



Stage Eirlab, contrôle d'un hoverboard

Antony THIERY

June 2024

Chapter 1

Important

1. Une grande partie du code a été écrit et addapté par M.BOUSSICAULT Adrien
2. Le principal problème pour la suite est l'asservissement des moteurs qui ne se comportent pas de la même manière si les deux tournent dans le même sens ou dans un sens opposé.
3. L'hoverboard 1 a souffert à 2 reprises : un court-circuit contre l'armature et une surchauffe des transistors. Celui-ci ne démarre plus pour le moment, mais peut en principe être réparé. (Détails section 3.2.1 Problème sur l'hoverboard 1). L'hoverboard 2 fonctionne sans problème.
4. coté R = coté batterie, coté L = côté carte mère
5. Si les deux moteurs tournent à pleine vitesse et qu'il ne s'arrête pas quelle que soit la commande envoyée, alors il faut éteindre l'hoverboard, envoyé une commande neutre via le terminal (1550,1550) et redémarrer l'hoverboard.
6. Si le bouton ON/OFF a été maintenue trop longtemps, l'hoverboard s'est probablement recalibré. La commande neutre n'est alors plus (1550,1550).
7. Les gyroscopes ont été détachés de l'hoverboard pour permettre de communiquer à leur place.
8. La radio est facultative, tout fonctionne sans.

9. En analysant la difficulté que l'on a à faire tourner les moteurs de l'hoverboard à la main, on peut repérer un court-circuit. La roue L de l'hoverboard 1 a peut être un court-circuit sur son fil bleu.
10. Les fils reliés à rien sont des masses pour faciliter les mesures.

Chapter 2

État actuel du projet

2.1 Matériel

1. deux hoverboards complets démontés
2. Une carte esp32
3. Codes de la carte esp32: filtre.ino, filter.hpp, queue.hpp
4. Codes d'analyse du comportement : lecteur.py, Serie.py, identification.py, solve_linear_equation.py
5. Codes de calcul du filtre de l'esp32 : synthesis_of_analogic_and_digital_filters.py

2.2 Objectifs

L'objectif du projet et de récupérer la carte de commande et de puissance de l'hoverboard pour prendre le contrôle sur les moteurs, dans le but de réaliser une plateforme holonome.

2.3 État actuel

Il est pour le moment possible d'imposer une vitesse aux roues d'un des hoverboards via le port série. Il est également possible de récupérer la vitesse de chaque roue grâce au fichier Serie.py. Une fois le fichier lancé, la séquence programmée de vitesse pour la roue se lance. Une fois l'acquisition terminée, La courbe de vitesse est affiché et est enregistré dans output.csv.

Les données enregistrées dans output.csv peuvent ensuite être analysés grâce aux fichiers lecteur.py, identification.py, solve_linear_equation.py.

Le filtre interne de l'esp32 peut être recalculé à l'aide de synthesis_of_analogic_and_digital_filters.py.

2.4 Exemple d'utilisation

1. Attacher les moteurs à un support fixe (ça pousse)
2. Alimenter l'hoverboard en 36V (les alims 32V font l'affaire)
3. Brancher l'esp32 au port série de l'ordinateur. (de préférence COM10)
4. Ouvrir un terminal série (Termite 3.4 par exemple)
5. Taper help pour la liste des commandes, mettre la vitesse des deux moteurs sur neutre (velR 1550 puis velL 1550)
6. Reset l'esp32 (bouton EN)
7. Dans le fichier Serie.py, décrire la séquence de commande que l'on veut effectuer dans la liste L_modifR (une nouvelle commande sera envoyé toutes les periode_nouvelle_commande en secondes)
8. Lancer le script python, on obtient le fichier output.csv contenant l'expérience.

