INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
UNIVERSITÉ D'ANGERS

Projet SAE S4

Télémètre TOF

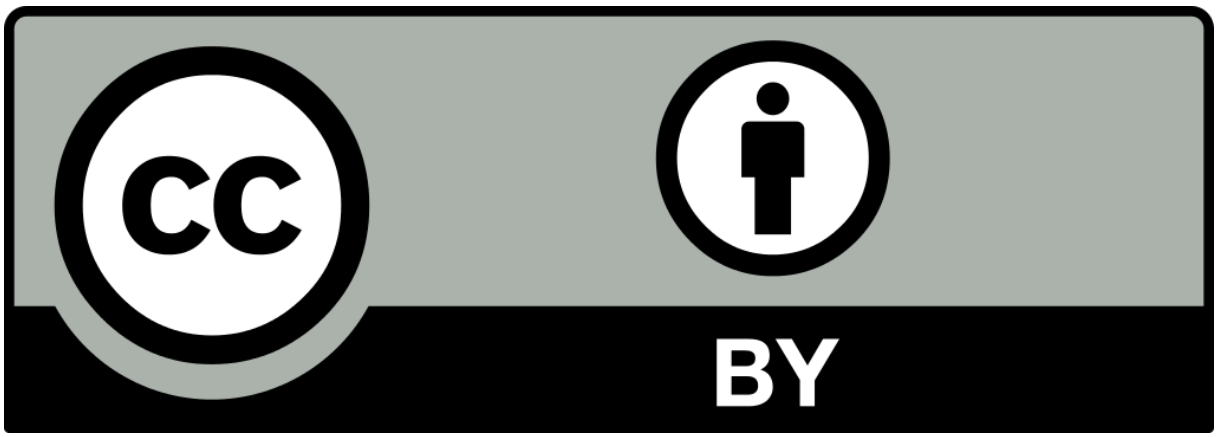
life.augmented

CLEMENT Ewan BURLOT Gweltaz

GEII – 2300

2022-2023

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs. Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.

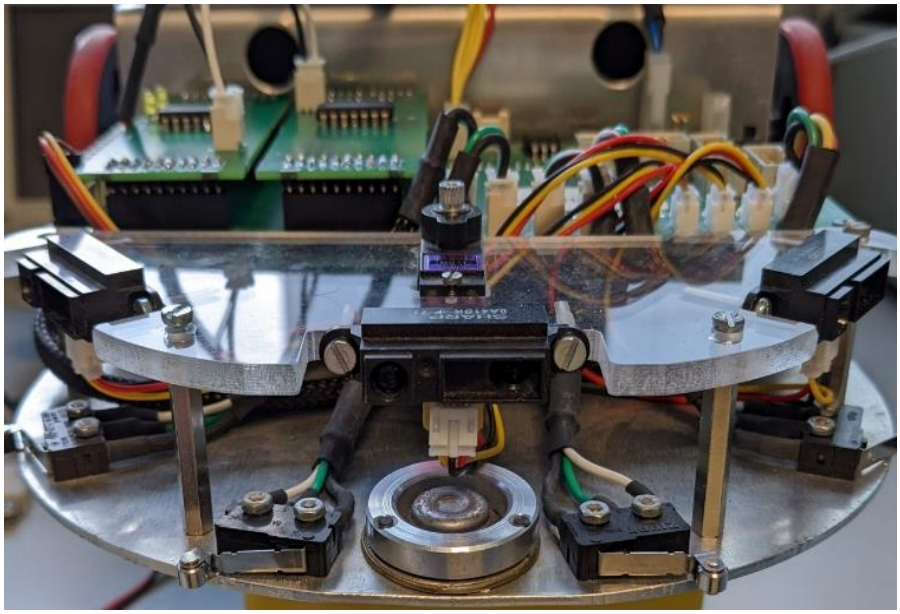


Sommaire

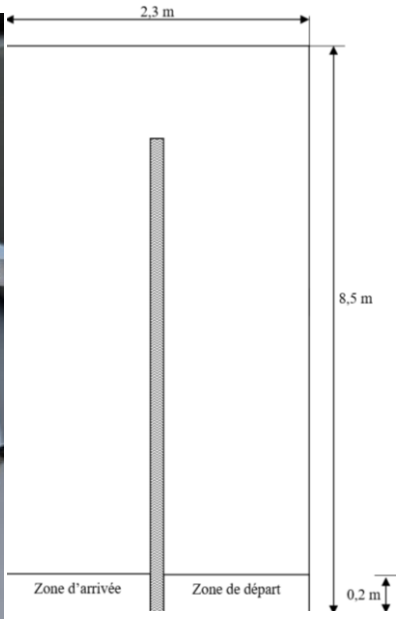
Avant de commencer, nous souhaitons exprimer nos remerciements. Tout d'abord, nous tenons à remercier chaleureusement l'IUT d'Angers pour avoir mis à notre disposition tout le matériel nécessaire à la réalisation de notre projet. Nous souhaitons également exprimer notre gratitude envers le site Adafruit et Wayne Holder pour avoir partagé des informations qui nous ont été d'une grande utilité.
Enfin, nous aimerions adresser nos remerciements les plus sincères à M. Lucidarme pour sa direction du projet et son assistance précieuse en répondant à toutes nos interrogations.

Introduction :

Depuis plusieurs années, les premières années de GEII à l'IUT d'Angers découvrent l'électronique en réalisant un projet autour d'un robot. Le but du projet *SAE robot* de parcourir le circuit ci-dessous le plus rapidement possible. Pour réaliser ce cahier des charges, les étudiants possèdent chacun un robot, doté de plusieurs capteurs qui lui permettent d'éviter les obstacles. Celui-ci est équipé de 3 capteurs SHARP et de 4 interrupteurs servant de capteur de contact lorsque le robot heurte un obstacle.

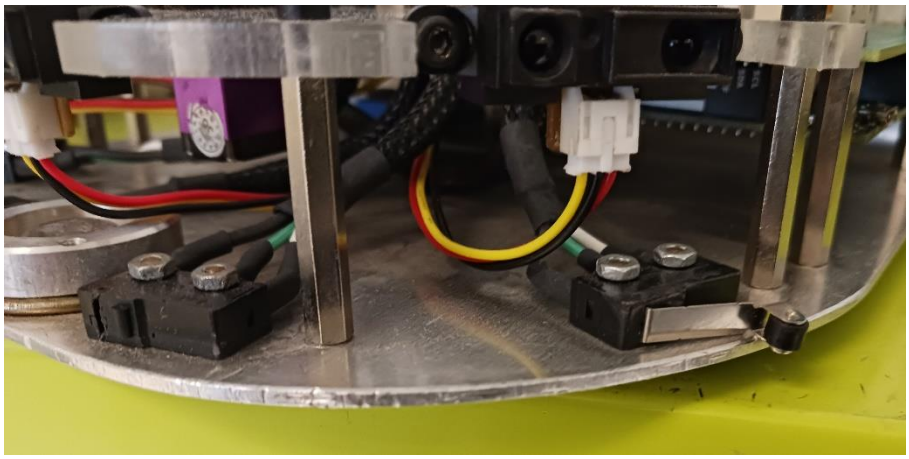


La face avant du robot équipé de ces 7 capteurs



Le circuit à parcourir par le robot

Le problème rencontré est qu'avec les nombreux chocs que peut subir le robot durant la course, les interrupteurs se cassent régulièrement. Il est assez compliqué pour les techniciens de l'IUT de remplacer ce composant une fois en panne.



A gauche, un des interrupteurs du robot endommagé

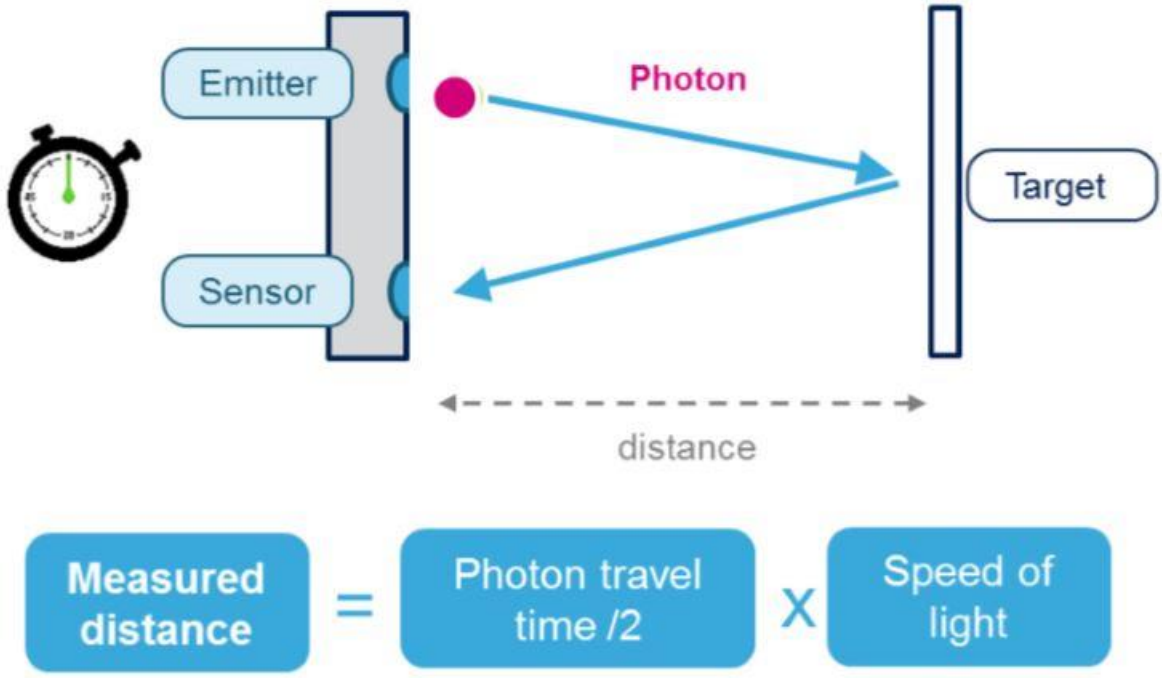
Notre objectif était donc de trouver un système alternatif aux interrupteurs afin de fiabiliser le capteur de contact.

Cahier des charges :

Comme dit plus haut, notre capteur doit pouvoir être capable de résister aux chocs encaissés par le robot. De plus pour des raisons pédagogiques, pour simuler l'effet de l'interrupteur, le capteur doit pouvoir renvoyer un signal logique lorsque le robot est en contact avec un obstacle. Le capteur utilisé doit aussi être capable de faire une mesure précise peu importe la matière de l'obstacle situé en face de lui (Surface brillante/mat/réfléchissante etc…). Pour finir, le capteur doit être assez petit pour pouvoir être installé sur un robot du projet SAE de première année.

Solutions techniques :

La solution qui nous a été directement proposée a été d'utilisé un capteur dit « Time of flight ». Le principe de ce capteur est d'utiliser une onde (mécanique ou électromagnétique) comme une onde sonore ou une source lumineuse pour mesurer une distance. Le capteur mesure le temps entre l'émission et la réception de l'onde émise. Etant donné que la vitesse de cette onde est constante (et est connue) on peut déterminer sa distance grâce à la formule : Vitesse = Distance/Temps.



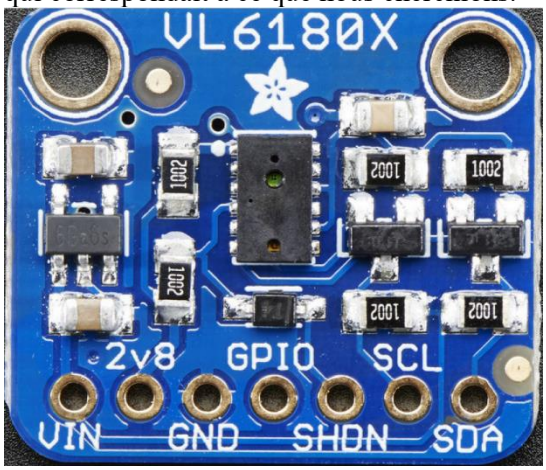
Source : stmicroelectronics-vl53l1x-tof-proximity-sensor

Le capteur qui nous été proposé a été le capteur VL6180X de chez STM Micoreltronics, il permet de mesurer des distances jusqu'à environ 20cm a une fréquence d'environ 100Hz. L'avantage que possède ce capteur par rapport aux capteurs SHARP est que la mesure de distance est indépendante de l'obstacle sur laquelle le laser est réfléchi. De plus il existe une version Shield de ce capteur. Cela nous permet de commencer le développement du programme pendant la conception de la carte capteur.

