

VroomVroomProject

Voiture Télécommandée



L'École des Ingénieurs Scientifiques

Informations complémentaires



Cette présentation a pour objectif de comprendre les étapes clés de notre projet, comment nous nous sommes organisés. Pour plus d'information et de détails, checkez :

Lien du git :

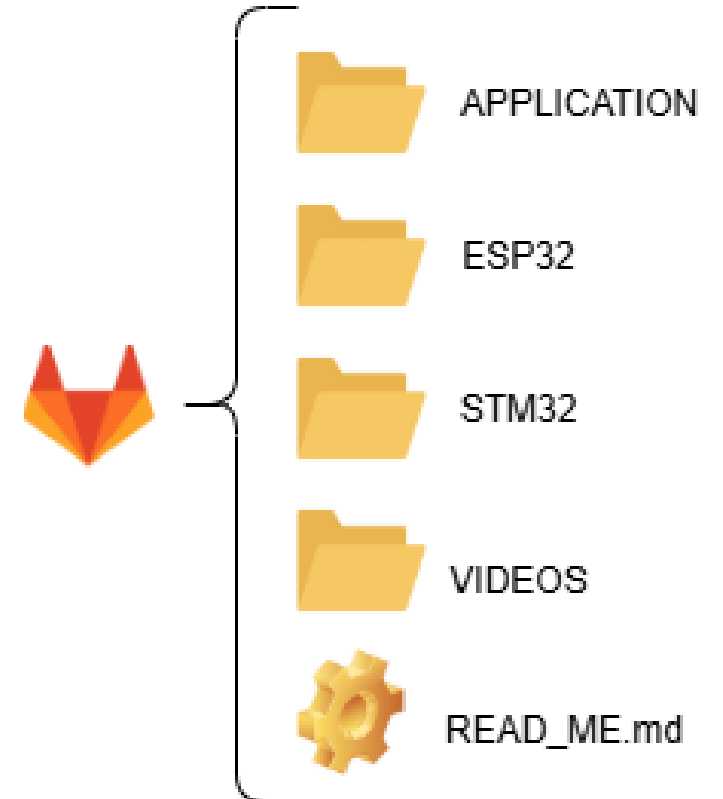
<https://gitlab.ecole.ensicaen.fr/rozoy/vroomvroomproject.git>

Lien du Trello :

<https://trello.com/b/GzUCtK32/vroomvroomproject>

Dans le git :

- > **VIDEOS** : Des vidéos sont présentes dans le dossier VIDEO du GIT pour une meilleure immersion dans le projet.
- > **READ_ME** : Indispensable afin de compiler le projet sans difficulté.
- > **APP / ESP32 / STM32** : Différentes sections du projet



Présentation de l'équipe



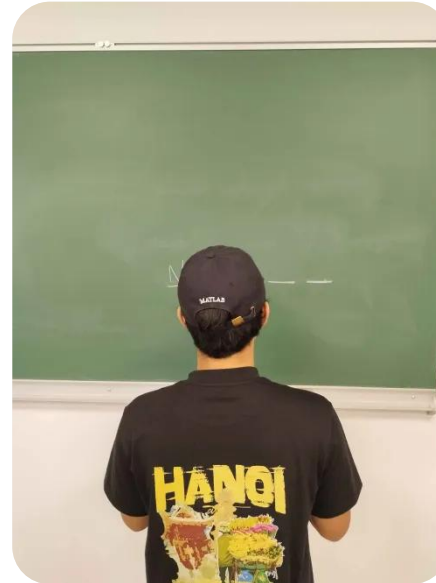
Alix Corbille

Dev LEDs
Debugs



Etienne Rozoy

Manager/Dev
STM32



The Phong Douangm(...)

Dev App
ESP32
STM32
Debug



Loïc Ricard

Dev STM32
Moteur (PWM)



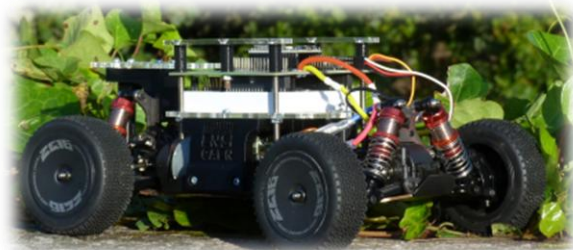
Esteban Vantorre

Dev ESP32
SPI

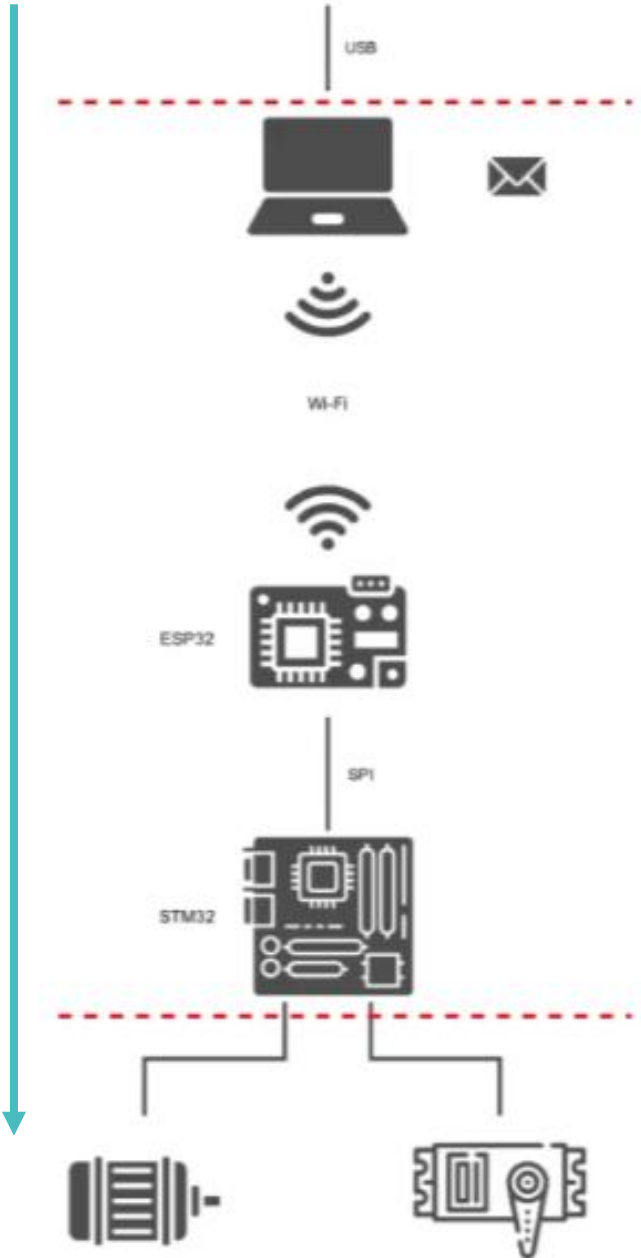
Architecture fil rouge



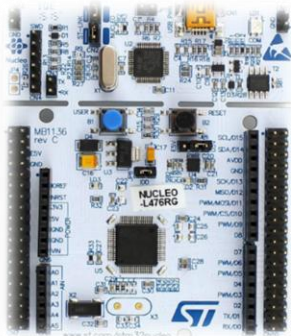
Carte ESP32



Données

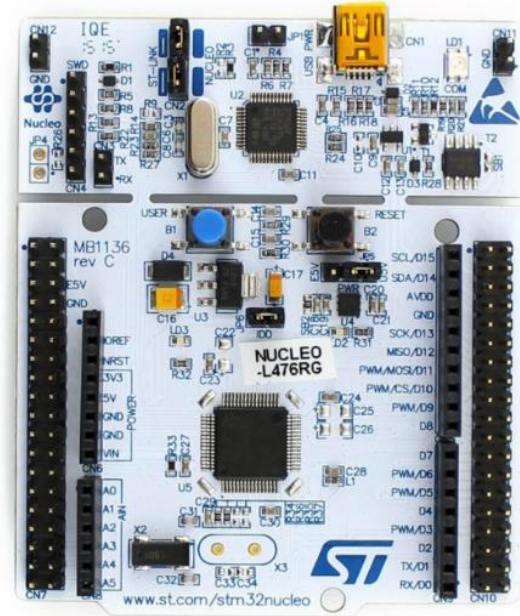
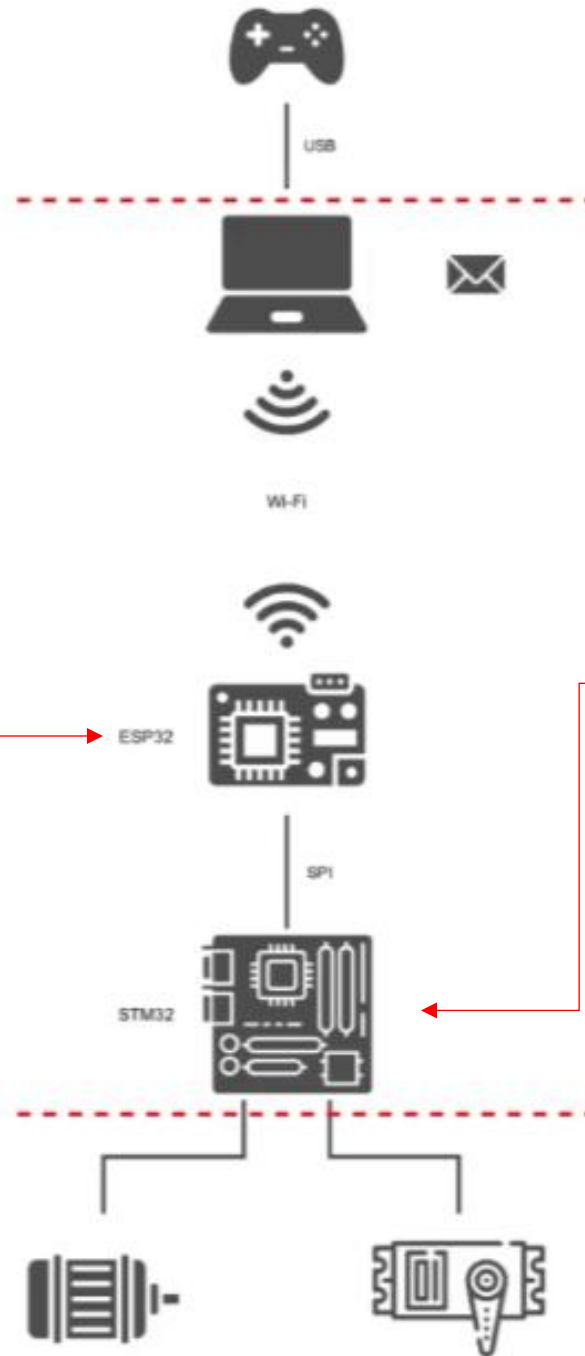


Manette Xbox



STM32

Matériels utilisés



L'**ESP32** est un microcontrôleur Wi-Fi et Bluetooth double-cœur, économique et polyvalent, conçu par Espressif pour des projets connectés et embarqués.