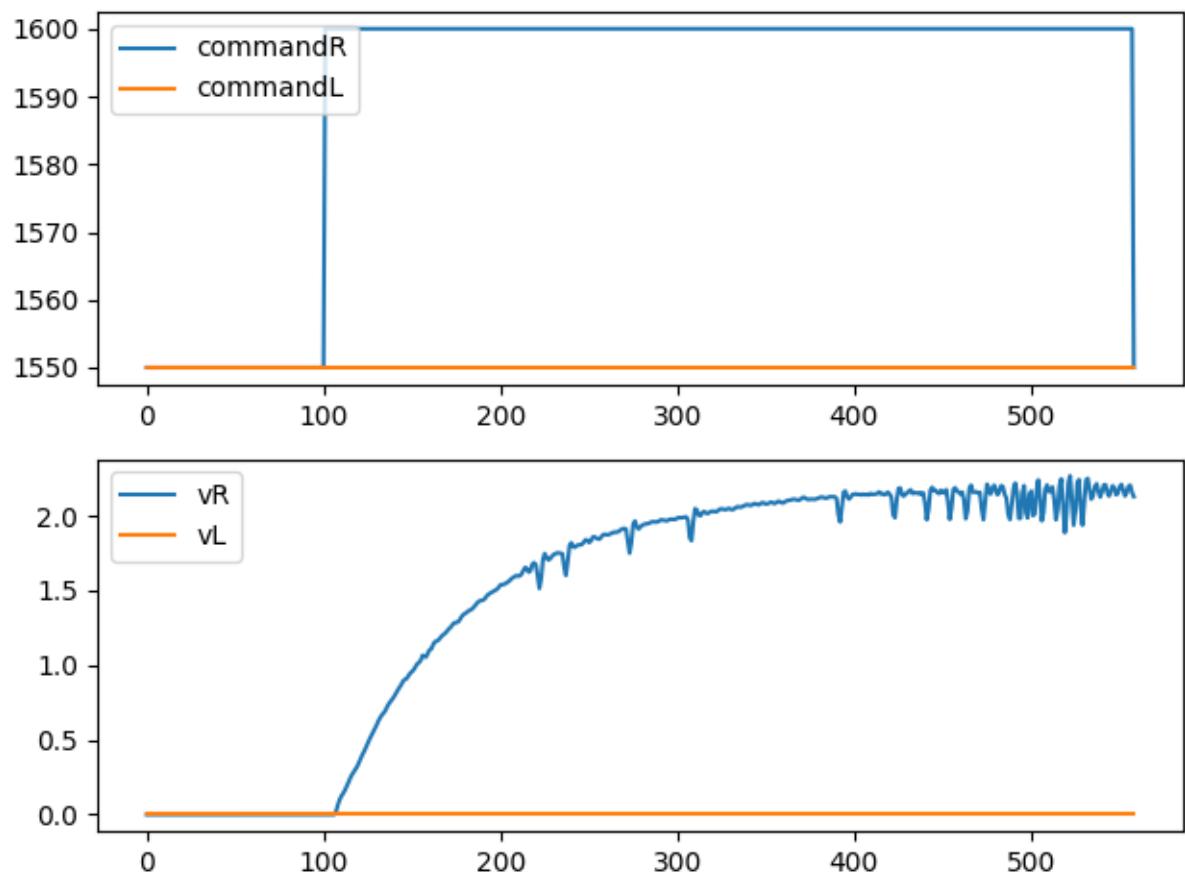


Figure 2.1: Exemple d'expérience pour une commande de type échelon

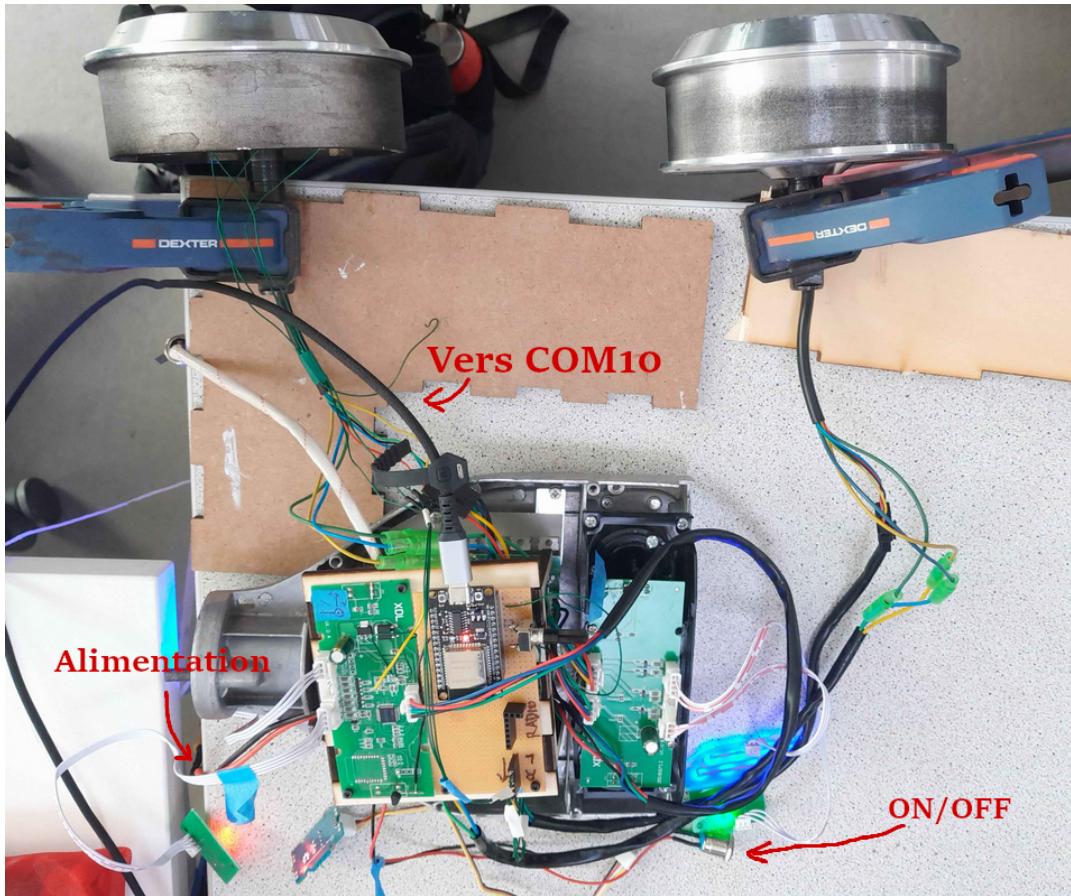


Figure 2.2: Montage

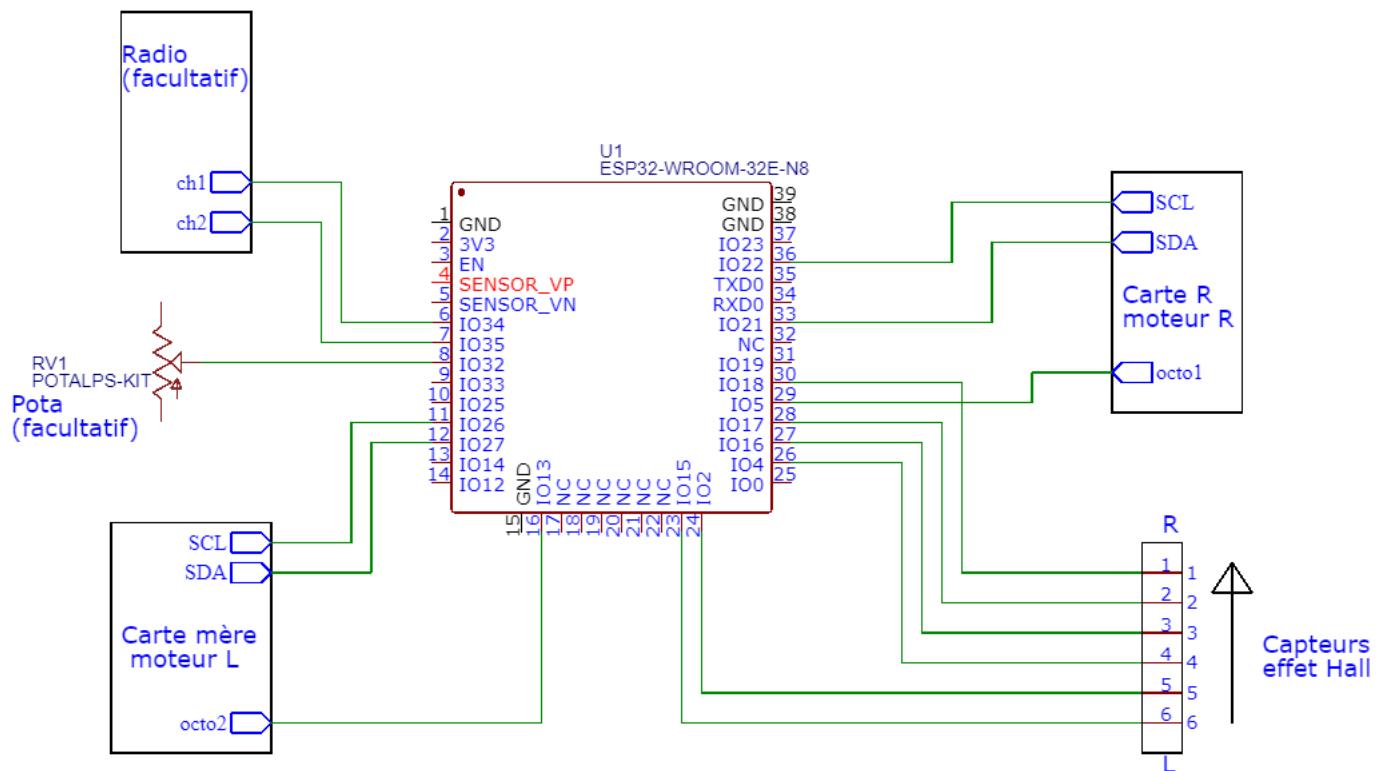


Figure 2.3: Plan carte jaune (simplifié)

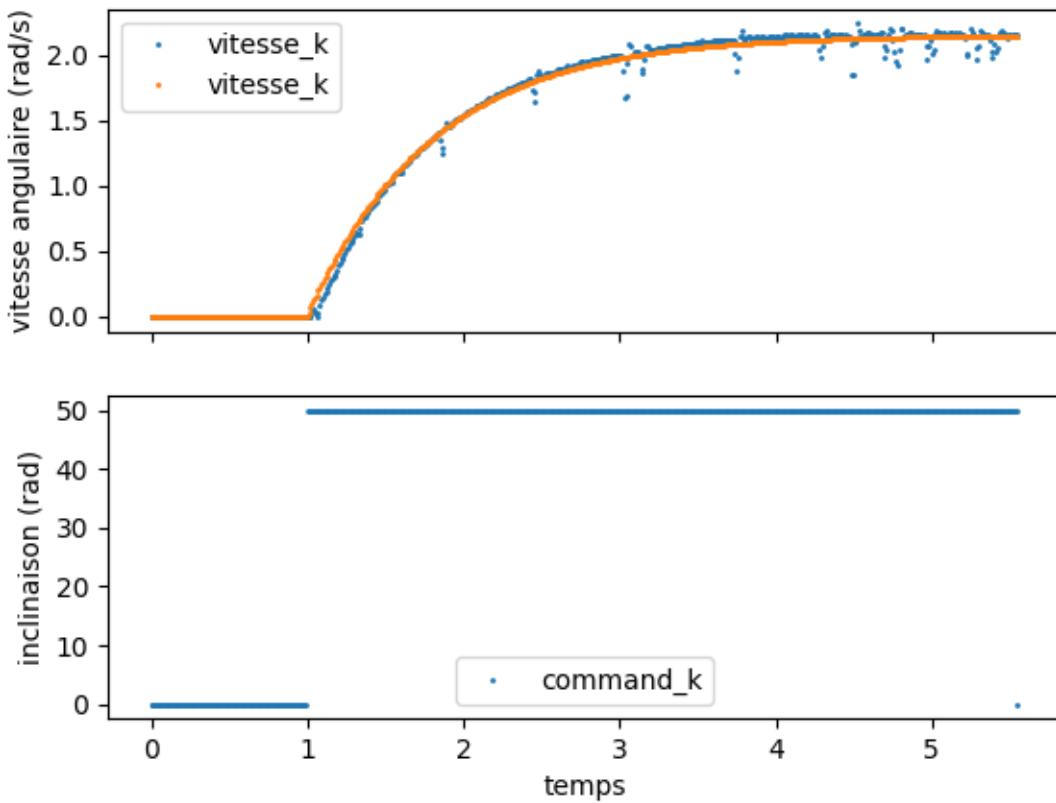


Figure 2.4: Résultat de l’analyse de l’expérience pour un modèle d’ordre 3 (données bleues, modèle en orange)