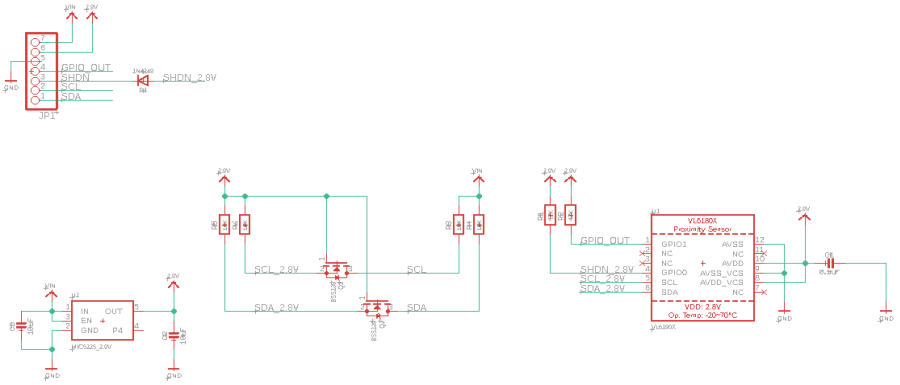
Conception – Hardware :

Schéma de la carte capteur :

Afin de correspondre aux caractéristiques du robot des premières années, l'objectif était de faire une carte avec le capteur VL6180X assez petite pour se placer dans les quelques centimètres qui lui seront dédiés. Le véritable enjeu de cette carte était de gérer les conversions de tensions. En effet, le capteur s'alimente en 2.8V alors que les alimentations extérieures ainsi que les sorties doivent être en 3.3V ou en 5V. Il fallait donc allier une carte avec le moins de composants possibles afin de limiter sa taille tout en faisant les conversions de tensions voulus.
Les idées de conceptions ont donc évolué au cours du temps. Au début, l'objectif était d'avoir une unique carte avec le capteur VL6180X, un microcontrôleur et un minimum de composants pour faire fonctionner le tout.
Nous avons fait un brouillon de schéma sur lequel se baser afin de choisir les composants à utiliser.
Mais en analysant la mise en place sur le robot nous nous sommes rendu compte que ce ne serait pas le plus simple à mettre en place. L'une des raisons étaient que nous voulions piloter une led à partir de GPIO0 mais nous nous sommes rendu compte que ce n'était pas possible.
Nous avons donc changé d'avis pour partir sur deux cartes qui s'assembleraient. Dans un premier temps il fallait faire la première carte avec le capteur VL6180X et le strict minimum pour la faire fonctionner (comme expliqué plus tôt). Puis dans un second temps, faire une deuxième carte avec un microcontrôleur pour communiquer entre le robot et la carte capteur.
En effectuant des recherches nous avons trouvé une carte faite par Adafruit pour le capteur VL6180X qui correspondait à ce que nous cherchions.

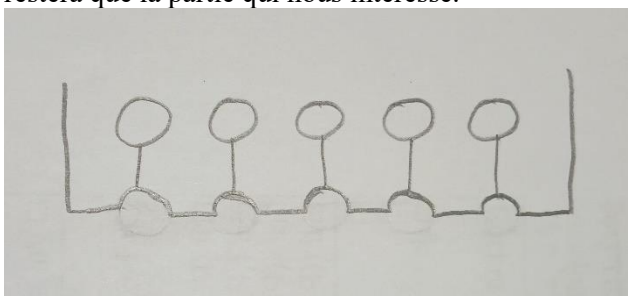


Pour gagner du temps et être sûr des composants, nous avons récupéré le schéma Eagle d'Adafruit pour travailler directement dessus (voir-schéma ci-dessous).

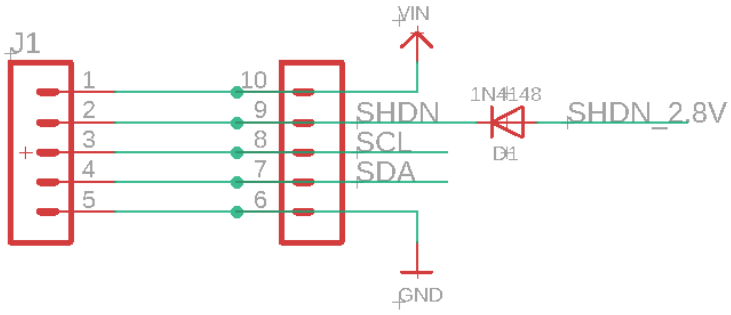


Le schéma se divise en plusieurs parties. A droite, nous avons le capteur VL6180X avec le câblage indiqué dans la documentation du capteur : vl6180x.pdf
Au milieu, il y a la partie pour l'adaptation de tension avec des transistors. L'objectif ici est de transformer les BUS SCL et SDA qui arrivent avec une tension de 2.8 V en une tension V_{IN} qui dans notre cas, sera du 3.3V ou du 5V.
La partie de gauche a pour rôle, d'adapter la tension grâce à un régulateur. Il transforme une tension V_{IN} en 2.8 V.
En haut à gauche, il y a une broche d'E/S avec une diode que nous avons a testé sur *LTSpice* pour vérifier son bon fonctionnement. Nous voulions être sûr que le 5V à gauche ne perturbait pas le 2.8V à droite (test détaillé plus loin).
Une fois le schéma récupéré, il y avait quand même quelques petites modifications à faire pour correspondre avec nos besoins.
Il fallait changer la broche de connexion (tout en haut à gauche) afin de pouvoir souder cette carte à 90° sur la seconde. L'objectif de base était d'avoir une broche de ce type :

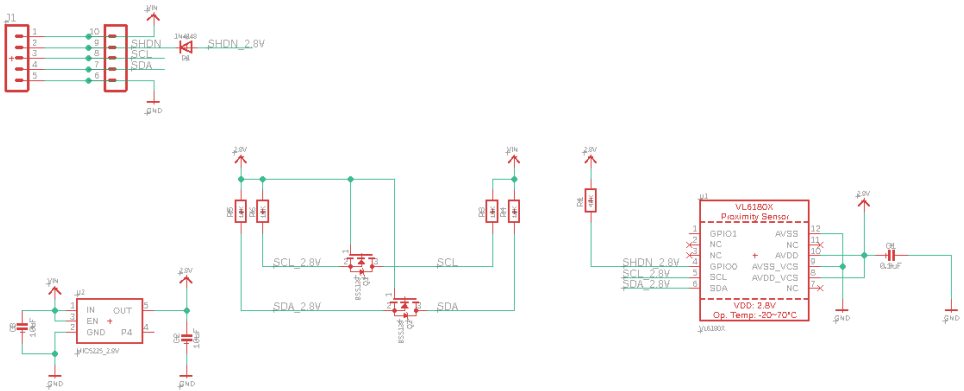
Mais on nous a dit que faire l'empreinte d'une broche de ce type n'était pas possible. Nous sommes donc parties sur une autre option similaire (voir schéma ci-dessous). Le but est de faire deux trous reliés entre eux avec l'un à moitié en dehors de la carte. Le but est que quand la découpe de la carte sera faite, il restera que la partie qui nous intéresse.



L'empreinte de cette broche étant plus simple que l'autre, nous avons pu la récupérer sur Eagle :



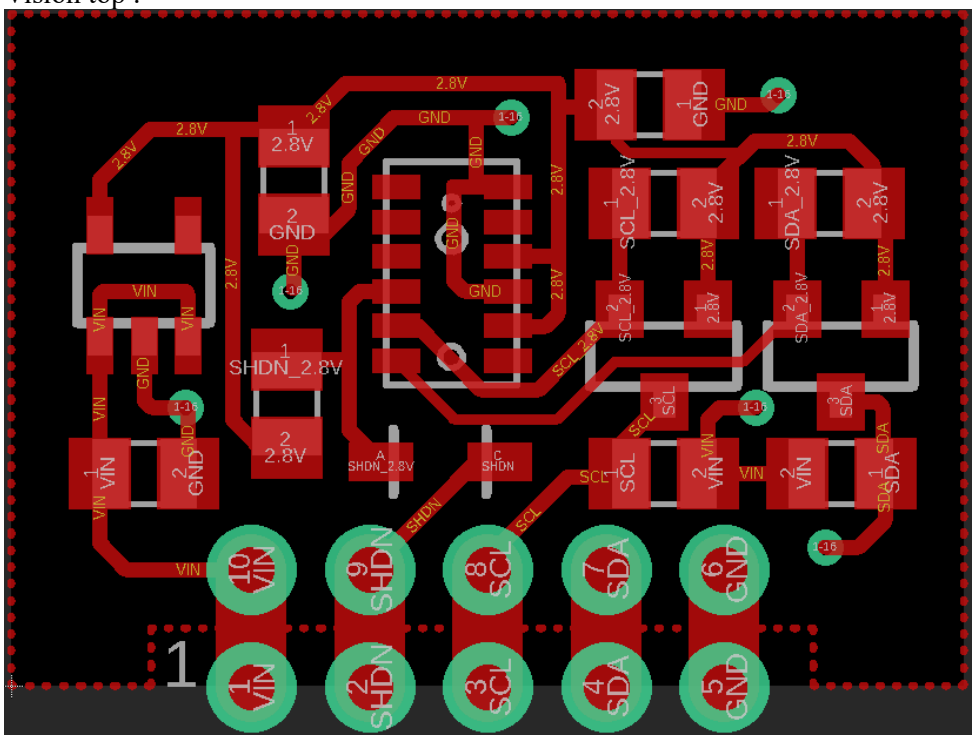
Les 3 voies sont entre la masse et l'alimentation car cela permettra de les isoler un peu plus des perturbations électromagnétiques. Les voies GPIO_OUT et 2.8V ont été supprimé car nous en n'avions pas besoin.
De même pour GPIO1 du capteur VL6180X qui a été enlevé afin d'avoir notre schéma final :



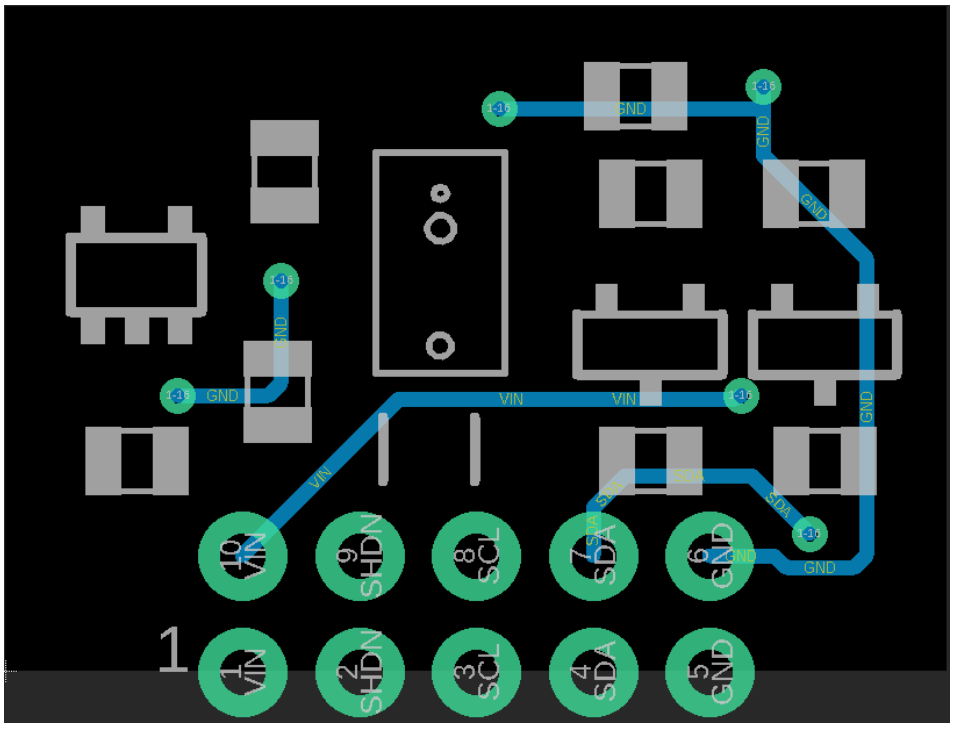
Routage :

Pour le routage, nous nous sommes basés sur l'image de la carte d'Adafruit pour le placement des composants. Ça nous a permis de gagner du temps et d'optimiser leurs placements. Nous avons ensuite fait le routage, que nous avons recommencé plusieurs fois car il y avait beaucoup de problèmes sur le premier. Nous avons notamment dû replacer les pistes pour optimiser l'espace et avoir le moins de vias possibles. Les pistes ont dû être agrandies car elles étaient trop petites. Et nous avons rajouté un plan de masse en plus d'avoir connecté les GND entre eux afin d'être sûr qu'ils soient bien tous reliés.