Fonction ici :

- ➔ Réception donnée en UDP (WIFI)
- ➔ Envoie donnée en SPI (vers STM32)

Le **STM32** est une famille de microcontrôleurs ARM Cortex de STMicroelectronics.

Fonction ici :

- ➔ Réception donnée de l'ESP32 (SPI)
- ➔ Dirige moteur par signaux PWM (Direction/Vitesse)
- ➔ Indicateur LEDs

Etapes de communication

Communication USB (Manette – APP)

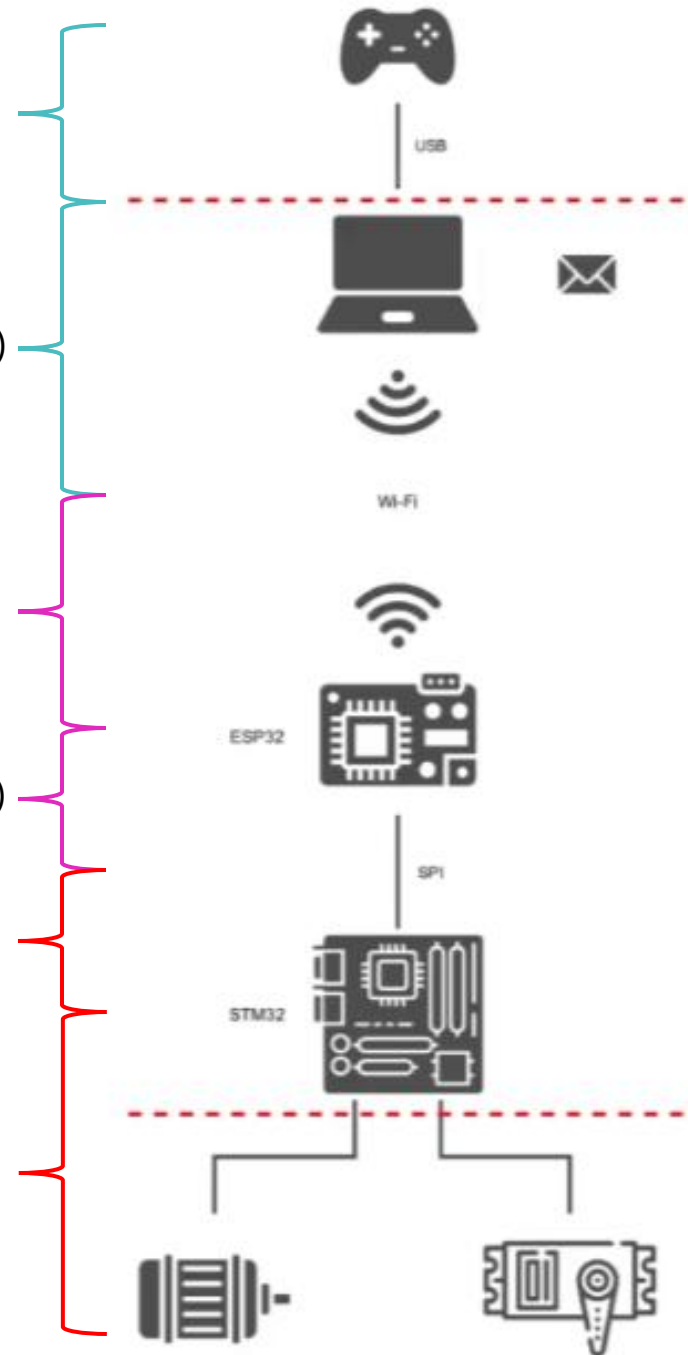
Envoi de données par UDP (APP – UDP)

Réception de données UDP (UDP – ESP32)

Envoi de données par SPI (ESP32 – SPI)

Réception de données SPI (SPI – STM32)

Envoi de données PWM (STM32 – Moteur)



Application (APP)

ESP32

STM32

UDP : Protocole réseau léger, sans connexion ni garantie de livraison, utilisé pour transmettre rapidement des paquets de données.

SPI : Protocole de communication série synchrone, rapide et simple.