2.5 Exemple d’analyse de l’expérience

1. Ouvrir le script lecteur.py pour un exemple d’analyse sur un modèle d’une roue
2. Lancer le script, celui-ci s’exécute en analysant les données de output.csv.
3. Vérifier que l’erreur rms est faible devant les valeurs de sorties
4. On obtient les matrices d’identification E et F.

Le fichier lecteur.py n’est qu’un fichier d’exemple d’analyse pour un moteur à la fois. Pour une analyse plus complexe, il faut se référer à identification.py

Chapter 3

Compte rendu détaillé chronologique

3.1 Démontage

Démontage et explication brushless hoverboard de même modèle

3.1.1 Carte R (cote batterie)

accéléromètre/gyrosopes + gestion voyant batterie + optocoupleur

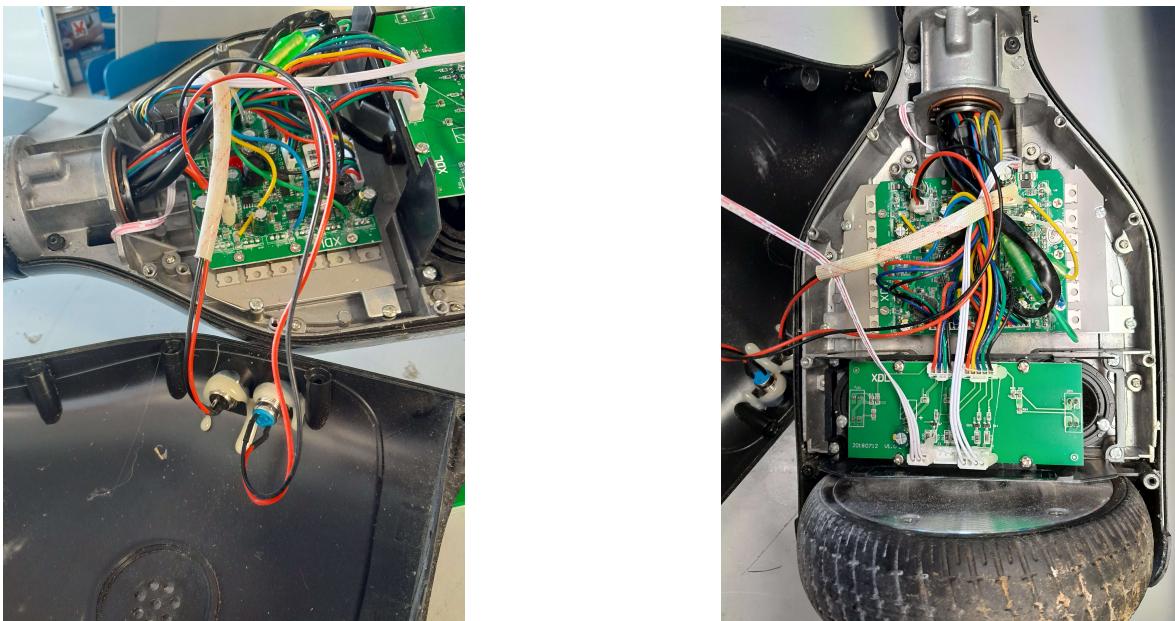
3.1.2 Carte L (cote carte mère)

gestion voyant carte mère + optocoupleur

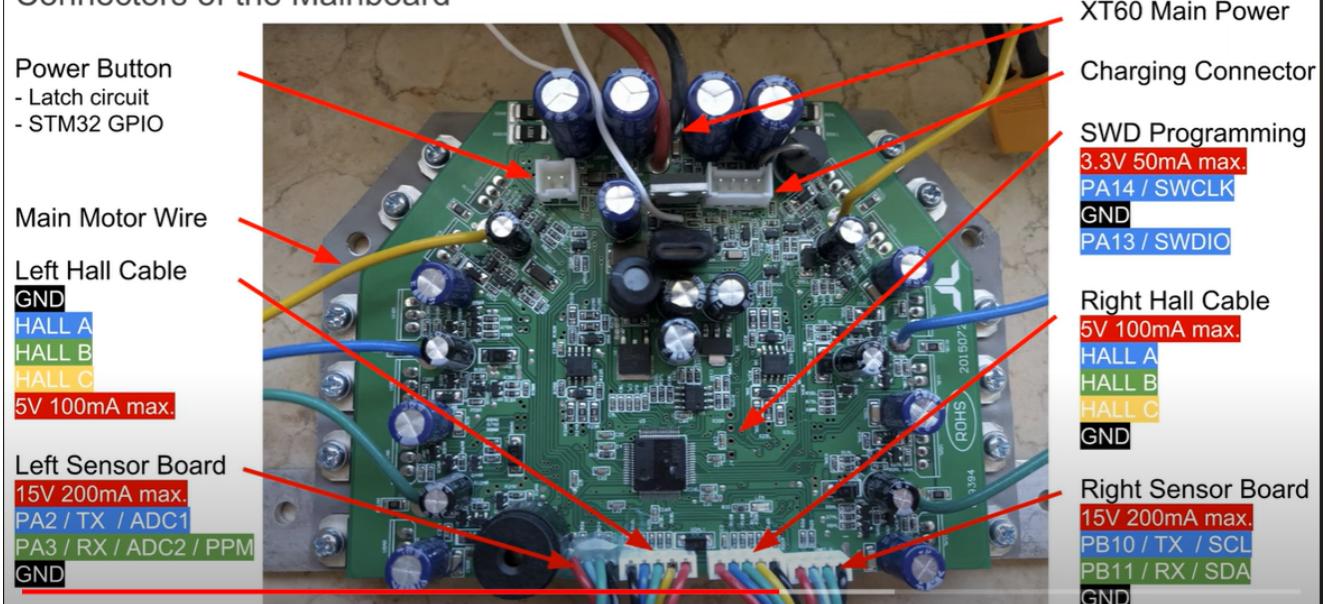
3.2 Fonctionnement des hoverboards

Attention, notre hoverboard ne correspond pas aux modèles classiques que l'on trouve sur internet. Il est donc risqué de tenter un flash sur les microcontrôleurs.

[tuto écrit, tuto pour le hack, mais mauvais modèle](#)
Le microcontrôleur principal de la carte mère n'est pas comme dans les autres hoverboards un STM32F103 mais un GD32F103. Ces composants sont quasiment identiques, mais il existe cependant des [problème de portabilité](#) dans le cas d'un flash.



Connectors of the Mainboard



Il semble que l'on ne puisse pas récupérer le code du STM pour le mettre dans le GD32F103 cependant les cartes plus récentes sont équipés du GD mais ce ne sont pas les mêmes cartes.