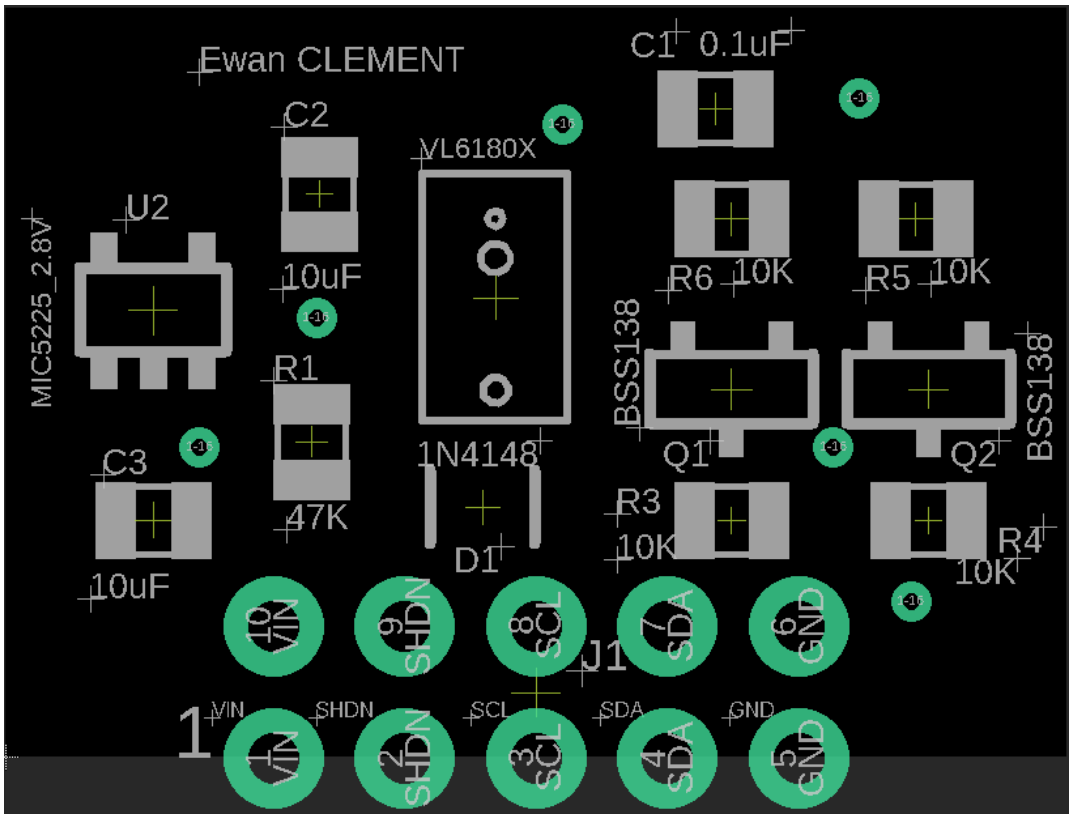
Vision top :



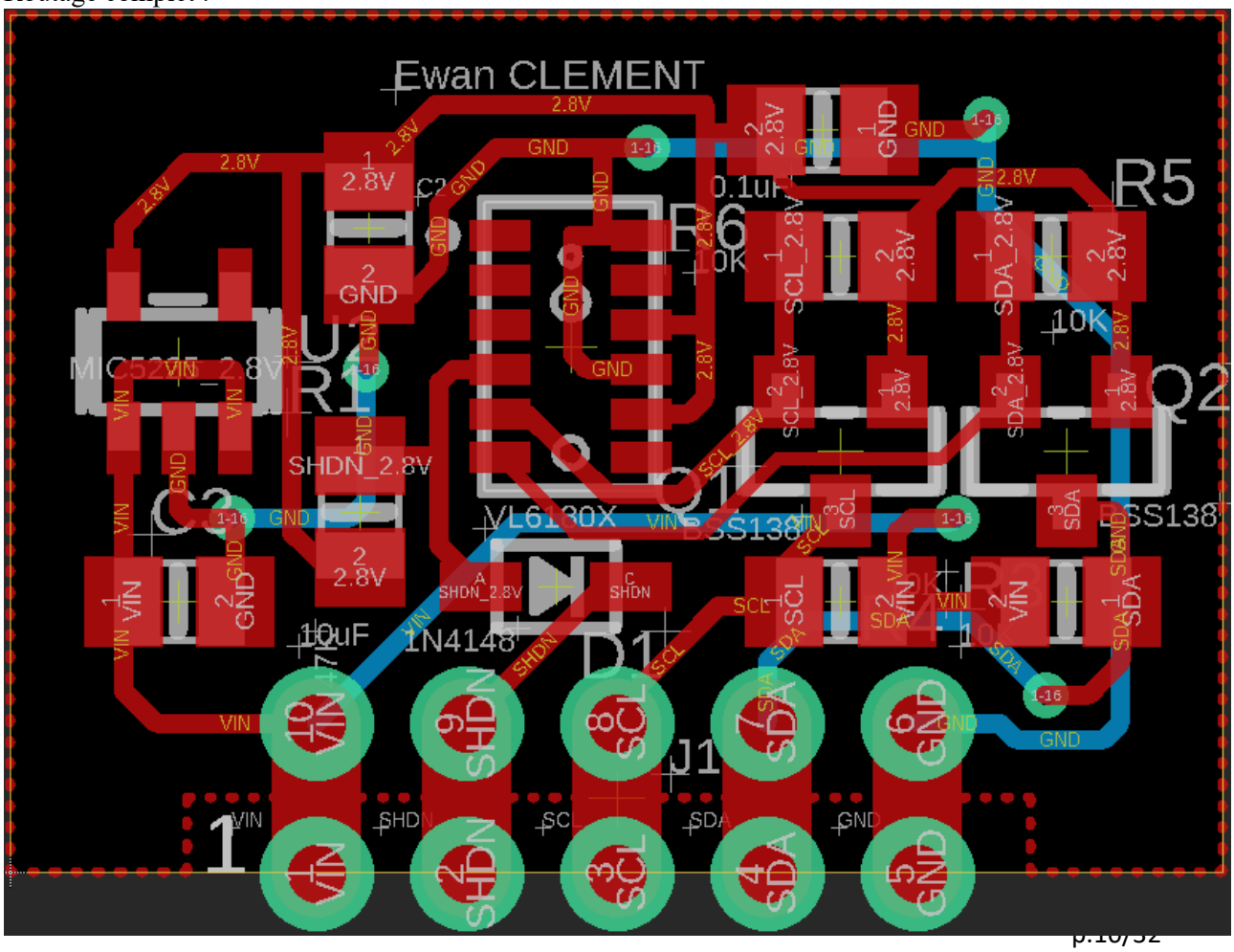
Vision bottom :



Vision nom :

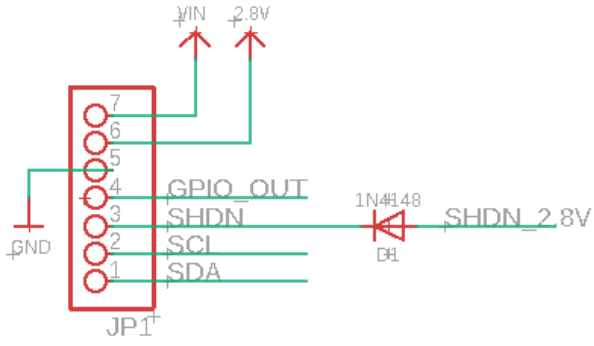


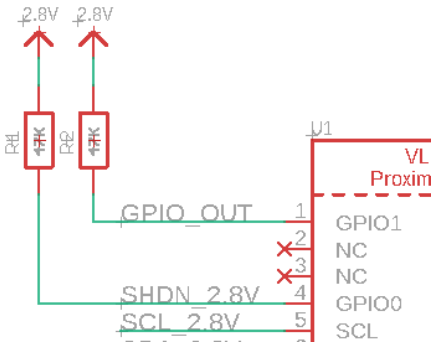
Routage complet :



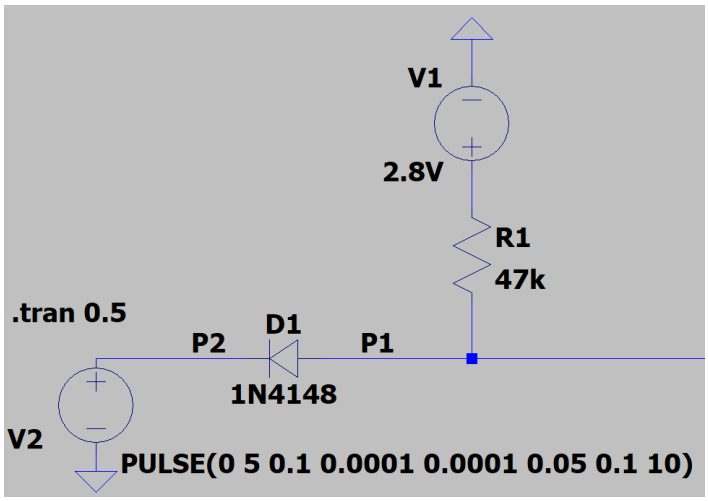
Simulation diode :

Après avoir trouvé le schéma de la carte capteur faite par Adafruit, nous trouvions que le montage avec la diode pour adapter la tension était peu conventionnel. En effet, la diode sépare la partie qui est en 2.8V et celle qui est en 5V. Nous devions donc vérifier si ce montage était bien fonctionnel.