Chapitre 1

Application

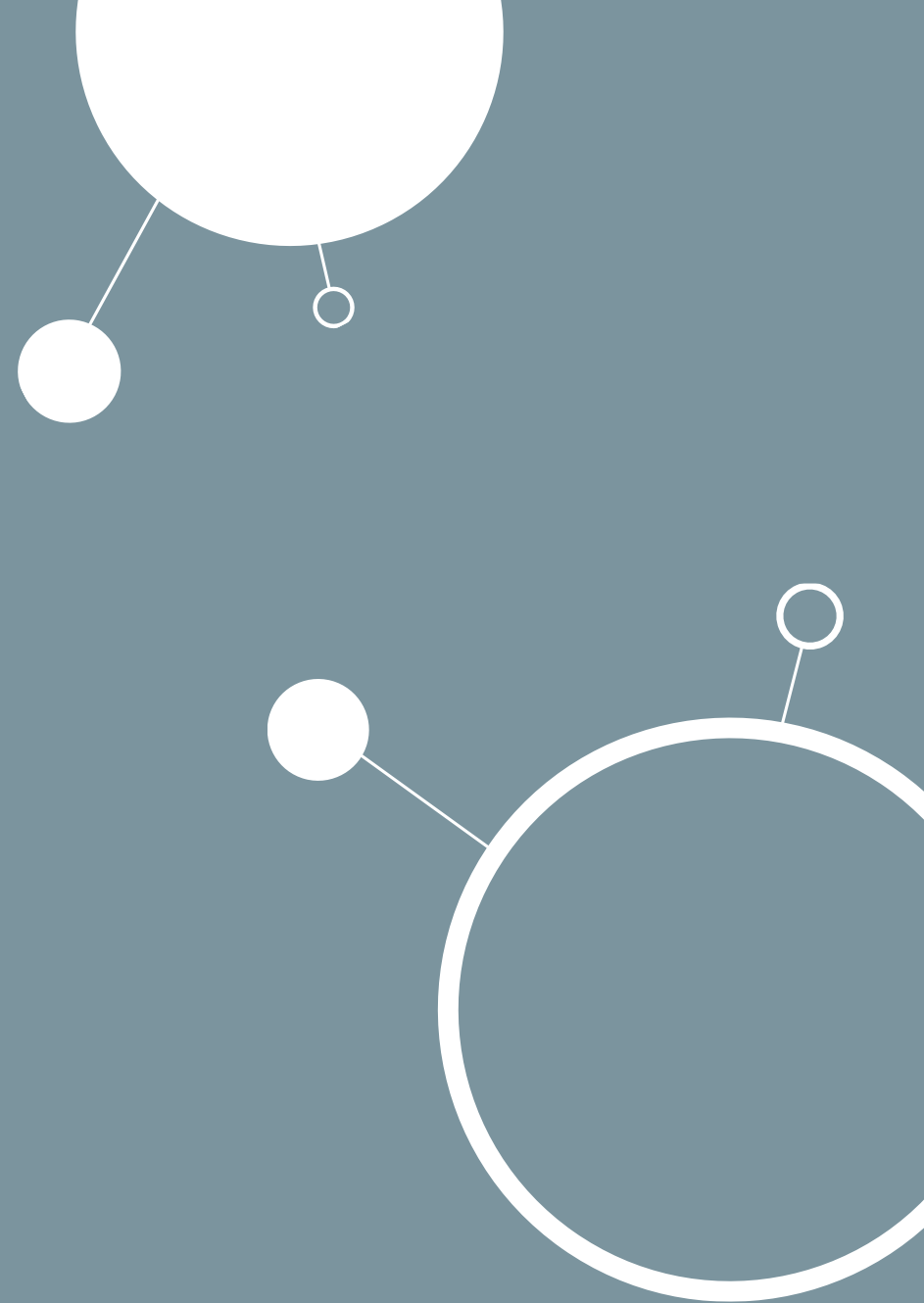
The Phong Douangmanivong

Principe / Fonctionnement

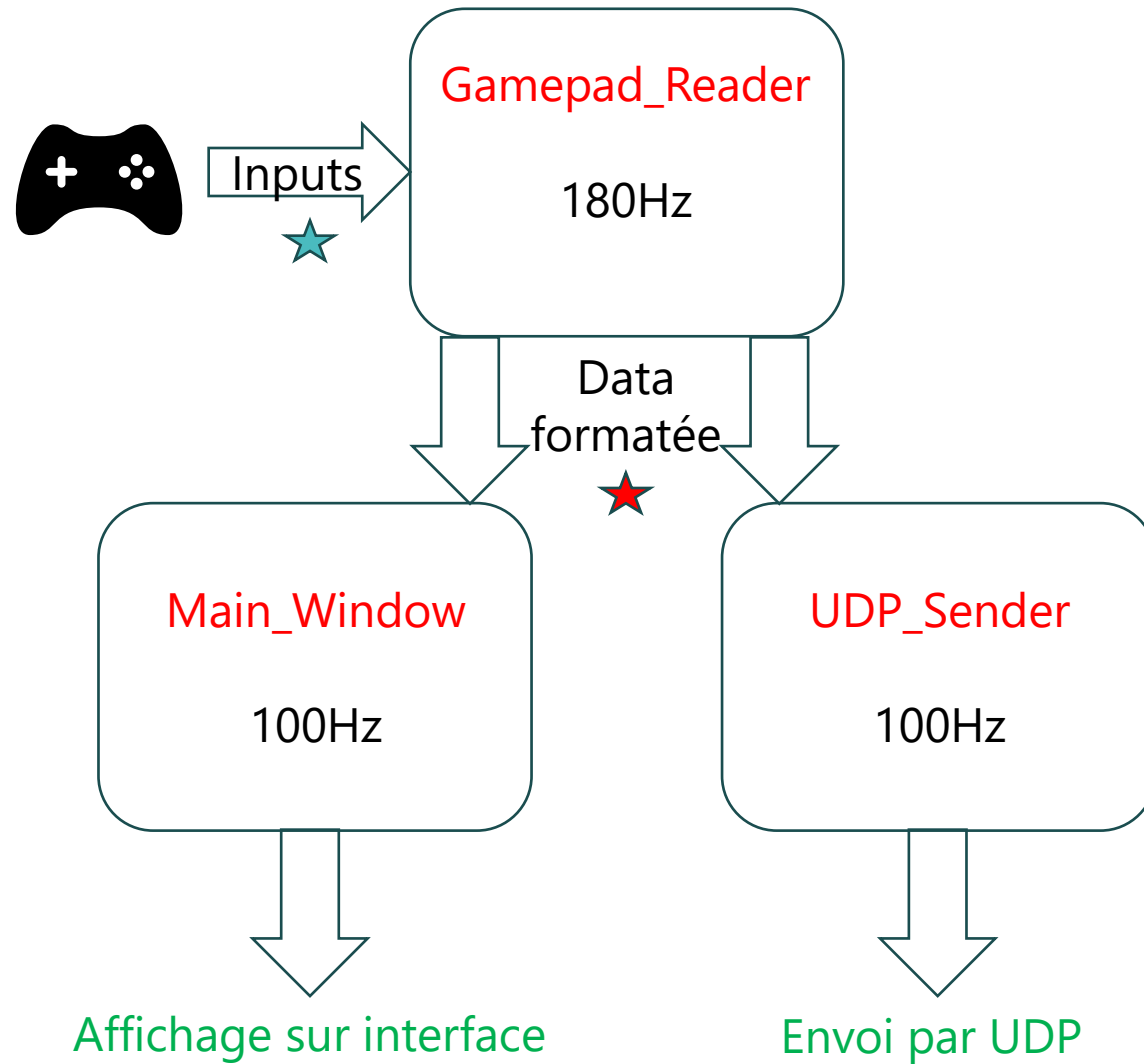
Mappage

Architecture

Implémentation

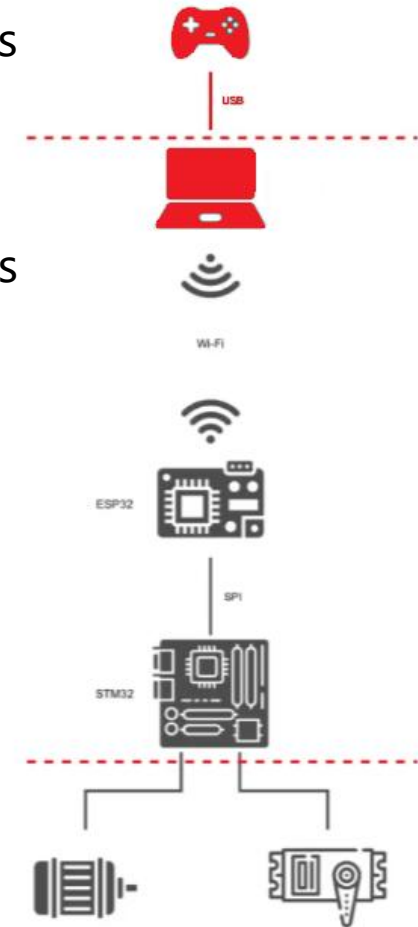


Application (Principe de fonctionnement)



★ Mappage boutons/joysticks avec pygame sur Linux

★ Mappage boutons/joysticks au format générique du tableau

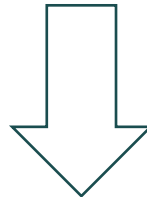


Application (Mappage)

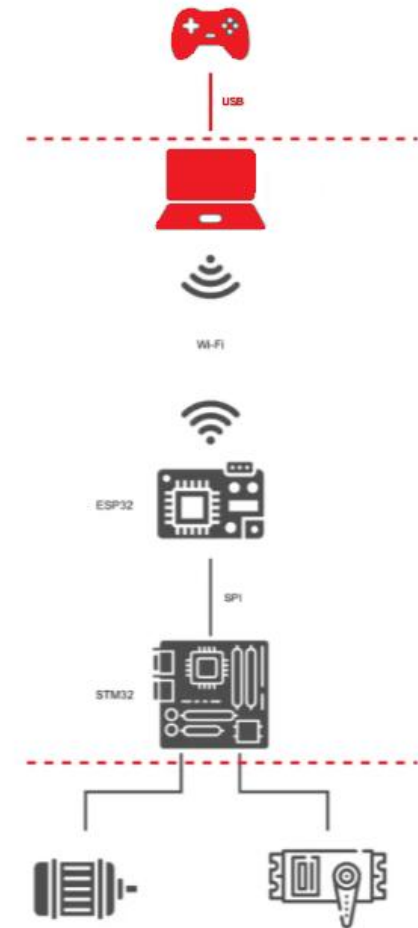
Button per bit	Ø	Right Joystick	Left Joystick	→	←	↓	↑	Xbox	Select	Share	RB	LB	B	A	Y	X
bit	32 -- 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Buttons	Left Joystick x-axis	Left Joystick y-axis	Right Joystick x-axis	Right Joystick y-axis	LT	RT
32 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits

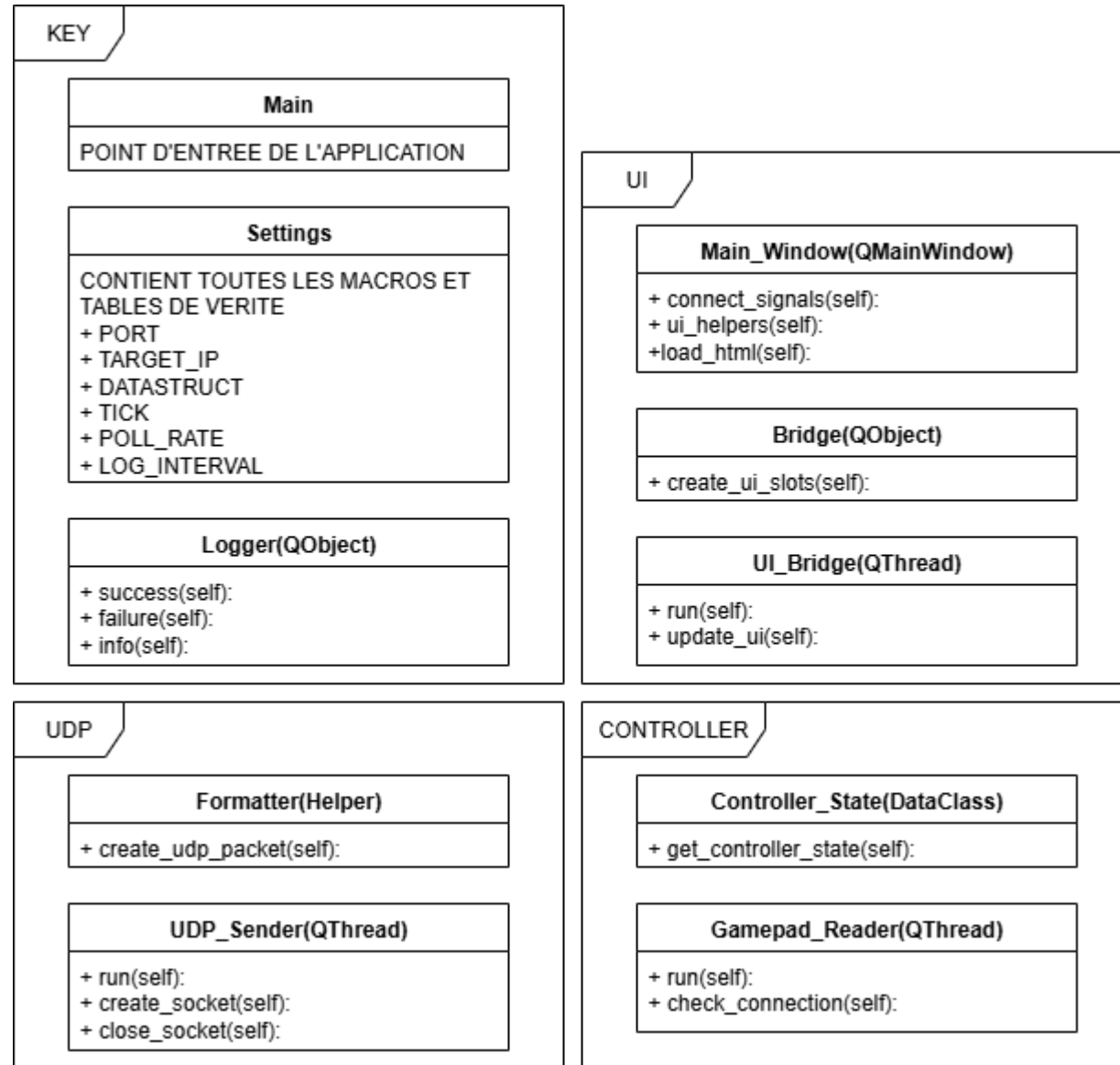
1 paquet de 280 (1 int32 et 6 float32)



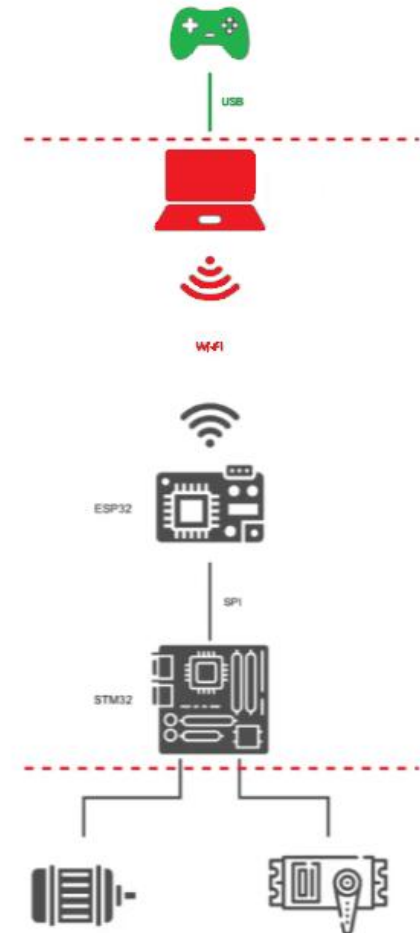
DataStruct défini comme
table de vérité pour
formater et décoder les
paquets UDP par la suite