3.2.1 Problème sur l'hoverboard 1

1. Court-circuit :

Court-circuit de la batterie sur l'hoverboard 1 avec le dessous du circuit en contact à l'armature, l'hoverboard s'est éteint instantanément produisant une étincelle, l'hoverboard réagit toujours en branchant la batterie, mais le bouton on/off ne semble plus fonc-

tionner.

De plus, le microcontrôleur de la carte R1 a un court-circuit entre une de ses pattes 3.3V et la masse. A priori les autres composants vont bien. Le problème venant du microcontrôleur de la carte R1, sachant que nous ne possédons pas le code de celui-ci, je décide de m'occuper plus tard de ce problème et continue avec la carte R2.

2. MOSFET grillés :

En testant le fonctionnement des moteurs, certains MOSFET semblent avoir grillé, la carte n'étant plus posée sur sa base refroidissant. La carte ne démarre plus. [Ref des mosfets supposément grillés](#). Je teste donc la diode interne de ces MOSFET ainsi que la résistance Vds après avoir déchargé les grilles.

Solution provisoire : Je passe sur l'hoverboard 2

3.2.2 Démarrage de l'hoverboard 2

Pour démarrer, les gyroscopes doivent être parallèles au sol et dans le bon sens sinon le buzzer sonne.

Pour le côté L des hoverboards, le gyroscope est situé dans la carte mère. Ainsi, les deux moteurs fonctionnent correctement et sont contrôlés en vitesse. Lorsque l'hoverboard est positionné de telle sorte à ce qu'il avance ou recule, même à petite inclinaison, il fait saturer ses moteurs à vitesse maximale. Ce fonctionnement est normal, car l'hoverboard accélère jusqu'à ce que l'humain se trouvant dessus se retrouve de nouveau à la verticale.

3.2.3 Conclusion sur le fonctionnement des hoverboards

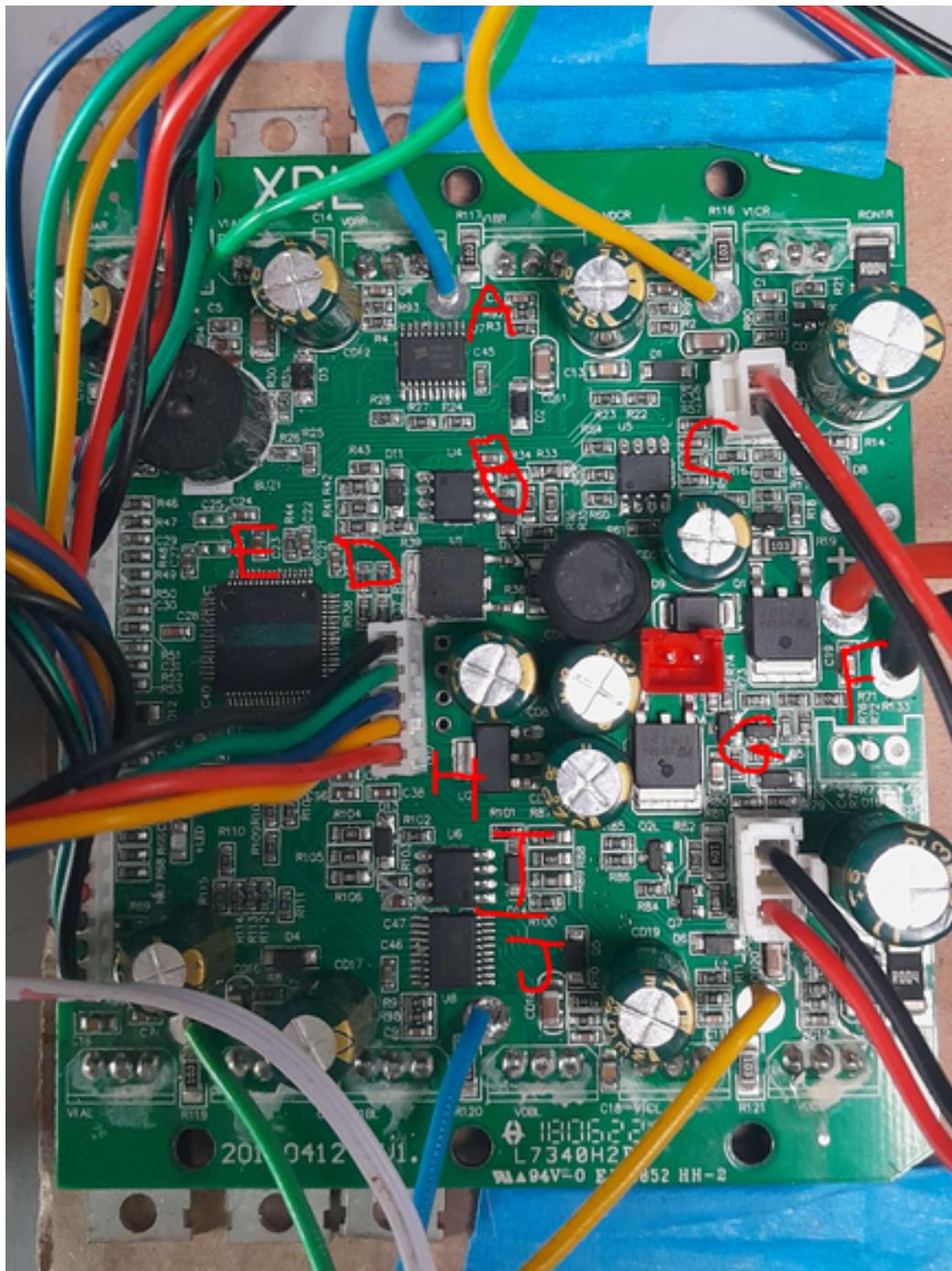
L'hoverboard 2 fonctionne correctement. L'hoverboard 1 ne peut pour le moment pas démarrer.

3.3 Composants de l'hoverboard

3.3.1 composants de la carte mère

1. A: FORTIOR FD6287T FD1039 [surement ce gate driver ,la compagnie](#)

2. B: LS9632 1841 (AOP)
3. C: LS9632 1841 (AOP)
4. D: 78M05 A1718 (voltage régulateur)
5. E: GD32F103 RCT6 CE8B6I5 AC1815 ARM (Microcontrôleur)
6. F: transistor
7. G: transistor
8. H: transistor
9. I: LS9632 1841 (AOP)
10. J: FORTIOR FD6287T FD1039 *surement ce gate driver*
11. en dessous E: IC2681 H21661 1447 (gyroscope)



3.3.2 composants carte R

1. MM32F031 C6T6 735627-n4 63F1 811 peut être ça ,datasheet en anglais (pas exactement identique)
2. IC2681 716AA1 1749 (gyroscope), recherche du modèle:
hypothèse 1, map registre Selon la datasheet, nous nous intéressons

à la rotation selon y.(mauvais registre, bon i2c adress, bon nombre de pins, bon branchement, bon axe)

hypothèse 2 (mauvais registres, mauvais nombre de pins, bon i2c adress)

hypothèse 3(bon nombre de registre)

hypothèse 4(mauvais nombre registre)

3.4 Parler à la place des gyroscopes

L'idée est de communiquer à la carte mère des informations semblables à celles qui seraient envoyés par la carte R, nous pourrions contrôler le moteur R sans avoir besoin de flasher la carte.

3.4.1 Gyroscope R

Coté R le gyroscope se trouve dans la carte R, à l'inverse du côté L. Il est donc plus facile d'intercepter la communication entre le gyroscope et la carte mère.

3.4.2 Communication entre carte R et carte mère

On utilise l'oscilloscope Rhode Schwarz pour plus de précision et pour sa capacité à décrypter les protocoles de communication. On intercepte les signaux entre la carte R et la carte mère.