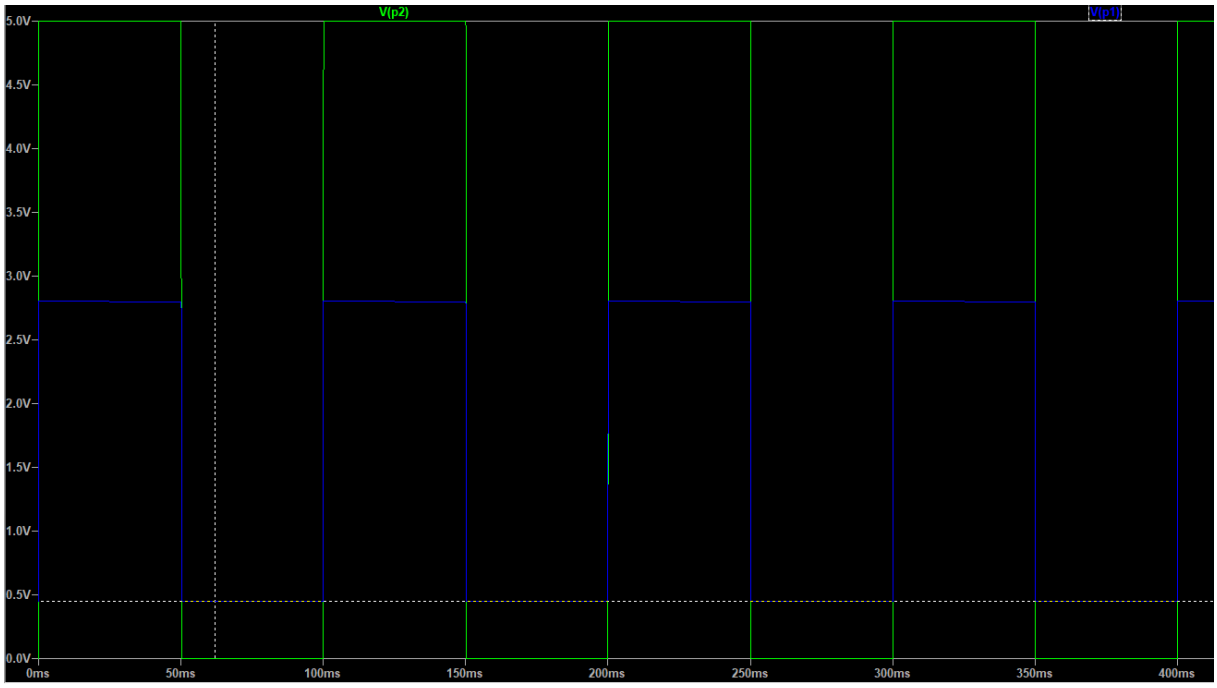




Sur le schéma ci-dessus, il fallait vérifier que la partie SHDN_2.8 reste bien à 2.8 V quand l'autre côté de la diode est alimenté par du 5 V et que les deux tombent à 0V en même temps. Voici les résultats de la simulation de ce montage sur *LTspice* :



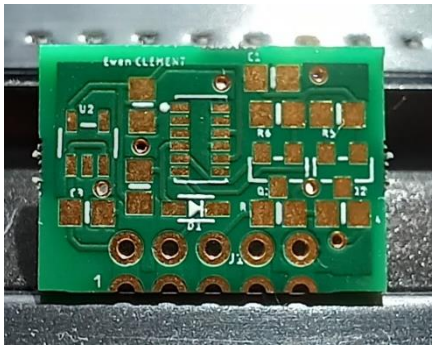
La courbe bleue est le point P1 (SHDN_2.8V)
La courbe verte est le point P2 (SHDN)



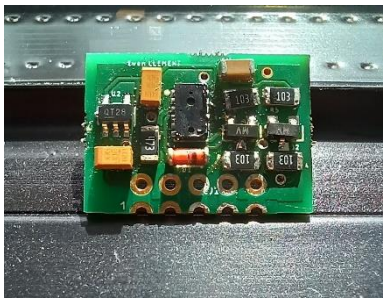
Cette simulation est cohérente. Quand la partie cathode de la diode est à 5V et sa partie anode est à 2.8V alors son état est bloqué. Les deux tensions sont donc indépendantes. Lorsque la partie cathode passe à 0V alors la diode devient passante et la tension côté anode chute. Ici la tension chute à 448 mV.
Cette simulation confirme que le schéma sur la carte vendu par Adafruit est bien fonctionnel.

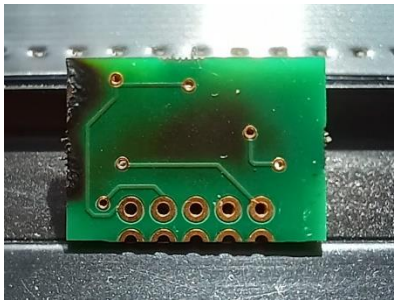
Soudure de la carte :

Une fois le circuit imprimé de la carte et les composants reçu, nous avons pu commencer à souder la carte. Il a fallu étudier la manière de souder le composant VL6180X car c'était la première fois qu'on avait à faire à des empreintes aussi petites.

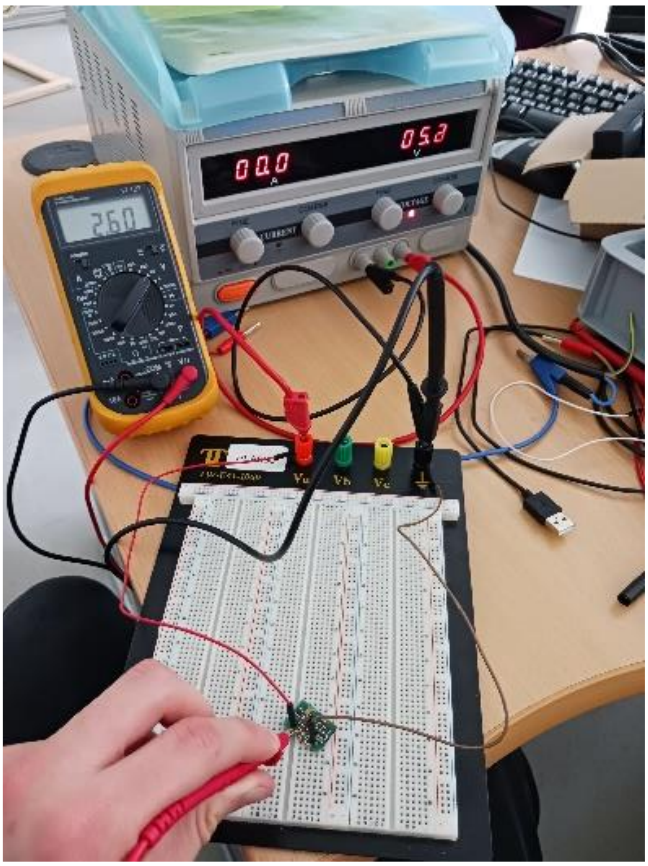


La solution qui paraissait la plus adaptée, était la station à air chaud. Etant donné que c'était la première fois que nous avions à utiliser cet appareil, il y eu quelques problèmes pendant son utilisation.





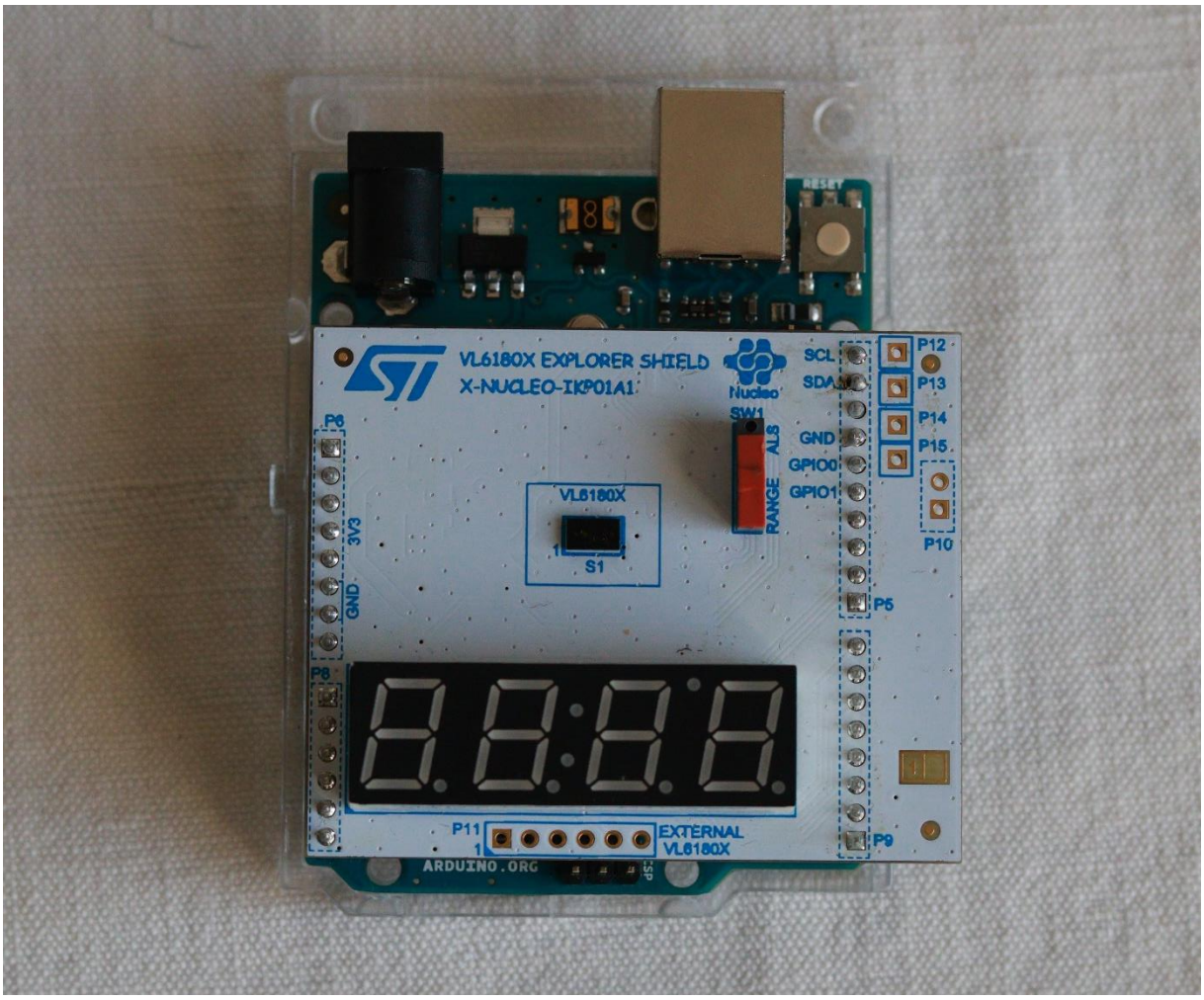
Comme on peut le voir ci-dessus, la carte a un peu brulé par derrière. Ces brûlures sont uniquement visuelles car elles n'ont pas atteint les circuits. Nous pouvons aussi voir que le plastique à droite du capteur au centre est légèrement brulé. Une fois toute la carte soudée, nous l'avons testé avec une alimentation, un multimètre et un oscilloscope et lorsqu'on approchait la main du capteur on voyait les signaux bougés sur l'oscilloscope.



Finalement, la carte avait l'air bien fonctionnel malgré les difficultés rencontrés pendant la soudure. Il ne reste plus qu'à tester la carte avec le programme sur Arduino.

Conception – Software :

Pour réaliser le programme associé au capteur TOF nous nous sommes basés sur le programme réalisé par Wayne Holder pour comprendre le fonctionnement du capteur ainsi que les registres nécessaires à son initialisation. De plus nous utiliserons les documentations fournies par le constructeur. N'ayant pas de carte capteur fonctionnelle au début du projet nous avons utilisé la carte Shield Arduino du VL6180X.



La Shield VL6180X Branchée sur une Arduino UNO R3

Cette Shield est composée du capteur VL6180X, de 4 afficheurs 7-segments et d'un switch permettant de passer en mesure de distance (Range) ou en mesure de luminosité ambiante (ALS – *Ambient Light Sensor*).

Protocole d'initialisation du capteur VL6180X :

Le capteur ToF VL8160X est un capteur optique utilisé afin de mesurer des courtes distance (Environ 15 cm dans la version que nous utilisons). Afin de pouvoir l'utiliser, il est nécessaire de réaliser une première phase d'initialisation.
Le capteur est composé de deux mode de mesures*:

- *ALS (Ambiant Light Sensor) :* Ce mode de mesure permet mesurer la luminosité d'une pièce, la mesure est effectuée en LUX. Dans le cadre du projet nous n'utiliserons pas ce mode
- *Range measurement* : La mesure de distance comme son nom l'indique, permet de mesurer la distance séparant le capteur d'un obstacle présent face à lui.