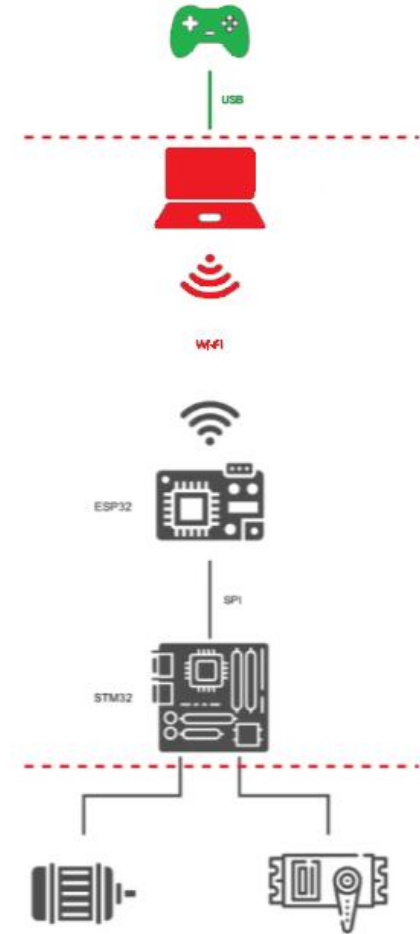
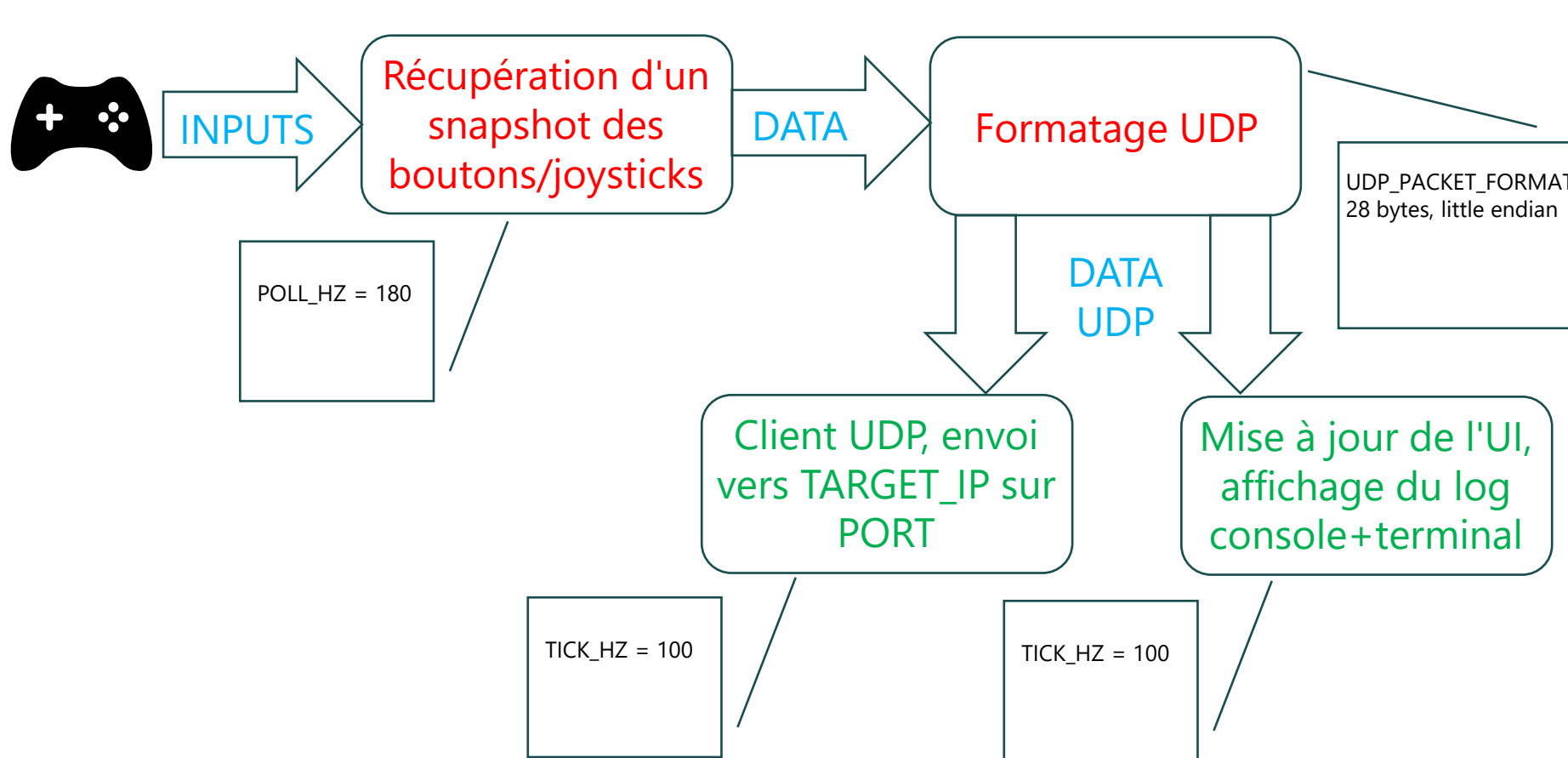
Application (Architecture)



Architecture multi-threadée, orientée objet avec signaux et slots sous PyQt5 Linux.

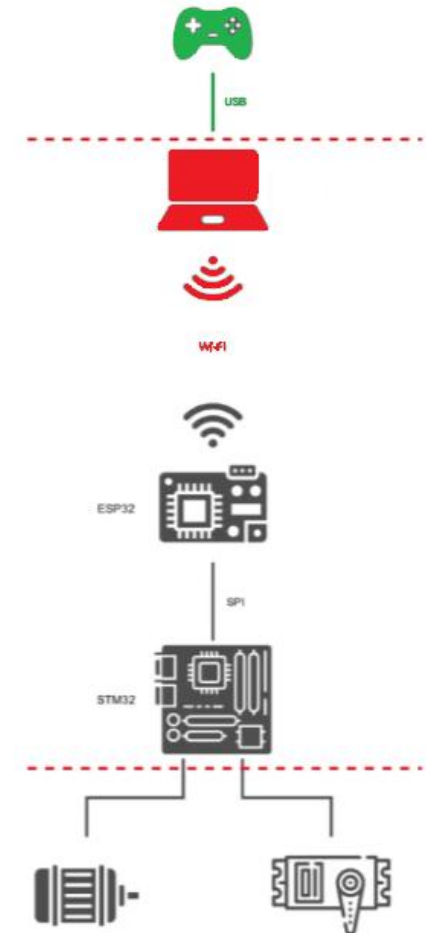


Application (Fonctionnement après implémentation)





**ENSI
CAEN**
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE





Chapitre 2

ESP32

The Phong Douangmanivong / Esteban Vantorre

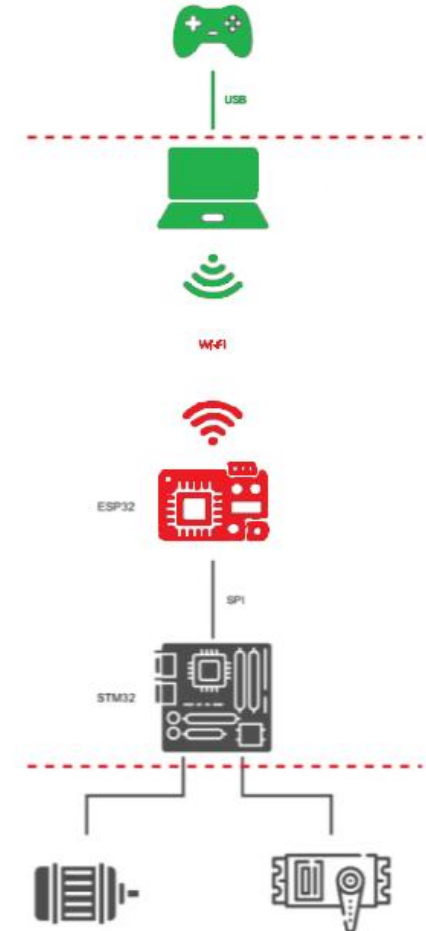
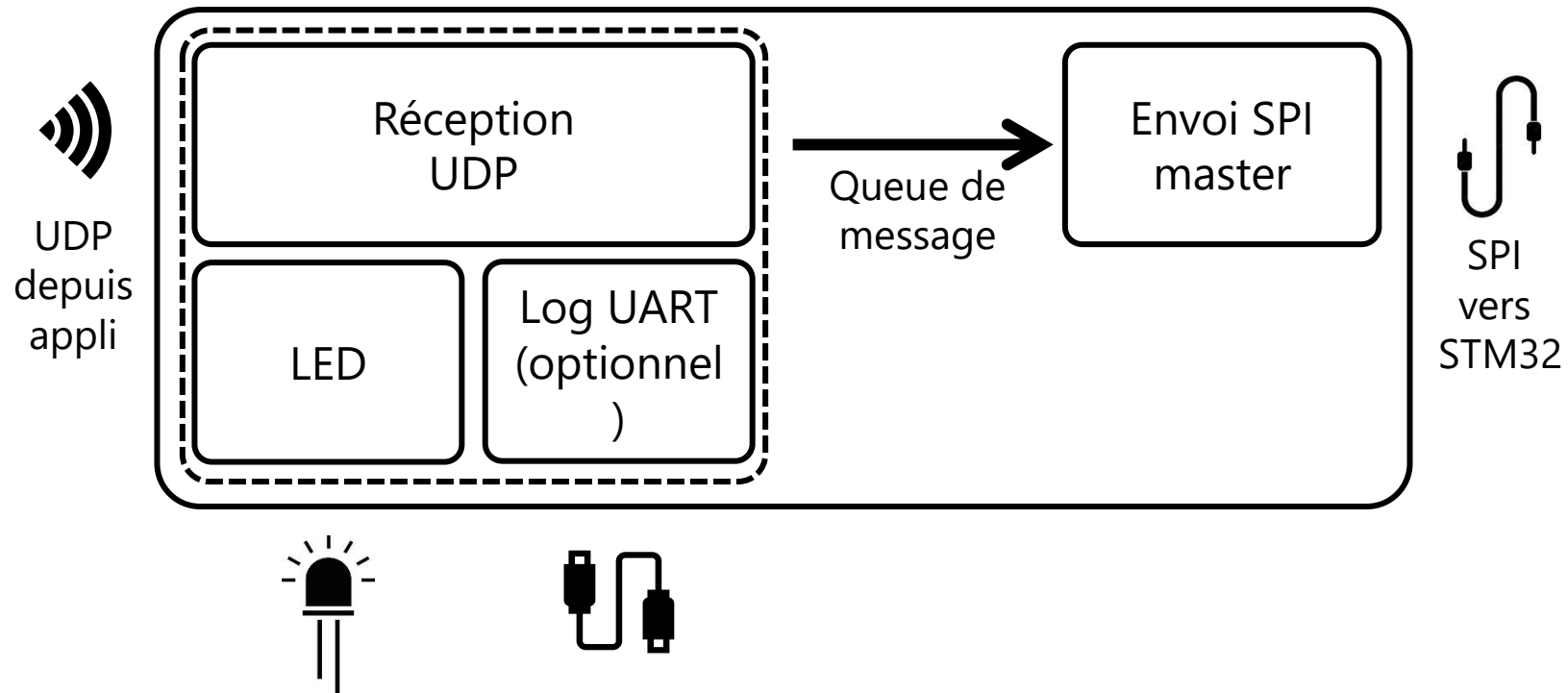
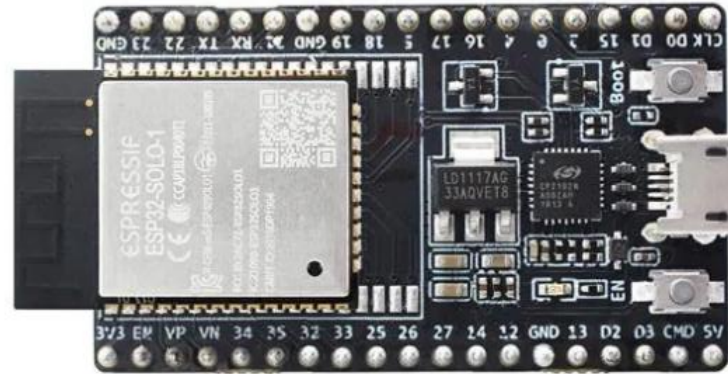
Architecture