Les fils de données sont connectés aux pins PA2 et PA3 du MM32F031. (que dans la datasheet en chinois Ces pins correspondent à une communication UART avec PA2 = UART2_TX/TIM2_CH3 et PA3 = UART2_RX/TIM2_CH4)

Après vérification, la communication semble être une communication UART en 9 bits sans bit de parité.

Cette piste n'a pas plus été explorée pour le moment.

3.4.3 Étude des signaux envoyés par le gyroscope R

Je détache je buzzer de la carte mère, car je vais devoir travailler avec la carte R à l'endroit.

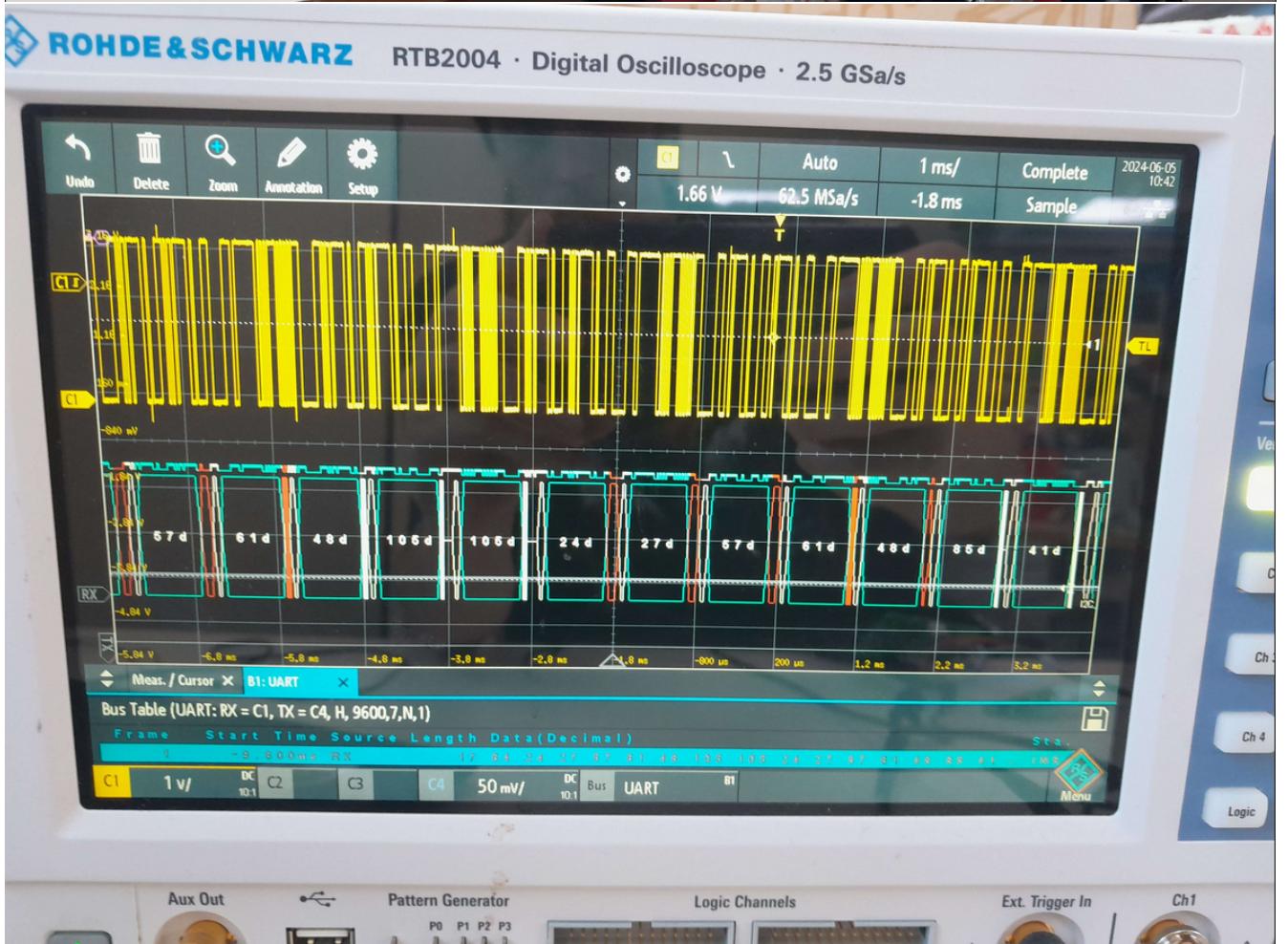
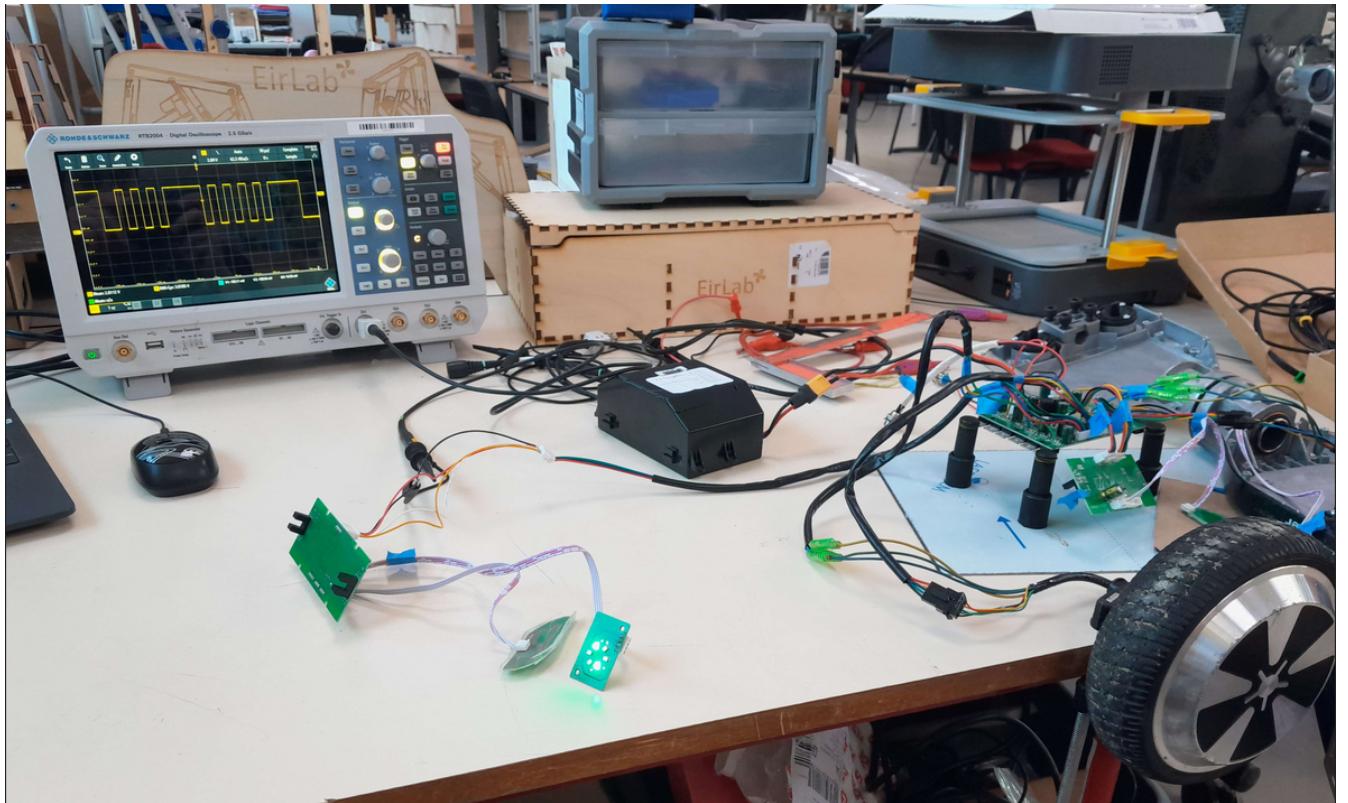


Figure 3.1: communication uart entre carte R et carte mère
14

3.5 Signaux de sortie du gyroscope R

Nous repérons 2 pins de sortie du gyroscope R qui communiquent en I2C.

