mac[] = {0xE8, 0x68, 0xE7, 0x30, 0x54, 0xDC};
              bool Appel = false;
              esp_err_t Status_recv;
              int Val;
              //uint8_t ValRecue;
              uint8_t My_Data = 1;
              Servo esc; //Creating a servo class with name as esc

              void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData,
              int len)
              {
               Appel = true;


                 memcpy(&My_Data, incomingData,sizeof(My_Data));


              // ValRecue = My_Data;

              }
```

```
void setup() {
 // put your setup code here, to run once:
 Val = 1500;
 Serial.begin(115200);


 WiFi.mode(WIFI_STA);


 esp_now_init();


 Status_recv = esp_now_register_recv_cb(OnDataRecv);



 esc.attach(5); //Specify the esc signal pin,Here as D8


 esc.writeMicroseconds(1000); //initialize the signal to 1000


 //delay(10)



}


void loop() {
 Serial.println(My_Data);
 // put your main code here, to run repeatedly:


//Création de la variable ValRecue, ce sera la ValRecueeur reçu
par le recepteur


 //ValRecue= analogRead(A0); //on lis sur la Broche A0 les données
analogique
```

```
 ////////////////////
 //Debut du code recepteur




 //Fin du code recepteur
 ////////////////////


 //ValRecue= map(ValRecue, 0, 1023,1000,2000); //mapping ValRecue
to minimum and maximum

//map(ValRecueeur,min_ValRecueeur,max_ValRecueeur,min_change,max_c
hange)
 //ValRecueeur : la variable
 //min_ValRecueeur ou max_ValRecueeur : ValRecueeur min ou max de
la variable (changer ces ValRecueeur si besoin)
 //min_change ou max_change : la nouvelle ValRecueeur min et max
qu'on soit avoir


 switch (My_Data)
 {
 case 0:
   if(Val<2000 && Appel == true)
   {
     Val++;
     delay(10);



   }
   else
     My_Data = 255;
   break;


 case 1:
   if (Val >1500 && Appel == true)
   {
```

```
      //Val--;
      Val = Val;


    }
    else
      My_Data = 255;
    break;


  case 255:


    if (Val > 1500)
    {
    Val --;
    delay (5);
    }
    break;


  default:


    break;
  }
  esc.writeMicroseconds(Val); //using ValRecue as the signal to esc
  Appel = false;
  //My_Data = 255;
  delay(50);
  Serial.println(Val);
 }
```

Code émetteur

Emission.h:

```
#ifndef __EMISSION__
#define __EMISSION__

#include <esp_now.h>


void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status);
void emissionInit(uint8_t *Adresse_Mac, esp_now_send_cb_t cb);
// void pairing (esp_now_peer_info* peerInfo);
void emission(uint8_t *Adresse_Mac, uint8_t My_Data);

#endif
```

Bp.h :

```
#ifndef __BP__
#define __BP__
#include <Arduino.h>

const int Bp1 = 18;
const int Bp2 = 19;

#endif
```

emission.cpp :

```
#include "Emission.h"
#include <Arduino.h>
#include <WiFi.h>
#include <esp_now.h>

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{

   if (status == ESP_NOW_SEND_SUCCESS)
   {
      Serial.println("Envoie  donné ok !!");
   }
   else
   {
      Serial.println("Erreur Envoie de donnée!!");
   }
}

void emissionInit(uint8_t *Adresse_Mac, esp_now_send_cb_t cb)
{
```

```
    // Initialisation du moniteur série
    Serial.begin(115200);

    // Definir l'ESP32 en tant que station WIFI
    WiFi.mode(WIFI_STA);

    // init ESP-NOW

    if (esp_now_init() == ESP_OK)
    {
        Serial.println("Initialisation OK !! ");
        Serial.println("cool");
    }

    else
    {
        Serial.println("Erreur Initialisation");
        ESP.restart();
    }

    // Appel d'une fonction de callback  **concept encore flou pour moi pour le moment **
    esp_now_register_send_cb(OnDataSent);

    /*
      Appairrage
    */

    // La structure Peerinfo

    esp_now_peer_info peerInfo;
    memcpy(peerInfo.peer_addr, Adresse_Mac, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;

    // Mise en place de la fonction peer et des verification afférentes

    if (esp_now_add_peer(&peerInfo) == ESP_OK)
    {
        Serial.println("Appairage Réussi");
    }
    else
    {
        Serial.println("Erreur Appairage");
    }
}

void emission(uint8_t *Adresse_Mac, uint8_t My_Data)
{
    esp_err_t Statut_Envoi = esp_now_send(Adresse_Mac, &My_Data, sizeof(My_Data));
```

```
    if (Statut_Envoi == ESP_OK)
    {
        Serial.println("Envoie OK pi!!");
    }
    else
    {
        Serial.println("Erreur Envoie !! ");
    }
}
```

Bp.cpp:

```
#include "Bp.h"
#include <Arduino.h>

void ReadBouton ( const int Pin ,uint8_t * StateBuf)
{
    (* StateBuf) = digitalRead( Pin );
}
```

main.cpp :

```
#include <Arduino.h>
#include <WiFi.h>
#include <esp_now.h>
#include "Emission.h"
#include "Bp.h"


uint8_t Adresse_Mac[] = {0xE8, 0x68, 0xE7, 0x30, 0x93, 0x18};

uint8_t My_Data = 0;

uint8_t StateBp1 = 0;
uint8_t StateBp2 = 0;

void setup()
{
  pinMode(Bp1, INPUT_PULLUP);
  pinMode(Bp2, INPUT_PULLUP);
  emissionInit(Adresse_Mac, OnDataSent);
  Serial.println("cool");

}

void loop()
{
  ReadBouton(Bp1,&StateBp1);
  ReadBouton(Bp2,&StateBp2);

  if ((StateBp1 == 0 || StateBp1 ==1) && StateBp2 ==1)
```

```
  {
    My_Data = StateBp1;

  }
  else
  {
    My_Data = -1;
  }
  emission(Adresse_Mac, My_Data);
  //delay(20);
}
```