**Note : Il est possible d'utiliser les deux modes de mesure en simultané (Intervaled mode). Dans le projet, nous utiliserons que le Range measurements cette option ne sera donc pas détaillée*

Durant ces mesures, deux modes d'acquisition sont disponibles :

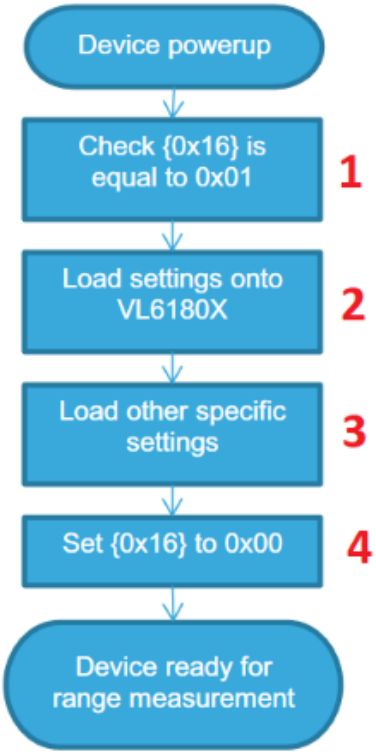
- Mode *Single-Shot* : permet de faire une seule acquisition à l'aide du capteur
- Mode *continuous*: Permet de faire des mesures en continu

Afin d'analyser les différents registres utiles à l'initialisation, nous nous baserons sur un programme de démo fonctionnel pour Arduino ainsi qu'avec la documentation technique fournie par le constructeur.

Initialisation du capteur - Chargement des paramètres du capteur :

Afin d'initialiser le capteur, le constructeur nous fournit cette procédure à suivre pour démarrer celui-ci :

Figure 3. Initialisation steps



1- Vérification du Registre 0x016 :

La phase d'initialisation commence par la vérification du Bit de poids faible dans le registre *0x016 « Fresh Out of Reset Register ».* Sur les 8 bits du registre, il y a uniquement le bit de poids faible qui est modifiable. Lorsque celui-ci est à 1, celui-ci peut être utilisé comme reset (p.*58* - vl6180x.pdf)

6.2.14 SYSTEM__FRESH_OUT_OF_RESET

7	6	5	4	3	2	1	0
RESERVED							fresh_out_of_reset
R							R/W

Address: 0x016

Type: R/W

Reset: 0x1

Description:

[0]	fresh_out_of_reset: Fresh out of reset bit, default of 1, user can set this to 0 after initial boot and can therefore use this to check for a reset condition

2- Chargements des paramètres SR03

Une série de paramètres sont à charger sur le capteur, nous n'avons pas trouvé d'information détaillées sur les registres mentionnés. Le constructeur recommande d'utiliser les paramètres que celui-ci donne dans la documentation (*p.24 vl6180x-basic-ranging-application-note-stmicroelectronics.pdf*).

9 SR03 settings

Below are the recommended settings required to be loaded onto the VL6180X during the initialisation of the device (see *Section 1.3*).

```
// Mandatory : private registers
WriteByte(0x0207, 0x01);
WriteByte(0x0208, 0x01);
WriteByte(0x0096, 0x00);
WriteByte(0x0097, 0xfd);
WriteByte(0x00e3, 0x01);
WriteByte(0x00e4, 0x03);
WriteByte(0x00e5, 0x02);
WriteByte(0x00e6, 0x01);
WriteByte(0x00e7, 0x03);
WriteByte(0x00f5, 0x02);
WriteByte(0x00d9, 0x05);
WriteByte(0x00db, 0xce);
WriteByte(0x00dc, 0x03);
WriteByte(0x00dd, 0xf8);
WriteByte(0x009f, 0x00);
WriteByte(0x00a3, 0x3c);
WriteByte(0x00b7, 0x00);
WriteByte(0x00bb, 0x3c);
WriteByte(0x00b2, 0x09);
WriteByte(0x00ca, 0x09);
WriteByte(0x0198, 0x01);
WriteByte(0x01b0, 0x17);
WriteByte(0x01ad, 0x00);
WriteByte(0x00ff, 0x05);
WriteByte(0x0100, 0x05);
WriteByte(0x0199, 0x05);
WriteByte(0x01a6, 0x1b);
WriteByte(0x01ac, 0x3e);
WriteByte(0x01a7, 0x1f);
WriteByte(0x0030, 0x00);
```

3- Chargements des paramètres spécifiques

En addition des paramètres fourni par le constructeur, d'autres registres sont à paramétrer par l'utilisateur en fonction de son utilisation.

```
// Recommended register setup
//p.55 -> Set the GPIO1 as a Interrupt Output
writeByte(0x0011, 0x10);

//p.54 -> Set the GPIO0 as Main Xshutdown input, when enabled other
bits of the register are ignored **(+Signal Polarity Selection
???)**
writeByte(0x0010, 0x10);

//p.75 -> Set Averaging sample period (=Reset Value)
writeByte(0x010a, 0x30);

//p.67 -> OK(Register fixed) SYSALS Analog Gain
//writeByte(0x003f, 0x46);

//p.64 -> Meaning at p.30 Calibration due to Temperature sensibility
from the sensor User entered repeat rate of auto VHV task ??? Repeat
Rate is enabled 0xFF = 255
writeByte(0x0031, 0xFF);

//p.67 -> Recommanded Settings for SYS ALS Integration period
(100ms)
//writeByte(0x0040, 0x63);

//p.64 -> Manual trigger for VHV recalibration enabled -- Perform a
single temperature calibration of the ranging sensor
writeByte(0x002e, 0x01);

// Optional registers
//P.63 -> Disable Early Convergence Estimate ***MODIFIED
writeByte(0x002d, 0x10);

//p.60 -> SYSRange Intermeasurments period **IN CONTINIOUS MODE** --
->Range 0-254 (0 = 10ms). Step size = 10ms
writeByte(0x001b, 0x10);

//p.66 -> SYS ALS Intermeasurments period Same bit range as SYS
Range Inter. period
writeByte(0x003e, 0x31);

//p.57 -> Interrup mode source For ALS Reading = 4 -> New sample
ready   **same for range readings
writeByte(0x0014, 0x04);

// Change fresh out of set status to 0
// Set the register from beginning to 0 to end the initialisation
process (c.f. diagram). Needs to be set at 0 once the boot is done
writeByte(0x0016, 0x00);
```

Dans notre cas, nous nous intéresserons seulement des registres paramétrant la mesure de distance

Registre 0x0011 configuration de GPIO1(p.55) :

La sortie *GPIO1* est configuré comme sortie d'interruption dans le code. Celui-ci est actif lorsque *GPIO1* est à un niveau bas. Dans notre utilisation du capteur GPIO1 ne sera pas utilisé.

6.2.10 SYSTEM__MODE_GPIO1

7	6	5	4	3	2	1	0
RESERVED		system__gpio1_polarity	system__gpio1_select				RESERVED
R		R/W	R/W				R/W

Address: 0x011

Type: R/W

Reset: 0x20

Description:

[5]	system__gpio1_polarity: Signal Polarity Selection. 0: Active-low 1: Active-high
[4:1]	system__gpio1_select: Functional configuration options. 0000: OFF (Hi-Z) 1000: GPIO Interrupt output
[0]	Reserved. Write 0.

Registre 0x010A configuration de GPIO0(p.54) :

Dans notre cas, *GPIO0* est configuré en tant que *Main Shutdown Input,* les bits de poids plus faible sont donc ignorés. Il permet de mettre le capteur en mode vielle afin de limiter sa consomation en courant lorsqu'il n'est pas utilisé.
Les Bits de poids faible peuvent être utilisés afin de configurer *GPIOI0* comme sortie d'interruption.

6.2.9 SYSTEM__MODE_GPIO0

7	6	5	4	3	2	1	0
RESERVED	system__gpio0_is_xshutdown	system__gpio0_polarity	system__gpio0_select				RESERVED
R	R/W	R/W	R/W				R/W

Address: 0x010

Type: R/W

Reset: 0x60

Description:

[6]	system__gpio0_is_xshutdown: Priority mode - when enabled, other bits of the register are ignored. GPIO0 is main XSHUTDOWN input. 0: Disabled 1: Enabled - GPIO0 is main XSHUTDOWN input.
[5]	system__gpio0_polarity: Signal Polarity Selection. 0: Active-low 1: Active-high
[4:1]	system__gpio0_select: Functional configuration options. 0000: OFF (Hi-Z) 1000: GPIO Interrupt output
[0]	Reserved. Write 0.

Registre 0x0006 configuration du Gain Analogique du capteur ALS (p.67) :

Dans notre cas, il n'est pas utile de configurer ce registre nous n'utiliserons pas le capteur de luminosité (ALS).
Il permet de configurer la valeur de gain en convertissant les valeur allant de 0 à 7 en binaire dans les bits [2..0]
Le gain permet d'amplifier le signal reçu par le capteur

Registre 0x0031 Calibration VHV (Very High Voltage) (p.64) :

Ce registre est utilisé pour régler la fréquence de la calibration VHV du capteur. Le capteur étant sensible à la température, les mesures peuvent être influencées par ce changement et peut donc, donner une valeur erronée de la distance. Il est donc nécessaire de réaliser une calibration de capteur. Dans notre cas, le capteur réalisera une calibration après toutes les 255 mesures.

6.2.30 SYSRANGE__VHV_REPEAT_RATE

7	6	5	4	3	2	1	0
sysrange__vhv_repeate_rate							
R/W							

Address: 0x031

Type: R/W

Reset: 0x0

Description:

[7:0]	sysrange__vhv_repeat_rate: User entered repeat rate of auto VHV task (0 = off, 255 = after every 255 measurements)

Registre 0x0040 Période du capteur de luminosité (p.67)

Dans notre Utilisation, nous ne servons pas du capteur de luminosité (ALS)

6.2.36 SYSALS__INTEGRATION_PERIOD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED							sysals__integration_period								
R							R/W								

Address: 0x040

Type: R/W

Reset: 0x0

Description:

[8:0]	sysals__integration_period: Integration period for ALS mode. 1 code = 1 ms (0 = 1 ms). Recommended setting is 100 ms (0x63).

Registre 0x002E : Recalibration VHV (p.64)

Nous n'avons pas compris l'utilité de ce registre, nous supposons qu'il permet de sélectionner le moment où la calibration VHV s'effectue.

6.2.29 SYSRANGE__VHV_RECALIBRATE

7	6	5	4	3	2	1	0
RESERVED						sysrange__vhv_status	sysrange__vhv_recalibrate
R						R/W	R/W

Address: 0x02E

Type: R/W

Reset: 0x0

Description:

[1]	sysrange__vhv_status: FW controlled status bit showing when FW has completed auto-vhv process. 0: FW has finished autoVHV operation 1: During autoVHV operation
[0]	sysrange__vhv_recalibrate: User-Controlled enable bit to force FW to carry out recalibration of the VHV setting for sensor array. FW clears bit after operation carried out. 0: Disabled 1: Manual trigger for VHV recalibration. Can only be called when ALS and ranging are in STOP mode

Registre 0x001B & 0x003E SYS Range/SYS ALS Intermearsuments period (p.60/p.66)

Ce registre permet de configurer le temps entre deux mesures ALS ou Range réalisées. Il est configuré de 0 à 254 par pas de 10 ms. Celui-ci s'applique uniquement en mode continu. De plus, dans notre cas nous utiliserons uniquement *l'intermeasurments Range.*

6.2.19 SYSRANGE__INTERMEASUREMENT_PERIOD

7	6	5	4	3	2	1	0
sysrange__intermeasurement_period							
R/W							

Address: 0x01B

Type: R/W

Reset: 0xFF

Description:

[7:0]	sysrange__intermeasurement_period: Time delay between measurements in Ranging continuous mode. Range 0-254 (0 = 10ms). Step size = 10ms.

6.2.34 SYSALS__INTERMEASUREMENT_PERIOD

7	6	5	4	3	2	1	0
sysals__intermeasurement_period							
R/W							

Address: 0x03E

Type: R/W

Reset: 0xFF

Description:

[7:0]	sysals__intermeasurement_period: Time delay between measurements in ALS continuous mode. Range 0-254 (0 = 10ms). Step size = 10ms.