3.5.1 MPU-6050/MPU-6000

Nous faisons l'hypothèse que le composant C2681 716AA1 1749 est le gyroscope [MPU-6050 ou MPU-6000](#) avec une communication I2C.

Plus tard, nous nous rendrons compte qu'il ne s'agit pas du MPU mais d'un autre composant remplissant la même fonction avec une disposition des pins de sortie à priori identique.

La section 9.2 de la datasheet nous apprend que le MPU est toujours esclave. Selon l'interception de communication, il aurait donc pour adresse sur le bus i2c "0x68".

(L'adresse I2C du MPU-6050 peut-être l'une des deux valeurs possibles, en fonction de la configuration de la broche AD0 : Si la broche AD0 est connectée à la masse (GND), l'adresse I2C est 0x68. Si la broche AD0 est connectée à la tension d'alimentation (VCC), l'adresse I2C est 0x69.)

[register map pour la communication i2c](#)

Côté MM le fil SCL est relié à PB6 et le fil SDA est relié à PB7. Ces pins peuvent correspondre à une communication I2C.

On a ci-dessous plusieurs interceptions de la com i2c du MPU pour plusieurs inclinaisons de la carte. On remarque les deux premiers bits des données semblent changer en fonction de l'inclinaison. (le blocage de la roue n'influe pas la communication)

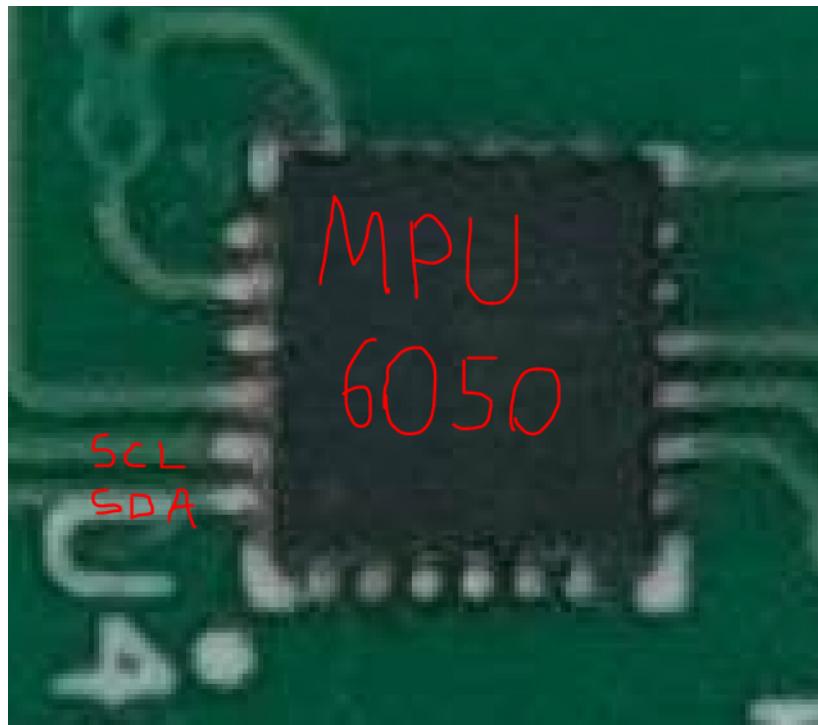


Figure 3.2: Branchement du gyroscope



Figure 3.3: Communication normale entre le MPU R et le microcontrôleur R.

À l'envers

D	E	F	G	H	I	J
SDA = C1)	Type	ID	Length	Data	State	
3 Write	0x68		2	0x7F00	OK	
4 Write	0x68		1	0x2D	OK	
5 Read	0x68		14	0xFD00598BBC0FFF5FFDA005D0730	Data Error+Stop	
6 Write	0x68		2	0x7F00	OK	
7 Write	0x68		1	0x2D	OK	
8 Read	0x68		14	0xFE90058BB68FFE0FFB300500680	Data Error+Stop	
9 Write	0x68		2	0x7F00	OK	
10 Write	0x68		1	0x2D	OK	
11 Read	0x68		14	0xFE6005E8BC00FFDFFF99004F0700	Data Error+Stop	
12 Write	0x68		2	0x7F00	OK	
13 Write	0x68		1	0x2D	OK	
14 Read	0x68		14	0xFD00640BC70FFD9FF8A00490720	Data Error+Stop	
15 Write	0x68		2	0x7F00	OK	
16 Write	0x68		1	0x2D	OK	
17 Read	0x68		14	0xFE100628BC00FFE0FF6F00490680	Data Error+Stop	

À l'endroit

B	C	D	E	F	G	
(I2C: Clock SCL = C2, Data SDA = C1)	Mark	Start Time in s	Type	ID	Length	Data
1		4.35E-02			6	0xFFFFDF0730
4		4.59E-02	Write	0x68	2	0x7F00
5		4.61E-02	Write	0x68	1	0x2D
6		4.63E-02	Read	0x68	14	0x01F800583CC00022000FFFE306B0
7		4.92E-02	Write	0x68	2	0x7F00
8		4.94E-02	Write	0x68	1	0x2D
9		4.96E-02	Read	0x68	14	0x022000783D6000130018FFDC0730
10		5.25E-02	Write	0x68	2	0x7F00
11		5.27E-02	Write	0x68	1	0x2D
12		5.29E-02	Read	0x68	14	0x01B000D83D40001CFFF9FFFF06F0
13		5.58E-02	Write	0x68	2	0x7F00
14		5.60E-02	Write	0x68	1	0x2D
15		5.62E-02	Read	0x68	14	0x01B000803D40000E0019FFDE06B0
16		5.91E-02	Write	0x68	2	0x7F00
17		5.93E-02	Write	0x68	1	0x2D
18		5.95E-02		0x01	0	

À l'envers + roue bloquée

A	B	C	D	E	F	G	H
Bus Table: I2C (I2C: Clock SCL = C2, Data SDA = C1)	Frame	Mark	Start Time in s	Type	ID	Length	Data
1		1	3.97E-02	Write	0x68	2	0x7F00
4		2	3.99E-02	Write	0x68	1	0x2D
5		3	4.00E-02	Read	0x68	14	0x033005A0B Restart
6		4	4.30E-02	Write	0x68	2	0x7F00
7		5	4.32E-02	Write	0x68	1	0x2D
8		6	4.33E-02	Read	0x68	14	0x03180518B Data Error+Stop
9		7	4.63E-02	Write	0x68	2	0x7F00
10		8	4.65E-02	Write	0x68	1	0x2D
11		9	4.66E-02	Read	0x68	14	0x02880570B Data Error+Stop
12		10	4.96E-02	Write	0x68	2	0x7F00
13		11	4.98E-02	Write	0x68	1	0x2D
14		12	4.99E-02	Read	0x68	14	0x02580588B Data Error+Stop
15		13	5.29E-02	Write	0x68	2	0x7F00
16		14	5.31E-02	Write	0x68	1	0x2D
17		15	5.32E-02	Read	0x68	14	0x02100540B Restart