Réception (UDP)

Envoie (SPI)

ESP32

ESP32-C3



ESP32

Réception UDP:

- Point d'accès wifi avec SSID et mot de passe
- Serveur UDP recevant les paquets

LED d'indication:

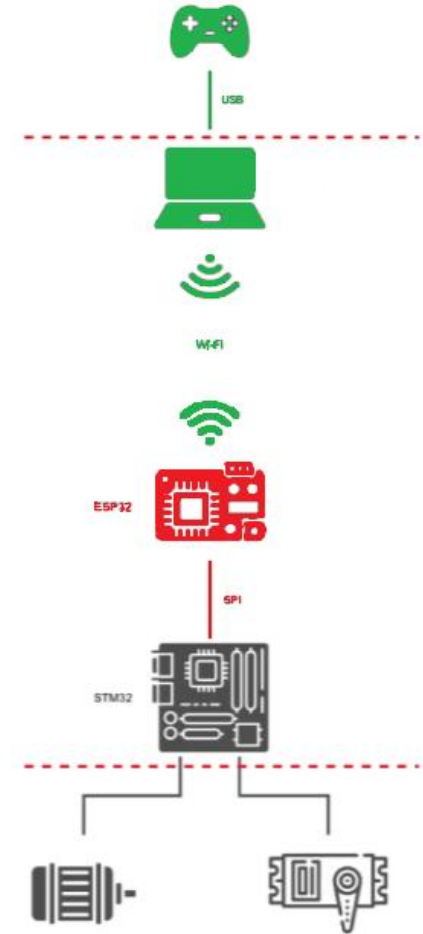
- Jaune au démarrage
- Cyan quand le point d'accès est prêt
- Vert quand un appareil est connecté au point d'accès

Logs UART:

- Informations sur l'état du point d'accès, les paquets reçus etc.

Envoi SPI:

- SPI en master (STM32 en slave)
- Envoie dès l'apparition d'un nouvel élément dans la queue

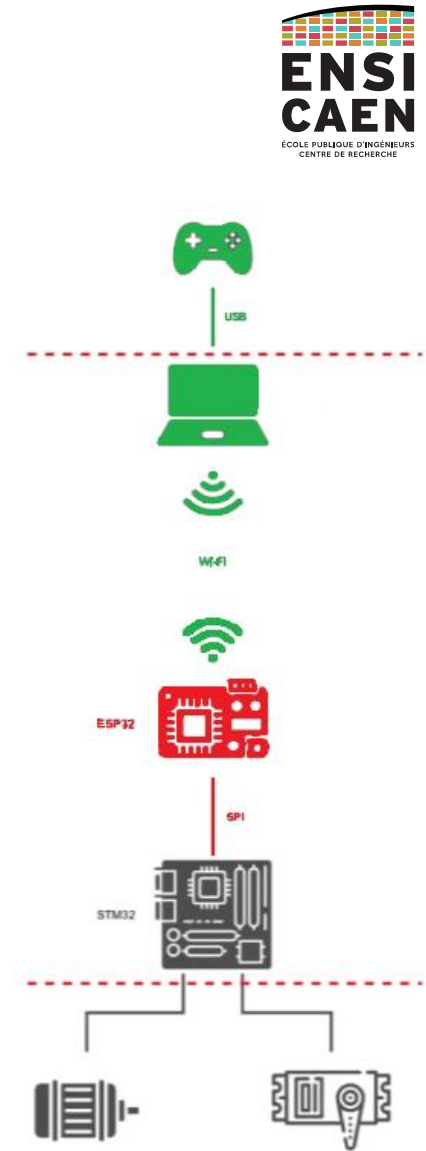


ESP32 (envoi SPI)



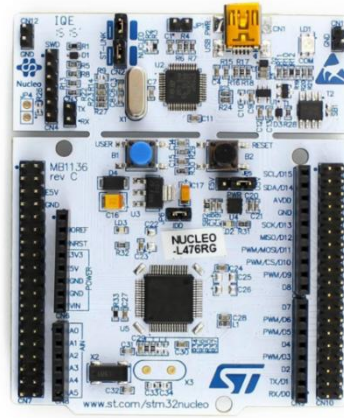
Broches SPI:

- Pin MOSI 2
- Pin MISO 1
- Pin SCLK 0
- Pin CS 5
- Pin HS 3



Chapitre 3

STM32

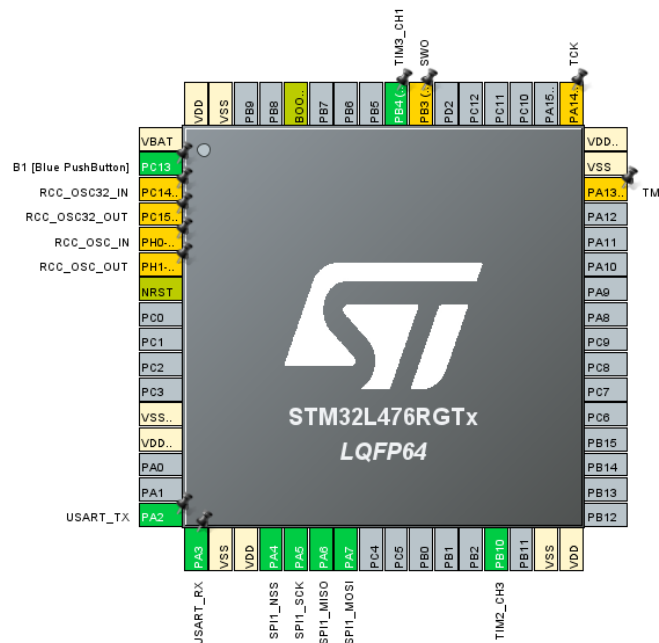


Etienne ROZOY / Loïc RICARD / Thef Douangmanivong
Alix Corbille

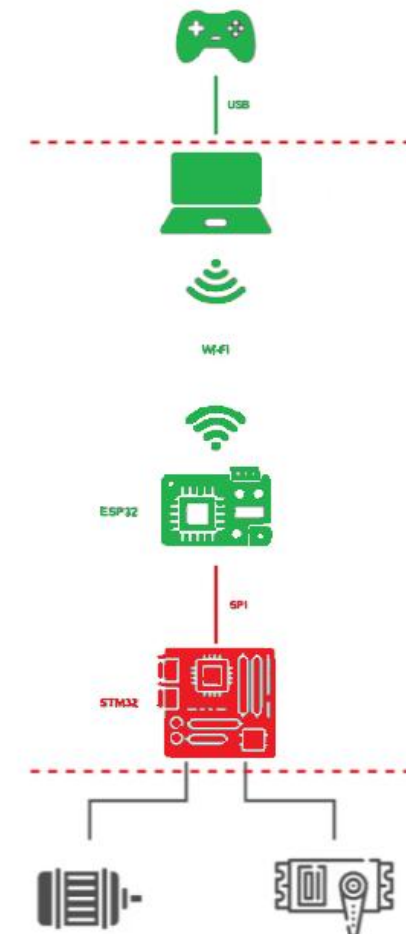
Mappage (pins)

Réception SPI

Signaux PWM (Direction / Vitesse)

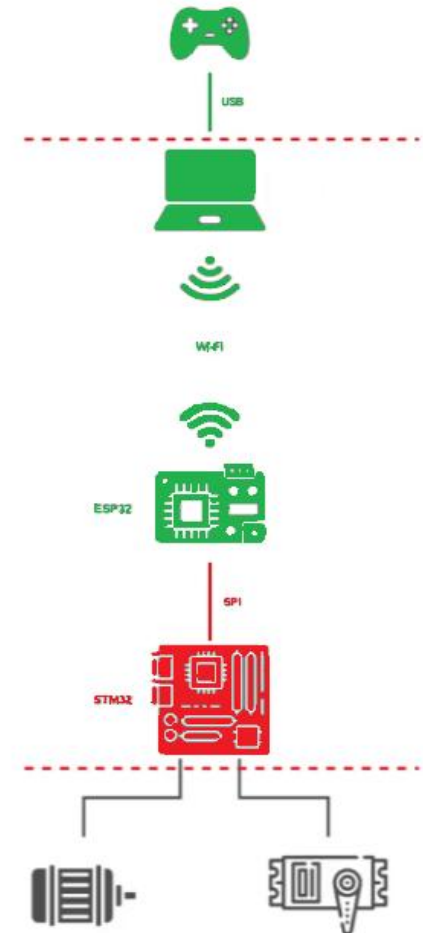
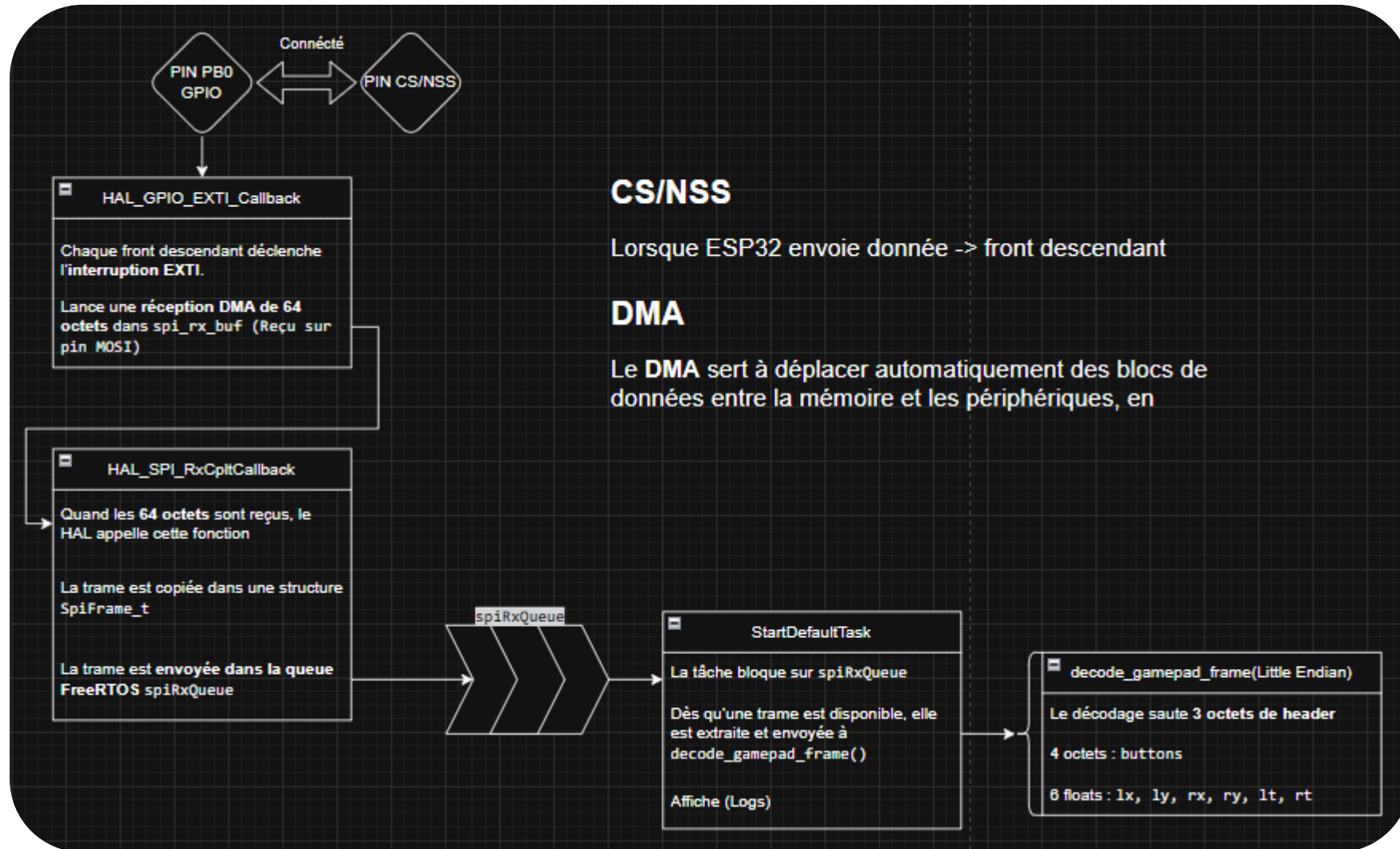