Registre 0x0024 Config Interruption GPIO ALS & Range

Ce registre permet de configurer la source d'interruption des mesures que l'on réalise. Il est configurable indépendamment pour le mode ALS & Range
Il est possible de configurer une interruption lorsque

- La valeur de seuil est supérieure à celle mesurée
- La valeur de seuil est inférieure à celle mesurée
- La valeur mesurée se situe entre le seuil bas et le seuil haut
- L'interruption se déclenche lorsque qu'une nouvelle mesure a été effectuée

6.2.12 SYSTEM__INTERRUPT_CONFIG_GPIO

7	6	5	4	3	2	1	0
RESERVED		als_int_mode			range_int_mode		
R		R/W			R/W		

Address: 0x014

Type: R/W

Reset: 0x0

Description:

[5:3]	als_int_mode: Interrupt mode source for ALS readings: 0: Disabled 1: Level Low (value < thresh_low) 2: Level High (value > thresh_high) 3: Out Of Window (value < thresh_low OR value > thresh_high) 4: New sample ready
[2:0]	range_int_mode: Interrupt mode source for Range readings: 0: Disabled 1: Level Low (value < thresh_low) 2: Level High (value > thresh_high) 3: Out Of Window (value < thresh_low OR value > thresh_high) 4: New sample ready

4- Vérification du Registre 0x016 :

La phase d'initialisation se termine par la re-vérification du Bit de poids faible dans le registre *0x016 "Fresh Out of Reset Register».* On change sa valeur par 0 afin de quitter la phase d'initialisation et, commencer les mesures.

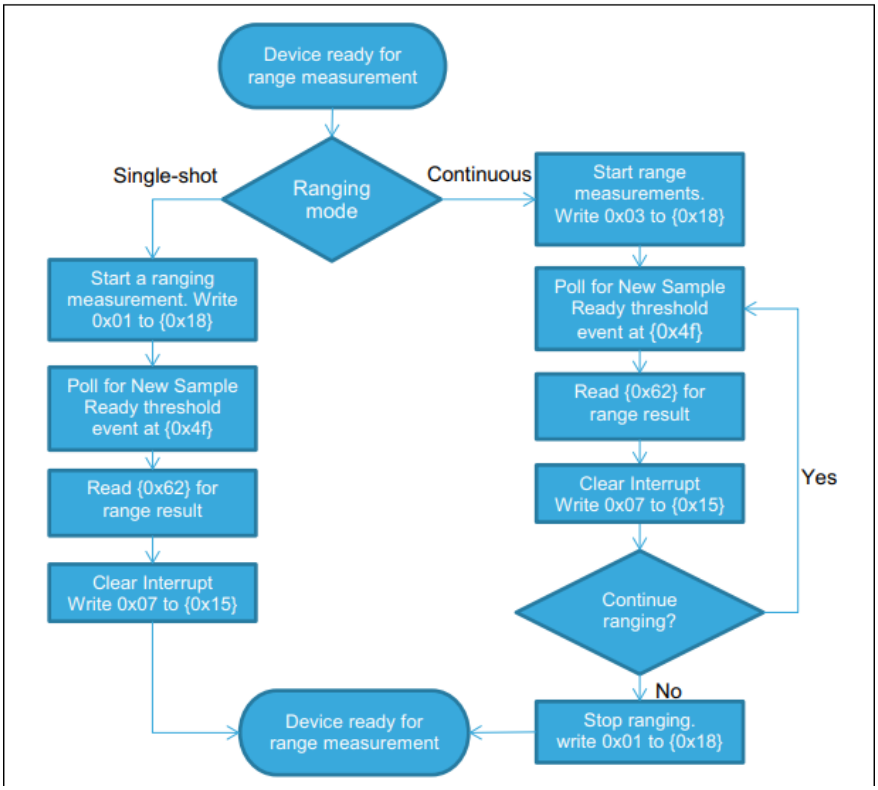
Registres et création du mode mesure continu :

Une fois le capteur initialisé, une deuxième procédure est nécessaire pour utiliser le capteur.
Dans notre cas, nous utiliserons le capteur en mode *Range continuous,* il faut donc au préalable paramétrer certains registres de l'initialisation correctement avant de démarrer le protocole pour faire des mesures.

Réalisation d'une mesure de distance en continu :

Pour réaliser cette procédure, la documentation du constructeur nous aide avec cet organigramme :

Figure 4. Flowchart for performing range measurements



Début de la série de mesures de distance :

Pour commencer les mesures il obligatoire d'écrire 0x03 dans le registre 0x018 *SYSRANGE__START (p.59)*

6.2.16 SYSRANGE__START

7	6	5	4	3	2	1	0
RESERVED						sysrange__mode_select	sysrange__startstop
R						R/W	R/W

Address: 0x018
Type: R/W
Reset: 0x0
Description:

[1]	sysrange__mode_select: Device Mode select 0: Ranging Mode Single-Shot 1: Ranging Mode Continuous
[0]	sysrange__startstop: StartStop trigger based on current mode and system configuration of device_ready. FW clears register automatically. Setting this bit to 1 in single-shot mode starts a single measurement. Setting this bit to 1 in continuous mode will either start continuous operation (if stopped) or halt continuous operation (if started). This bit is auto-cleared in both modes of operation.

Cela permet de sélectionner le mode *Ranging_Continuous*:
Cette partie du programme est à mettre dans la boucle d'initialisation

2- Boucle de mesure - Interrogations de flag

Cette partie du programme se situe dans le *void loop()* du programme.
Dans un premier temps, on interroge le registre *0x04f RESULT__INTERRUPT_STATUS_GPIO* pour savoir si un des 4 flags possibles (sur les Interruption de *Range measurements*) est levé. On réalise donc un masquage pour garder uniquement les 3 premiers bits du registre

6.2.39 RESULT__INTERRUPT_STATUS_GPIO

7	6	5	4	3	2	1	0
result_int_error_gpio		result_int_als_gpio			result_int_range_gpio		
R		R			R		

Address: 0x04F

Type: R

Reset: 0x0

Description:

[7:6]	result_int_error_gpio: Interrupt bits for Error: 0: No error reported 1: Laser Safety Error 2: PLL error (either PLL1 or PLL2)
[5:3]	result_int_als_gpio: Interrupt bits for ALS: 0: No threshold events reported 1: Level Low threshold event 2: Level High threshold event 3: Out Of Window threshold event 4: New Sample Ready threshold event
[2:0]	result_int_range_gpio: Interrupt bits for Range: 0: No threshold events reported 1: Level Low threshold event 2: Level High threshold event 3: Out Of Window threshold event 4: New Sample Ready threshold event

*Pour éviter les problèmes de mesures, on réalise une réinitialisation de ce registre par sécurité dans la boucle d'initialisation.

Lorsque la condition n'est plus valable (=Flag levé) cela veut dire qu'une mesure a été faite. On met en interruption la mesure de distance le temps de lire le résultat sur le registre *0x062 RESULT__RANGE_VAL.*

6.2.42 RESULT__RANGE_VAL

7	6	5	4	3	2	1	0
result__range_val							
R							

Address: 0x062

Type: R

Reset: 0x0

Description:

[7:0]	result__range_val: Final range result value presented to the user for use. Unit is in mm.

Une fois la lecture et de la distance effectuée, le registre de flag précédemment utilisé (0x04f) est remis à 0 afin de pouvoir réaliser une nouvelle mesure.

Cette boucle de mesure se répète indéfiniment tant que l'on souhaite réaliser une série de mesures. Si on souhaite mettre un arrêt à celle-ci, il faut réécrire dans le registre 0x018 mettre la valeur de 0x01.

Exécution du code sur Arduino :

Lors de l'exécution du code sur une Arduino, le capteur semble fonctionner mais très fréquemment, celui-ci ne renvoie plus d'information sur l'affichage série ou l'affichage 7-Segments. Avec certaines fonction *delay();* et *Serial.println();* placés dans la boucle *void loop ()*, il était possible de rendre le capteur "stable" et éviter ces problèmes.
Cependant cela ralentissait inutilement le programme et n'expliquait pas la source du problème. Au départ, nous avons revérifié certains paramètres des registres initialisés mais ceux-ci étaient correctement paramétrés. Nous avons aussi utilisé le registre *0x04d RESULT__RANGE_STATUS,* ce registre renvoie une valeur en hexadécimal. Les 4 bits de poids fort de ce registre peuvent renvoyer vers un code d'erreur en fonction de la valeur. Malheureusement, aucun code d'erreur que nous avons corrigé ont affecté ce problème de "stabilité".

6.2.37 RESULT__RANGE_STATUS

7	6	5	4	3	2	1	0
result__range_error_code				result__range_min_threshold_hit	result__range_max_threshold_hit	result__range_measurement_ready	result__range_device_ready
R				R	R	R	R

Address: 0x04D

Type: R

Reset: 0x1

Description:

[7:4]	result__range_error_code: Specific error codes 0000: No error 0001: VCSEL Continuity Test 0010: VCSEL Watchdog Test 0011: VCSEL Watchdog 0100: PLL1 Lock 0101: PLL2 Lock 0110: Early Convergence Estimate 0111: Max Convergence 1000: No Target Ignore 1001: Not used 1010: Not used 1011: Max Signal To Noise Ratio 1100: Raw Ranging Algo Underflow 1101: Raw Ranging Algo Overflow 1110: Ranging Algo Underflow 1111: Ranging Algo Overflow
[3]	result__range_min_threshold_hit: **Legacy register - DO NOT USE** Use instead *6.2.39: RESULT__INTERRUPT_STATUS_GPIO*
[2]	result__range_max_threshold_hit: **Legacy register - DO NOT USE** Use instead *6.2.39: RESULT__INTERRUPT_STATUS_GPIO*
[1]	result__range_measurement_ready: **Legacy register - DO NOT USE** Use instead *6.2.39: RESULT__INTERRUPT_STATUS_GPIO*
[0]	result__range_device_ready: Device Ready. When set to 1, indicates the device mode and configuration can be changed and a new start command will be accepted. When 0, indicates the device is busy.

En analysant ligne par ligne lorsque le code était exécuté, nous avons pu comprendre que le programme restait bloqué dans la boucle *while()* qui servait à interroger le flag d'interruption de la mesure.
En regardant des programmes de démo réalisés par d'autre utilisateurs il m'a été possible de trouver une solution pour contourner ce problème de stabilité.

Ajout d'une fonction de TimeOut :

Afin de quitter la boucle *while(),* nous avons ajouté une fonction de TimeOut au sein de la fonction de mesure en continu:

```
int Range_Continuous ()
{
  //Set the Timeout Value (in ms) for the timeout function -- Set to 0 for
disable it
  unsigned long TimeOut = 100;

  //Recover the Start Time from the While loop
  unsigned long Start_Timer = millis();
  // Check if the flag is set to 0
  //Serial.println(Start_Timer);
  while((readByte(0x04f) & 0x07) == 0)
  {
//If the Timeout is different from 0 AND IF the Delta between The Start
Time and the millis() function is bigger than TimeOut Value then, the
function return -1.
    if ((TimeOut > 0) && ((millis() - Start_Timer) > TimeOut))
    {
      return -1;
    }
  }

   //Clear Interrupt Flag
   writeByte(0x015,0x07);
   return readByte(0x062);
}
```

- Dans un premier temps, j'initialise une variable avec une valeur quelconque en milliseconde (après tests nous avons estimé que 70ms fonctionnait correctement avec le programme). Lorsque le programme sera bloqué et dépassera cette valeur de temps, cela veut dire que le programme sera en TimeOut.

- Dans un deuxième temps j'initialise une seconde variable. Celle-ci prends la valeur de la fonction *millis()*. Cette fonction renvoie le temps écoulé (en ms) depuis la mise sous tension de l'Arduino. Il faut donc faire attention au type de valeur que l'on déclare puisque la fonction *millis()* renvoie une valeur en *unsigned long*. Une fois déclarée, la variable nous sert de référentiel pour réaliser un ΔT.

- Pour réaliser la fonction de TimeOut, on ajoute une condition dans la boucle *while()* qui interroge le registre de flag. Dans cette condition on réalise le ΔT entre le début de la boucle *while* (=*Start_Timer*) et le temps actuel. On ajoute ensuite, une deuxième condition afin de désactiver le TimeOut sans avoir besoin de commenter ou d'effacer les lignes de codes à l'aide de la comparaison du TimeOut avec 0. Si la variable *TimeOut* est égal à 0 alors la condition de *if()* ne sera jamais valable et la fonction de timeout ne sera jamais utilisée.