Rotation -90° selon X →

A	B	C	D	E	F	G
Bus Table: I2C (I2C: Clock SCL = C2, Data SDA = C1)	Frame	Mark	Start Time in s	Type	ID	Data
3		1	3.90E-02	Write	0x68	2 0x7F00
4		2	3.92E-02	Write	0x68	1 0x2D
5		3	3.94E-02	Read	0x68	14 0x41180510078000360031FF8F0680
6		4	4.23E-02	Write	0x68	2 0x7F00
7		5	4.25E-02	Write	0x68	1 0x2D
8		6	4.27E-02	Read	0x68	14 0x417004D807580046001FFF880630
9		7	4.56E-02	Write	0x68	2 0x7F00
10		8	4.58E-02	Write	0x68	1 0x2D
11		9	4.60E-02	Read	0x68	14 0x41D804E8073800540024FF8D0650
12		10	4.89E-02	Write	0x68	2 0x7F00
13		11	4.91E-02	Write	0x68	1 0x2D
14		12	4.93E-02	Read	0x68	14 0x41D004000770006F0024FF9D05E0
15		13	5.22E-02	Write	0x68	2 0x7F00
16		14	5.24E-02	Write	0x68	1 0x2D
17		15	5.26E-02	Read	0x68	14 0x421003F806E800750034FF9E0630

Rotation 90° selon X →

A	B	C	D	E	F	G	H	I
Bus Table: I2C (I2C: Clock SCL = C2, Data SDA = C1)	Frame	Mark	Start Time in s	Type	ID	Length	Data	State
3		1	3.92E-02	Write	0x68	2	0x7F00	OK
4		2	3.94E-02	Write	0x68	1	0x2D	OK
5		3	3.96E-02	Read	0x68	14	0xC0F8017001C8005E0062001D0630	Data Error+Stop
6		4	4.25E-02	Write	0x68	2	0x7F00	OK
7		5	4.27E-02	Write	0x68	1	0x2D	OK
8		6	4.29E-02	Read	0x68	14	0xC0400118027000700060001C0640	Data Error+Stop
9		7	4.58E-02	Write	0x68	2	0x7F00	OK
10		8	4.60E-02	Write	0x68	1	0x2D	OK
11		9	4.62E-02	Read	0x68	14	0xC07000F002900084006300160680	Data Error+Stop
12		10	4.91E-02	Write	0x68	2	0x7F00	OK
13		11	4.93E-02	Write	0x68	1	0x2D	OK
14		12	4.95E-02	Read	0x68	14	0xBFE80040380008F006300250640	Address Error+Data Error+Stop
15		13	5.24E-02	Write	0x68	2	0x7F00	Data Error+Stop
16		14	5.26E-02	Write	0x68	1	0x2D	OK
17		15	5.28E-02	Read	0x68	14	0xBFE8FF9803700081004F000906C0	Data Error+Stop



Figure 3.4: séquence d'initialisation

Séquence d'initialisation

La séquence d'initialisation ne semble comporter que des requêtes d'écriture. Celles-ci ne devraient donc pas poser de problème, on peut les ignorer. Le MPU n'a qu'à transmettre un acknowledge à chaque demande.

Au début les demandes de rafraîchissement du microcontrôleur pour le gyroscope s'enchaînent puis sont cadencés à partir de 320 ms.

3.5.2 Première communication

Nous allons utiliser un ESP32 pour communiquer avec le microcontrôleur à la place du gyroscope. Ainsi, nous pourrons simuler un changement d'angle de l'hoverboard et ainsi contrôler les moteurs.

Carte ESP 32

[I2C sur ESP32](#)

[utiliser la carte en tant qu'esclave i2c](#)

L'installation de diodes de pull-up côté esp32 permet à celui-ci de mieux communiquer.

Premier problème : 1 bit ne passe pas à tous les coups. Or ce bit est essentiel. Je vais donc essayer de raccourcir les fils pour limiter leur effet capacatif.

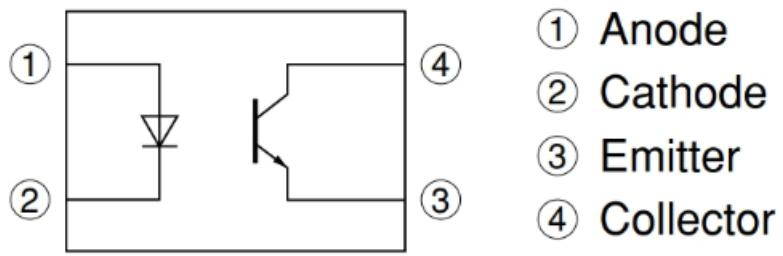
Solution : L'utilisation de setTimeOut(200) dans le setup permet de régler le problème du bit.

3.5.3 Les optocoupleurs

Les moteurs ne démarrent pas tant que les optocoupleurs ne sont pas obstrués. Il faut donc être en mesure de simuler leurs obstruction via l'esp32.

La sortie des optocoupleurs génère une tension de 2.81 V quand obstrué, 0V sinon.

J'utilise un transistor NMOS [BS270](#) pour le remplacer et contrôler la mise en marche via l'esp32.



easybom™

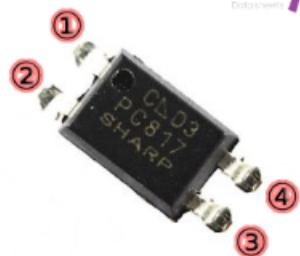


Figure 3.5: Schéma optocoupleur

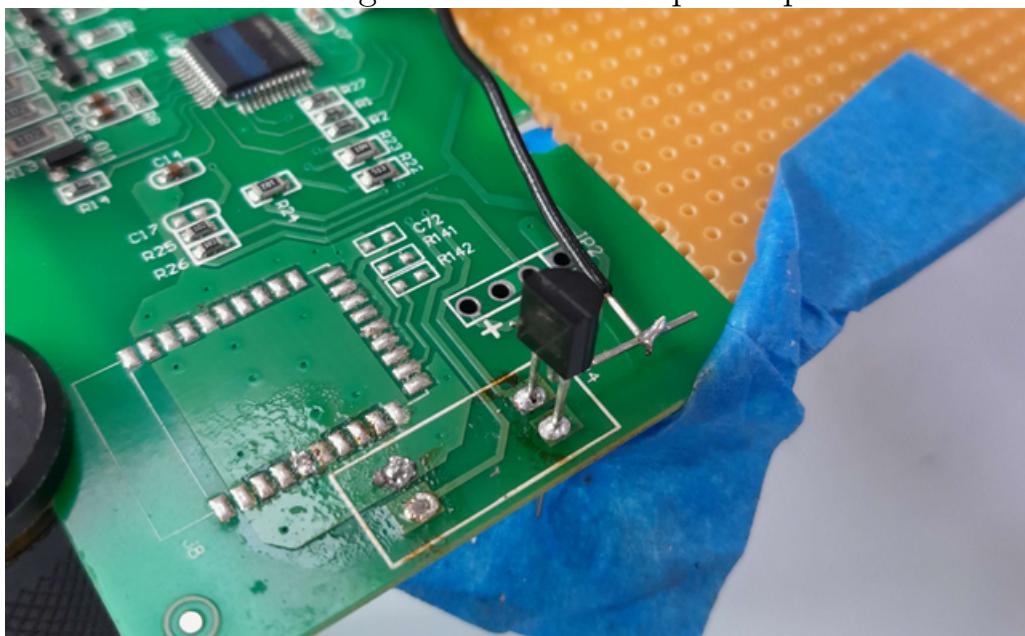
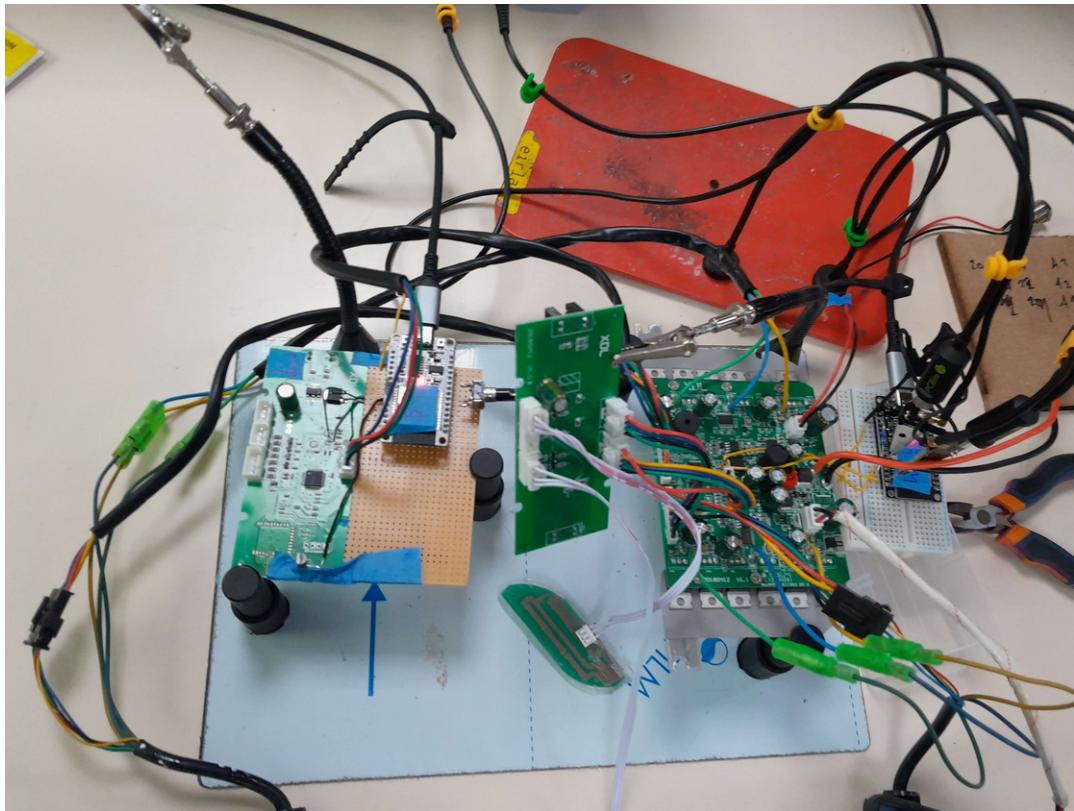