2 pins communication UART (logs)



- **SCK (Serial Clock)** : signal d'horloge généré par le maître pour synchroniser les échanges SPI.
- **MISO (Master In Slave Out)** : ligne de données envoyées du périphérique esclave vers le maître. (Pas utilisé)
- **MOSI (Master Out Slave In)** : ligne de données envoyées du maître vers l'esclave.

STM32 (Réception SPI)



STM32 (Envoie PWM)

Format des données

Button per bit	Ø	Right Joystick	Left Joystick	→	←	↓	↑	Xbox	Select	Share	RB	LB	B	A	Y	X
bit	32 -- 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Buttons	Left Joystick x-axis	Left Joystick y-axis	Right Joystick x-axis	Right Joystick y-axis	LT	RT
32 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits

1 paquet de 28o (1 int32 et 6 float32)

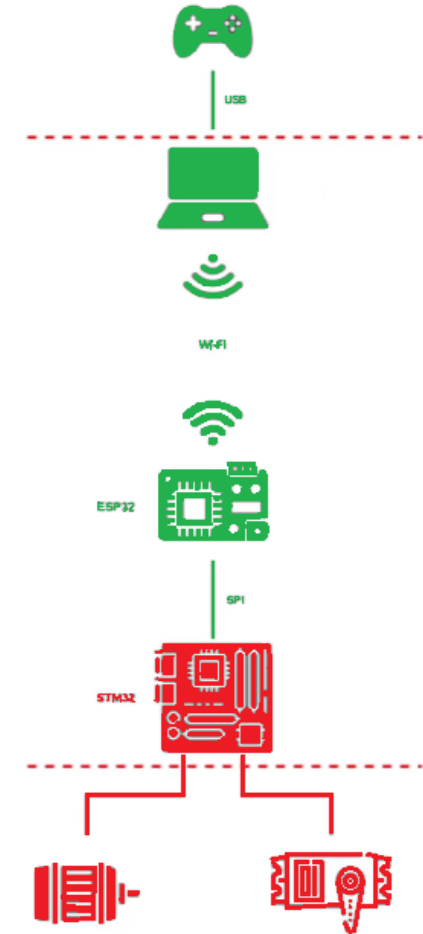
Direction

Reculer

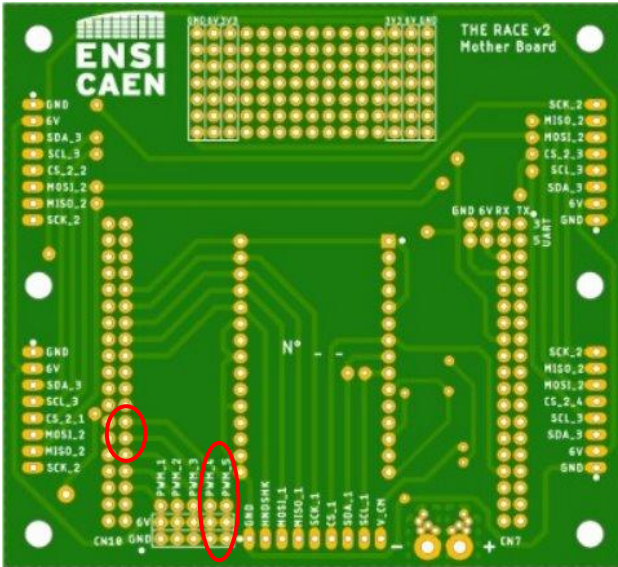
Avancer

Problème : Deux gâchettes contrôlent un seul moteur : conflit d'intérêt

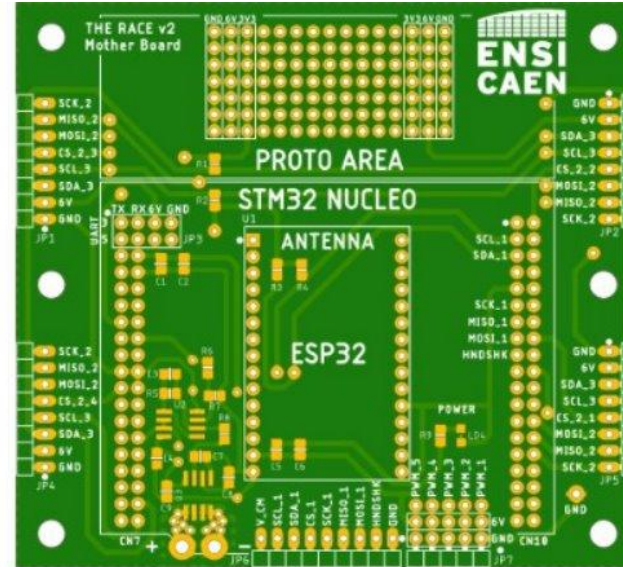
Solution : Sommer leurs valeurs pour éviter les conflits



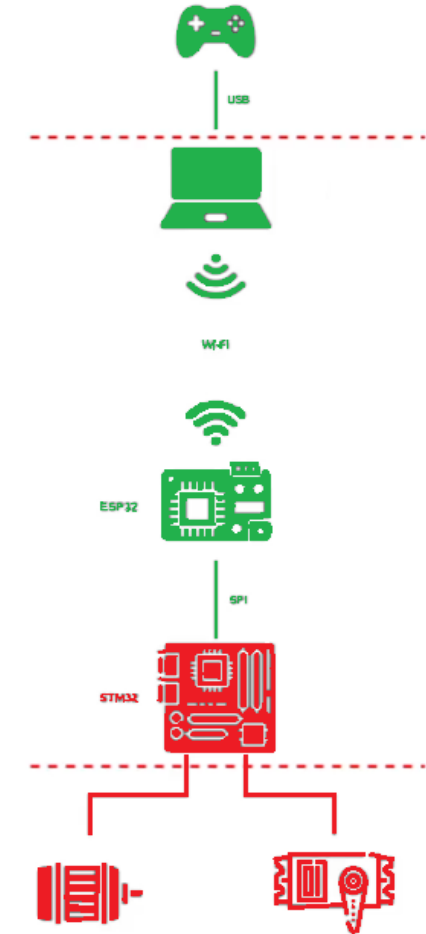
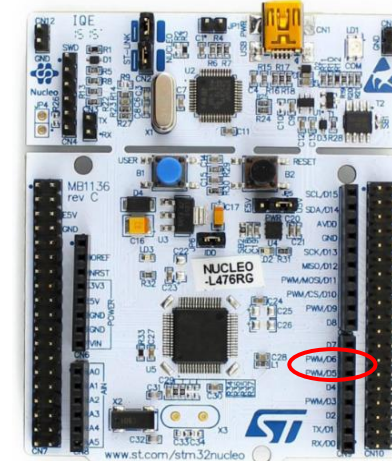
STM32 (Envoie PWM)



Bottom view



Top view

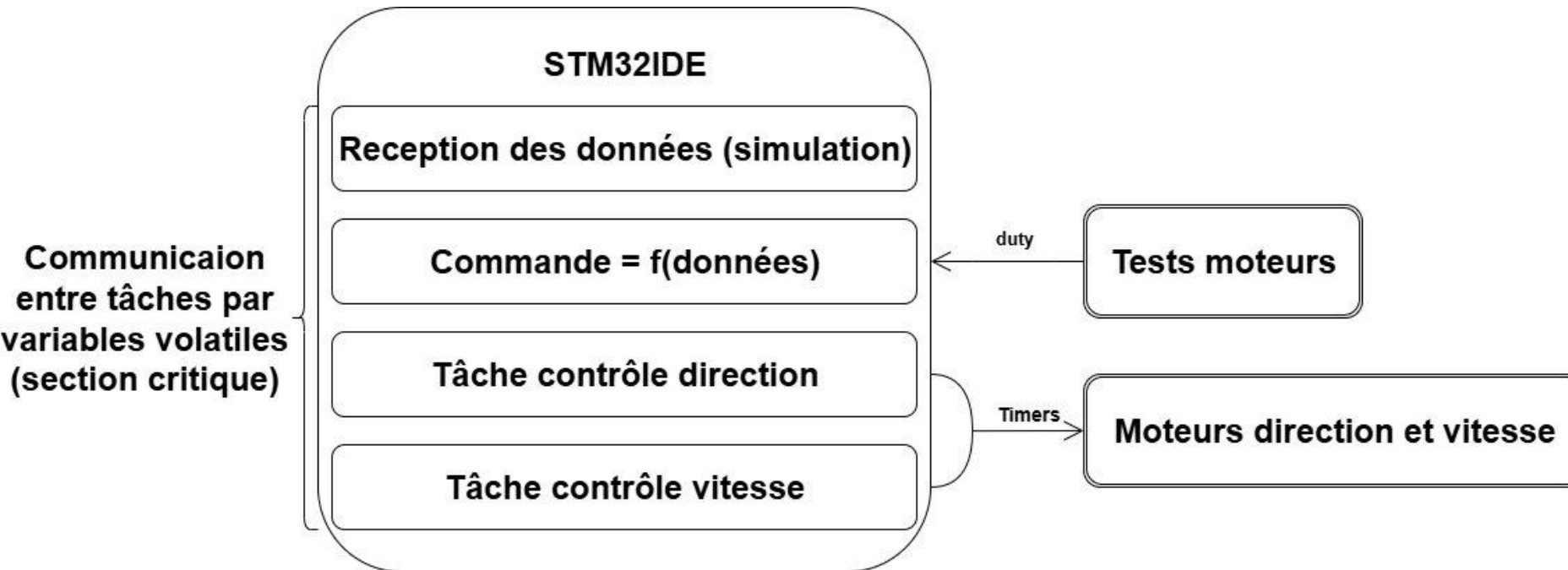


PWM_4 et PWM_5 étaient branchés aux moteurs, la bottom view permet d'identifier les broches de la carte NUCLEO concernées :