- Lorsque les deux conditions sont réunies, notre fonction renvoie une valeur négative. Cela permet de séparer cette valeur, des valeurs de distance que le capteur renvoie lorsque celui-ci est en fonctionnement.

Gestion de la fonction TimeOut :

Au sein du programme lorsque la fonction renvoie un *-1* la fonction effectue un arrêt de la mesure, réinitialise les paramètres chargés lors de la phase de démarrage puis ré-autorise la mesure une fois le capteur prêt.

```
if (Result<0)
  {
    Serial.println("TIMEDOUT");
    Serial.println(readByte(0x04D));
    //Reeboot the sensor registers
    //writeByte(0x0018,0x01);
    writeByte(0x0016,0x00);
    VL6180X_prog();
    writeByte(0x0016,0x01);
  }
  else {

    //Serial.println(Result);
     writeNumber(Result);
  }
```

Conclusion :

Grâce à la fonction de TimeOut, le capteur est désormais fonctionnel, il reste des erreurs souvent liées au démarrage. Un simple redémarrage de l'Arduino permet de supprimer le problème. Désormais le programme doit pouvoir évoluer afin de gérer plusieurs capteurs simultanément.

Mesure de la fréquence de mesure du capteur VL6180X :

Une fois le programme de mesure *Range Continuous* fonctionnel, nous n'avons aucun registre paramétrable ou de registre en lecture pour connaitre la fréquence à laquelle le capteur réalise ces mesures de distance.

Pour connaitre cette fréquence, il est possible d'utiliser le signal de la sortie de GPIO1 lorsque celui-ci est configuré. Quand GPIO1 est configuré correctement nous pouvons l'utiliser comme sortie d'interruption (p.55)

6.2.10 SYSTEM__MODE_GPIO1

7	6	5	4	3	2	1	0
RESERVED		system__gpio1_polarity	system__gpio1_select				RESERVED
R		R/W	R/W				R/W

Address: 0x011

Type: R/W

Reset: 0x20

Description:

[5]	system__gpio1_polarity: Signal Polarity Selection. 0: Active-low 1: Active-high
[4:1]	system__gpio1_select: Functional configuration options. 0000: OFF (Hi-Z) 1000: GPIO Interrupt output
[0]	Reserved. Write 0.

L'interruption est active lorsque le capteur a réalisé une mesure, et retourne a son état initial lorsque la mesure a été affiché sur l'afficheur 7 segments :

6.2.12 SYSTEM__INTERRUPT_CONFIG_GPIO

7	6	5	4	3	2	1	0
RESERVED		als_int_mode			range_int_mode		
R		R/W			R/W		

Address: 0x014

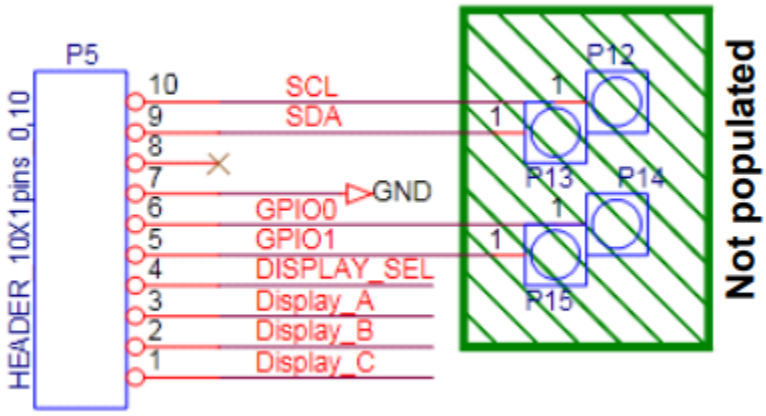
Type: R/W

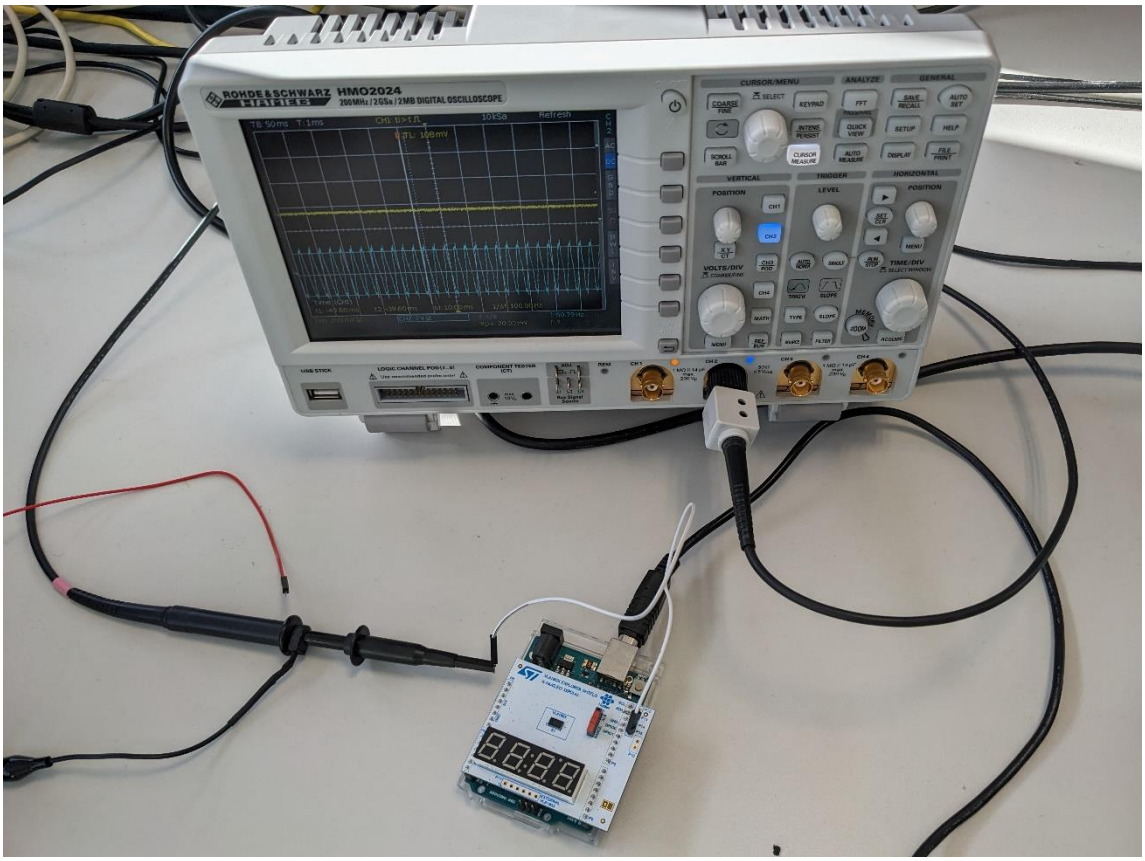
Reset: 0x0

Description:

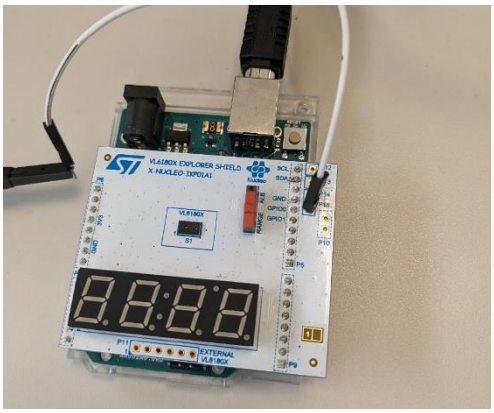
[5:3]	als_int_mode: Interrupt mode source for ALS readings: 0: Disabled 1: Level Low (value < thresh_low) 2: Level High (value > thresh_high) 3: Out Of Window (value < thresh_low OR value > thresh_high) 4: New sample ready
[2:0]	range_int_mode: Interrupt mode source for Range readings: 0: Disabled 1: Level Low (value < thresh_low) 2: Level High (value > thresh_high) 3: Out Of Window (value < thresh_low OR value > thresh_high) 4: New sample ready

On réalise ensuite le montage avec l'oscilloscope d'après le schéma de la documentation, on place la sonde sur la borne P15 de la carte Shield VL6180X :





La masse est prise directement sur le port USB (câble Rouge)



Branchement sur la Shield VL6180X

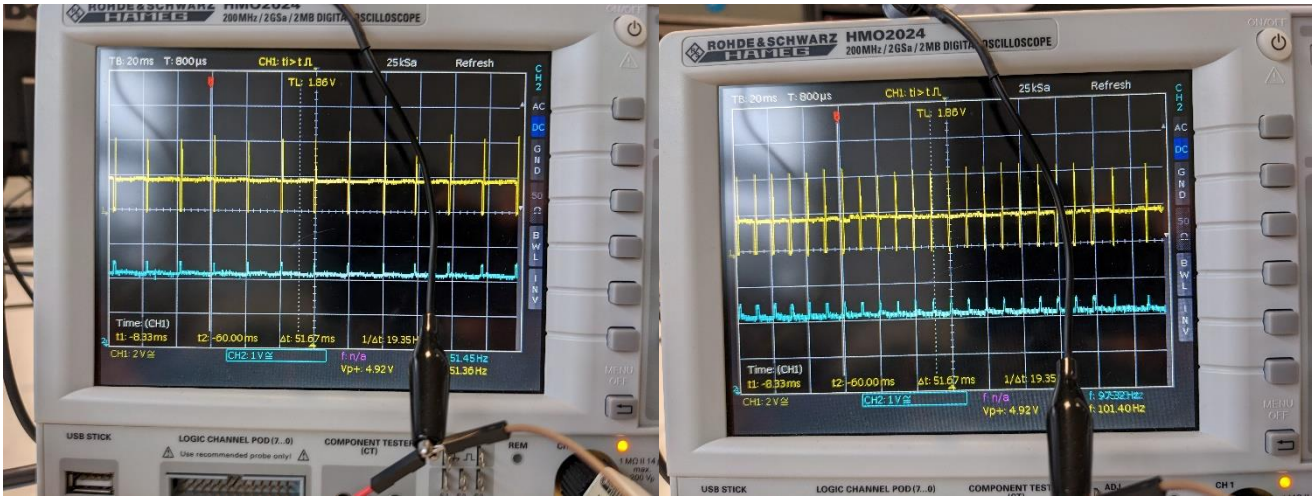
Résultat des mesures :

A l'oscilloscope, on mesure une fréquence d'environ 17Hz lorsqu'aucun obstacle n'est présent devant le capteur (Affiche *255* sur le 7 segment).

Lorsqu'un obstacle est présent entre environ 10cm et 15cm (100 et 150 sur le 7 segments), la fréquence de mesure augmente jusqu'à 50 Hz environ.

Lorsque l'obstacle est présent en dessous de 10 cm sa fréquence augmente de nouveau jusqu'à 100Hz.

Ces mesures correspondent bien au dossier technique fourni par le constructeur.