Figure 3.6: Transistor qui remplace l'optocoupleur



3.5.4 Gyroscope de la carte mère

Maintenant que l'esp32 peut remplacer le gyroscope R, il faut faire de même pour le gyroscope L. La difficulté est ici dans le placement du gyroscope au centre de la carte mère.

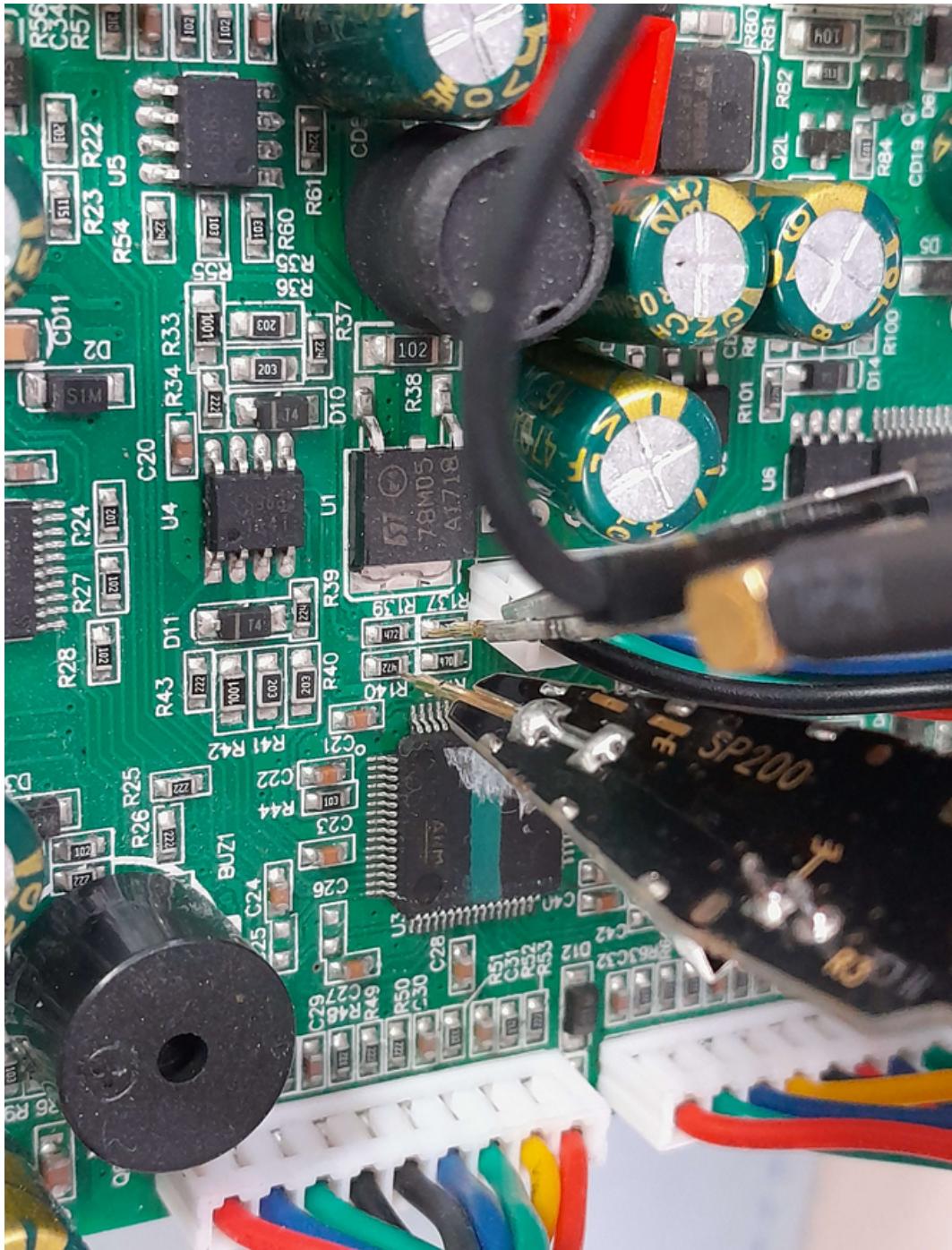


Figure 3.7: Branchement pour la communication SCL et SDA à la place du gyroscope L

Ici le test est fait avec un deuxième esp32, je vais maintenant essayer de faire passer toute la com sur le premier esp32.

En soudant mes fils, j'ai brûlé R1 39, et R1 38.

Les résistances ont été remplacées donc pas de séquelles.

3.5.5 Fonctionnement des deux roues

Je peux maintenant contrôler les deux moteurs via un module radio que j'ai ajouté pour faciliter le débogage.

3.6 Structure

Pour éviter les court circuits, je vais créer une structure pour le montage. J'utilise des vis et écrous en 2.5x12.

3.7 Asservissement

On aimerait contrôler en vitesse les deux moteurs, or ceux-ci suivent une loi inconnue. Il semble que lorsque les deux moteurs tournent pour avancer ou reculer, leurs vitesses soient commandées par la primitive de la commande. Si les deux moteurs tournent en sens opposé, ceux-ci sont commandés sans primitive.

Pour déterminer la commande, je me branche sur les trois capteurs effet hall de chaque roue. ainsi, je peux étudier la vitesse de chaque roue. Les valeurs des capteurs effet hall sont récupérées par l'esp32, transféré en vitesse, filtrés et décimés puis transféré à l'ordinateur. Le but est de minimiser la fréquence de transmission de données pour permettre la transmission sans perdre trop d'information. Grâce à un script python, il est possible de contrôler la vitesse des moteurs et d'étudier leur vitesse.