- PWM/D6
- PWM/D5

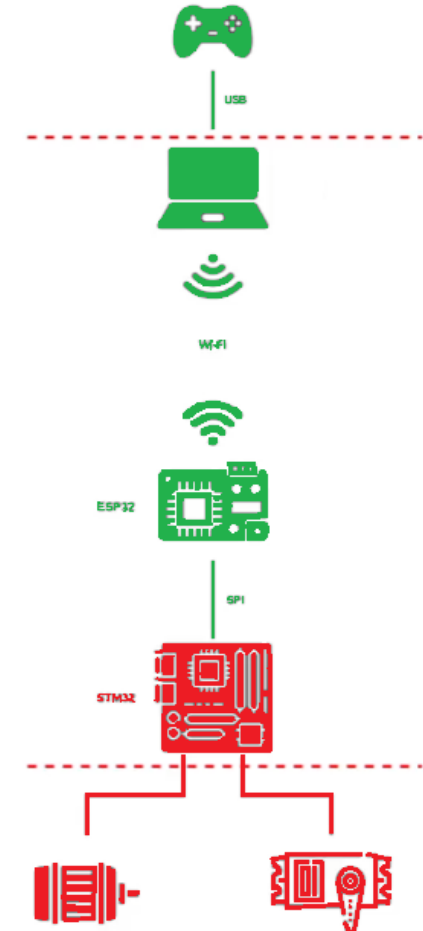
➡ Configuration des timers concernés

STM32 (Envoie PWM)



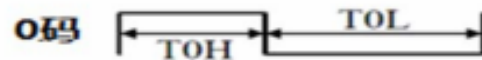
Réalisation des tests moteurs afin de déterminer expérimentalement les duty cycle à imposer pour le bon fonctionnement de la voiture :

- 6 à 8 % pour la direction
- 6.5 à 7.5 % pour la vitesse
- Point mort pour les deux moteurs à 7 %

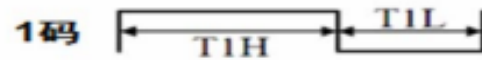


STM32 (Envoie PWM LED)

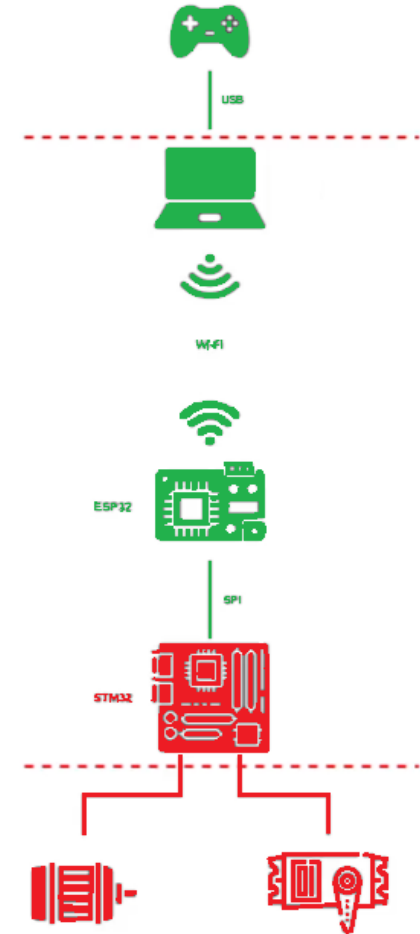
Input code:



avec T0H = 33% et T1H = 66%



Les LED fonctionnent avec un envoi de PWM à 800kHz, chaque bit dure 1.25µs et donc envoyer un 1 revient à envoyer un signal PWM avec un rapport cyclique de 66% (environ 0.8µs) et même chose pour 0 mais avec un rapport cyclique de 33% (environ 0.4µs). Ces données permettent uniquement de choisir quelle LED allumer, pour la couleur on va se servir de DMA (Direct Access Memory).

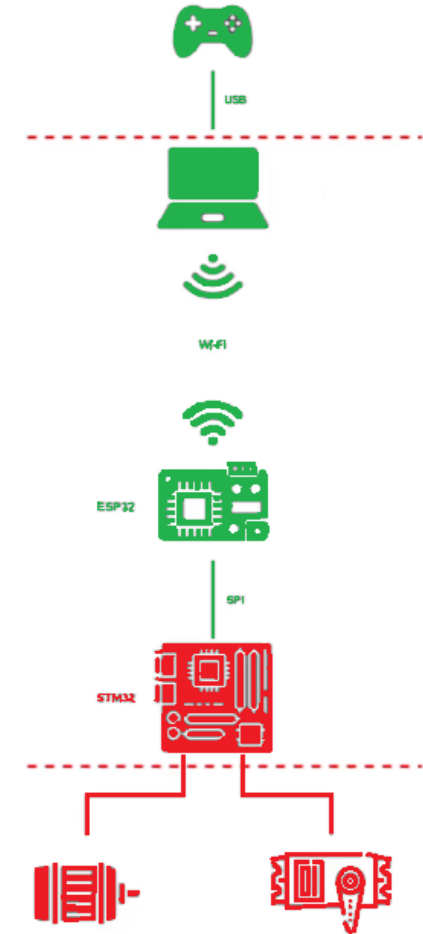
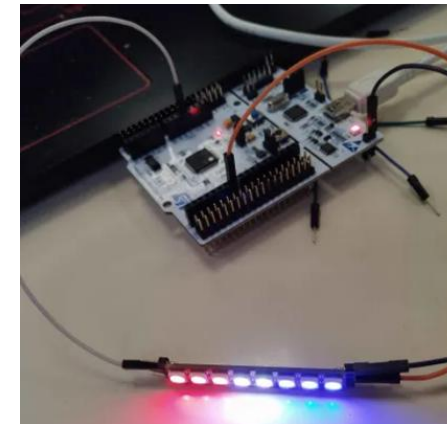
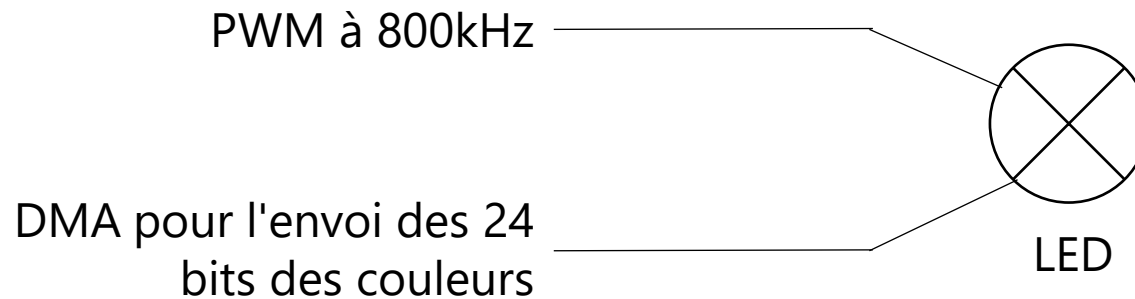


STM32 (Envoie PWM LED)

Format des données pour les couleurs :

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4
R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

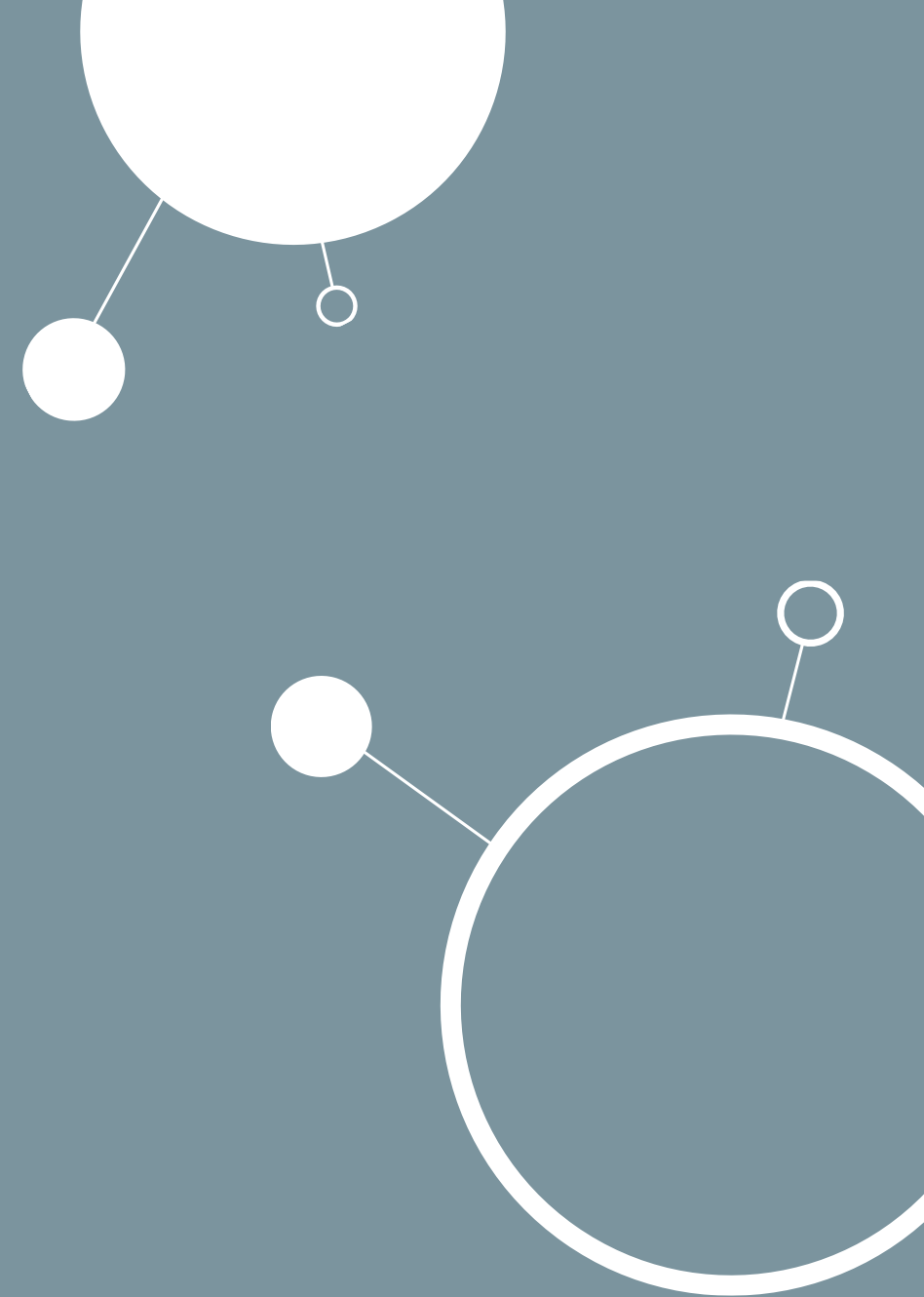
24 bits, 8 pour chaque couleur, stockés dans un buffer



Chapitre 4

INTEGRATION

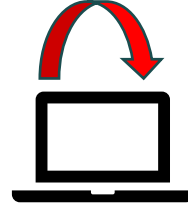
Etienne ROZOY / Loïc RICARD / Thef Douangmanivong
Alix Corbille



Intégration 1 (APP – ESP32)

Phase 1

Test serveur UDP
et envoi local



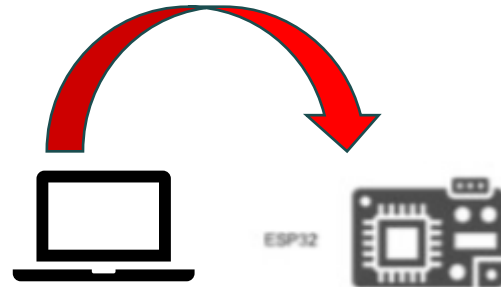
```

PROJECT
douangmanivong@douangmanivong-Latitude-3400: ~/Bureau
Triggers:  LT=1.000, RT=-1.000
-----
[16:12:49] Paquet #5121 reçu de 127.0.0.1:47208 (28 bytes)
RAW BYTES: 00 00 00 00 00 82 57 3f 00 80 3f 3f 00 00 00 00 00 00 00 80 00 fe 7
f 3f 00 00 80 bf
DECODED DATA:
Buttons: 0x00000000 (none)
Left Stick:  X=0.842, Y=0.748
Right Stick: X=0.000, Y=-0.000
Triggers:  LT=1.000, RT=-1.000
-----
[16:12:49] Paquet #5122 reçu de 127.0.0.1:47208 (28 bytes)
RAW BYTES: 00 00 00 00 00 7e 57 3f 00 5e 3f 3f 00 00 00 00 00 00 00 80 00 fe 7
f 3f 00 00 80 bf
DECODED DATA:
Buttons: 0x00000000 (none)
Left Stick:  X=0.842, Y=0.748
Right Stick: X=0.000, Y=-0.000
Triggers:  LT=1.000, RT=-1.000
-----

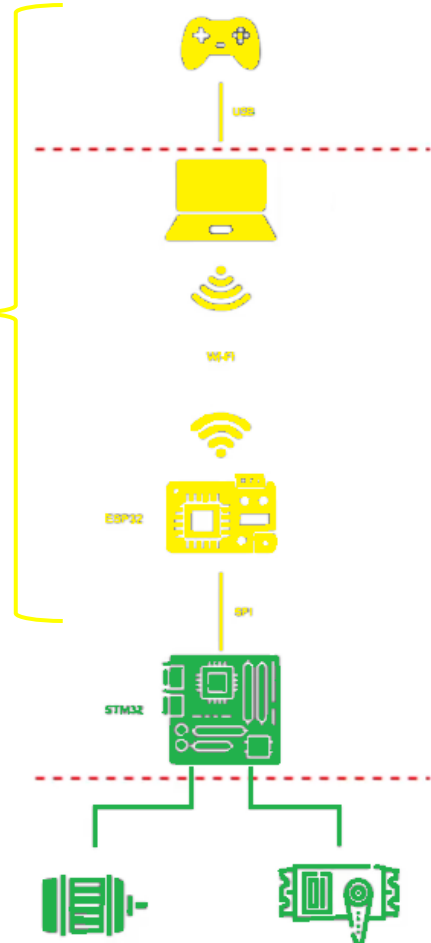
```

Phase 2

Serveur UDP sur
l'ESP avec logs
UART



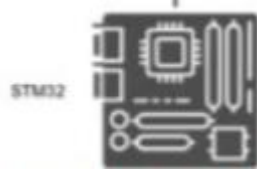
Intégration 1



Intégration 2 (Problèmes)

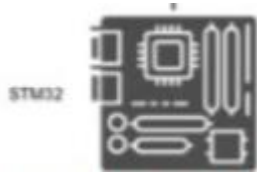
Pb 1

Ajout des timers à l'IOC de réception SPI



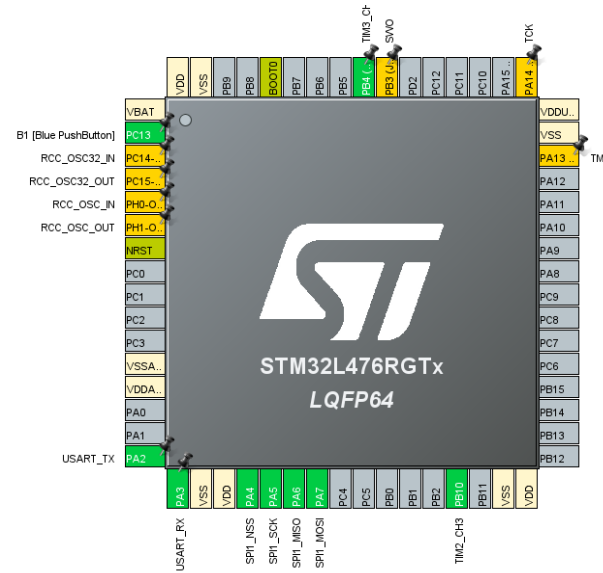
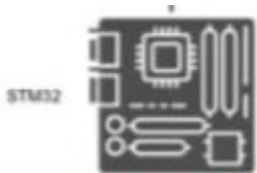
Pb 2

Passage de section critique à queue de message

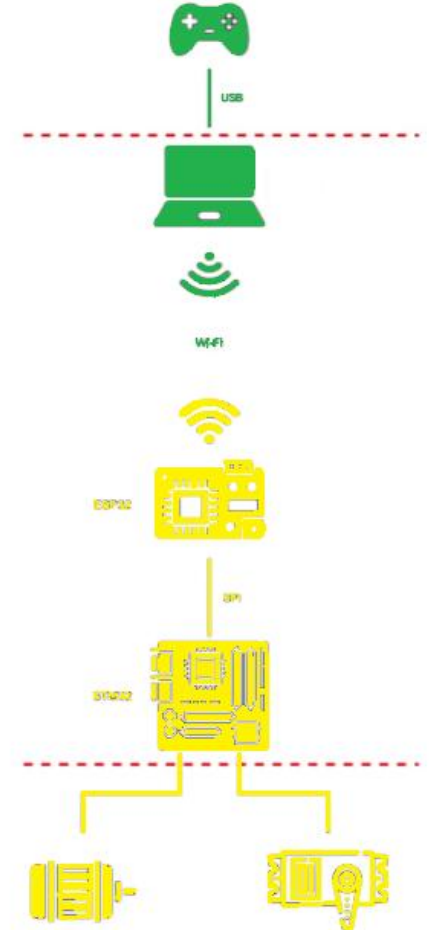


Pb 3

Débuggage après assemblage des tâches



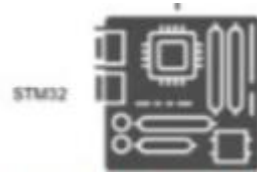
Le projet réception n'avait pas les **timers** et l'implémentation des tâches PWM passaient par **section critique** ce qui rendait la **synchronisation** difficile. La solution est de **modifier l'io** et d'implémenter **2 queues de message** pour la **direction** et la **vitesse**.



Intégration 2 (Problèmes)

Pb 4

Modification de la
taille du heap
freertos



Pb 5

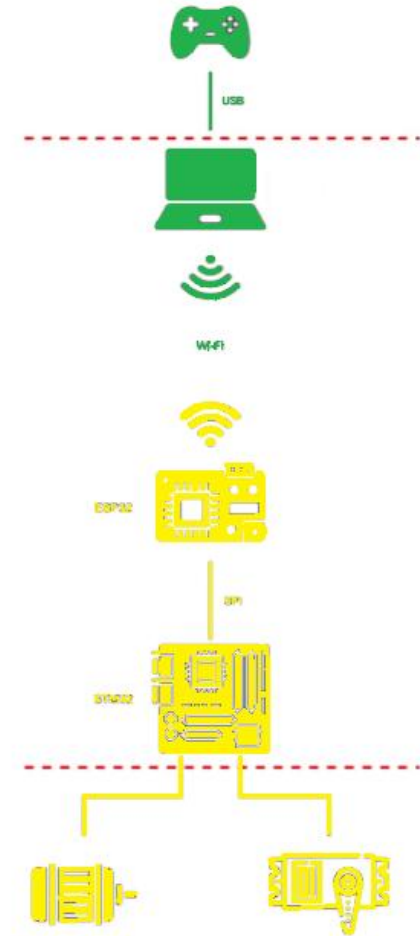
Ajout de log uart
et clamping du
duty PWM



```
#define configTOTAL_HEAP_SIZE ((size_t)(20*1024))
```

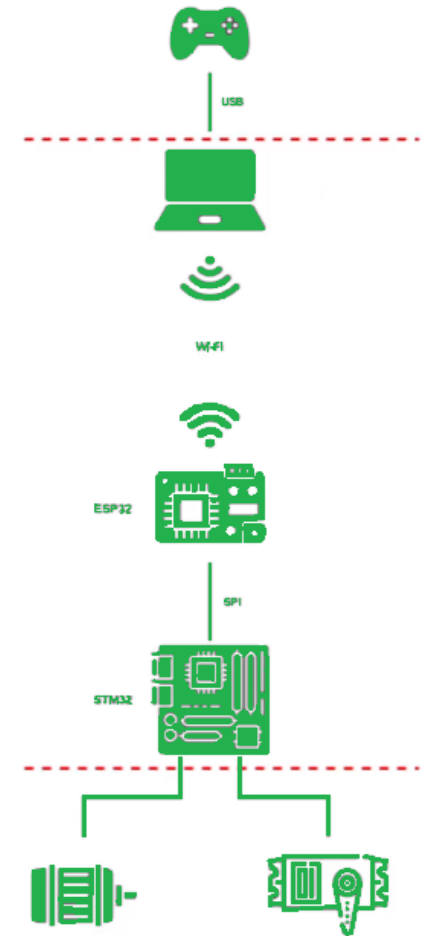
Les **logs UART** ont mis en avant le fait que les tâches autres que la réception SPI n'étaient pas créés. La solution a été de **passer de 3ko à 20ko**.

Pour éviter que les **signaux PWM** fournis aux moteurs n'**endommagent la voiture**, il a été décidé de **limiter les valeurs de duty**.



Intégration (Produit final)

VOIR VIDEO SUR LE GIT :D



Merci d'avoir lu

Etienne ROZOY / Loïc RICARD / Thef Douangmanivong
Alix Corbille / Esteban Vantorre



En espérant que cela a pu vous aider