Signaux obtenus sur GPIO1 (en jaune) pour 50Hz & 100Hz sur l'oscilloscope

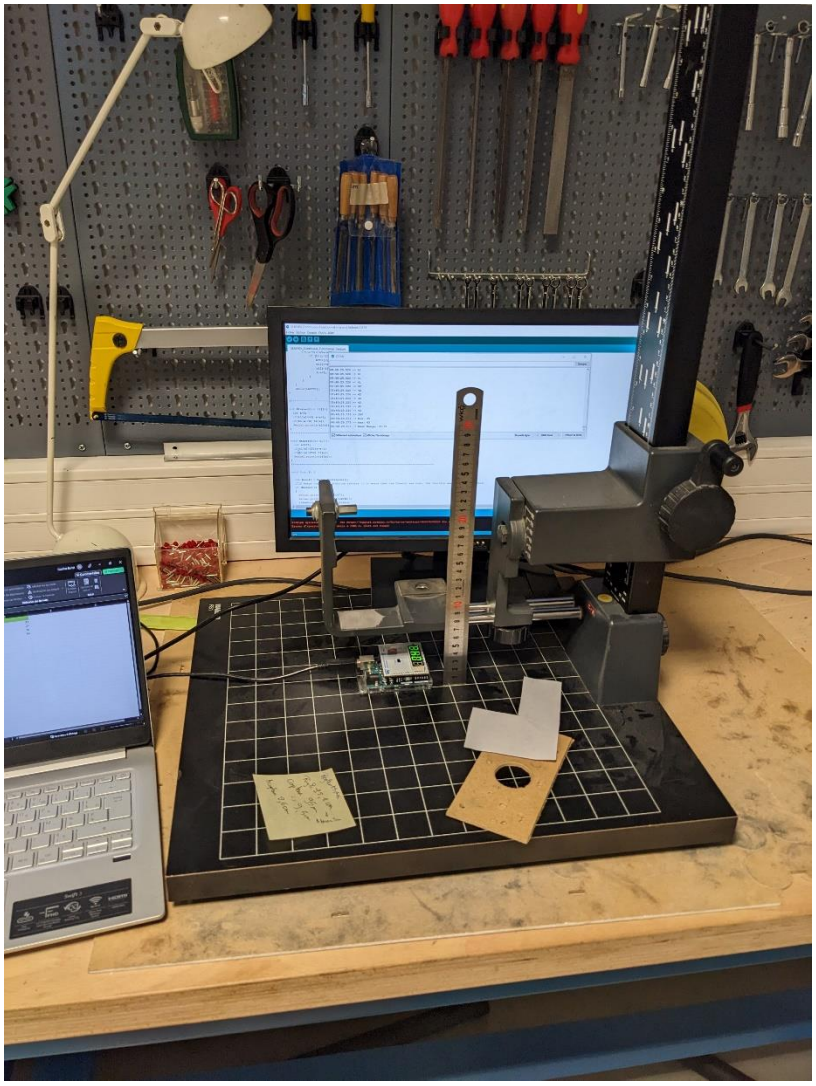
Note : Il est possible de ralentir cette fréquence si le besoin est nécessaire en changeant la valeur dans le registre 0x001b SYSRange Intermeasurments period

Mesure de la précision du capteur VL6180X :

Une fois le programme de permettant de réaliser des mesures en continu terminé. Nous n'avions aucune idée de la précision réelle du capteur (hormis les données fournies par le constructeur). De plus, dans un second temps, nous voulions nous assurer que peu importe la surface sur laquelle le laser réfléchit, la mesure retournée était correcte. Pour répondre à ces questionnements, nous avons mené plusieurs séries de mesures pour s'assurer du bon fonctionnement du laser.

Réalisation du banc de mesure

Pour réaliser nos mesures nous avons dû concevoir un banc de test à l'aide d'une règle et d'une platine ajustable en hauteur :



Au dos de la platine, nous avons accroché plusieurs surfaces (Surface claire/sombre/réfléchissante) afin de simuler les différents revêtements que le robot serait susceptible de rencontrer.

Procédure de mesures :

Pour chaque revêtement, une série de 13 mesures a été réalisée. On espace entre chacune de ces mesure la platine de 1cm.

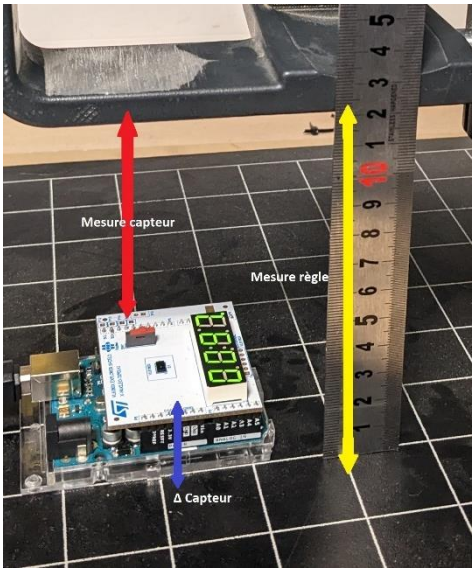
Un premier problème est auquel nous avons dû nous confronter pour obtenir une mesure fiable était le fait qu'il est difficile de lire la valeur mesurée par le capteur sur l'écran 7 segment du shield Arduino (en effet, la valeur du capteur varie toujours de quelques millimètres lorsqu'une mesure est effectuée). Pour remédier à ce problème, nous avons décidé de réaliser 240 mesures à chaque position de platine et d'en extraire la moyenne. Pour réaliser cela, nous avons modifié le programme Arduino afin qu'il puisse stocker la série de 240 mesures sous forme d'un tableau. Grâce a ce tableau nous somme ensuite capable de calculer la moyenne mais aussi, la valeur minimale et maximale. Elles nous permettent de s'assurer que les valeurs obtenues ne soient pas trop dispersées par rapport à la moyenne.

Un second problème est apparu lorsque nous avons commencé à réaliser nos séries de 240 mesures. En effet, la moyenne qui résultait des séries de mesures était très basse, (qu'importe la distance à laquelle la platine était séparée du capteur). Cela était dû au fait qu'à chaque démarrage du capteur, celui-ci mettait un certain temps avant d'être capable de réaliser une mesurer fiable. Le début du tableau était donc rempli de valeur inexploitable. Pour supprimer ces mesures indésirables nous avons décidé d'enregistrer les valeurs dans le tableau après $x^{ème}$ mesure réalisé par le capteur. Grâce à cela, le nombre de valeur autour de 255 et de 0 a beaucoup baissé. Pour les supprimer entièrement les valeurs indésirables du calcul de la moyenne, nous avons ajouté une condition dans le calcul de la moyenne afin que ceux-ci ne soient pas pris en compte :

```
void MeanTab (int Tab1[]){
  unsigned int sumVal=0;
  int b=0;
  for (int g=0;g<240;g++){
    if((Tab1[g]>0)&&(Tab1[g]<200))
    {
      b++;
      sumVal=sumVal+Tab1[g];
    }
  }
  Serial.println((float)sumVal/b);
  Serial.println(b);
```

Une fois le programme terminé, il nous suffit de récupérer la distance platine/sol capteur/sol et de réaliser le calcul suivant pour connaitre la distance réelle :

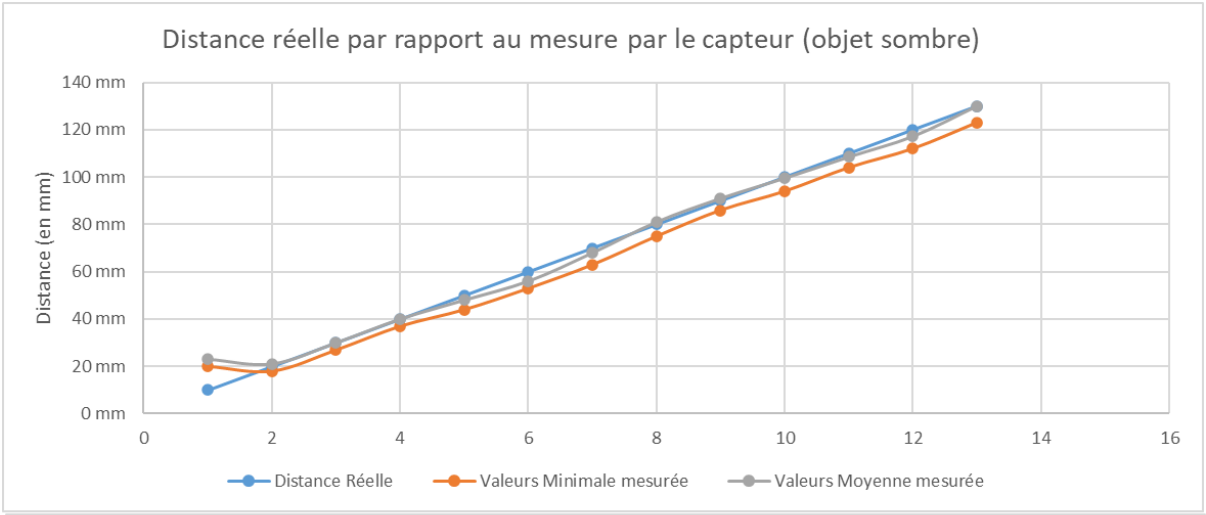
Mesure règle – Δcapteur

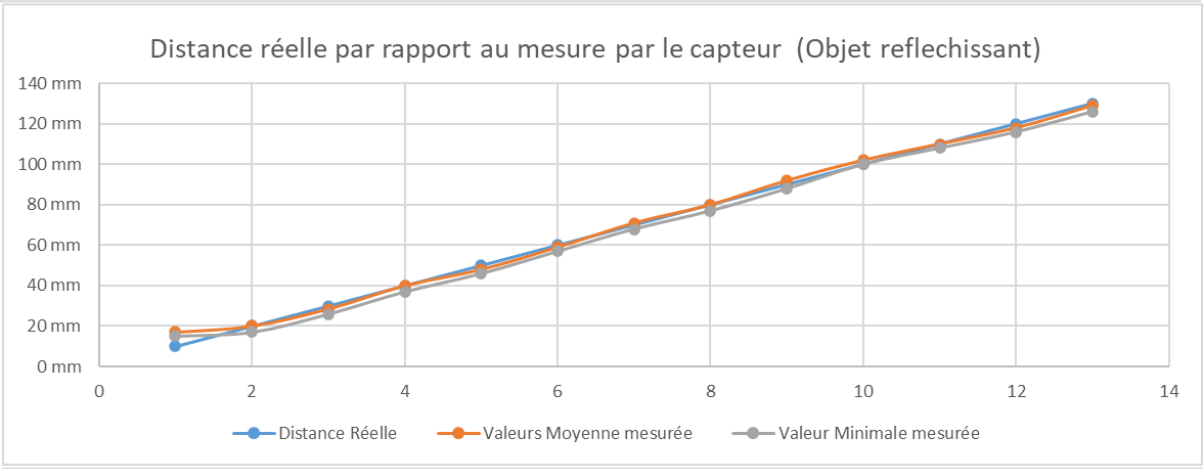


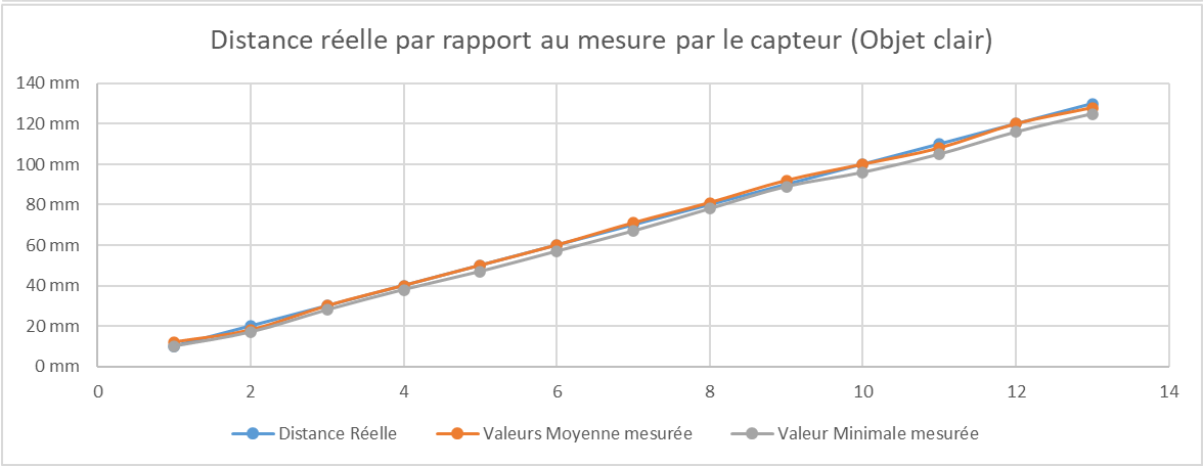
Une fois la valeur réelle et la valeur obtenue par le capteur, il suffit de les placer dans un fichier Excel pour pouvoir les comparer.

Interprétation des mesures :

Grace aux différentes mesures réalisées nous pouvons constater que le capteur fonctionne correctement peu importe le revêtement en face du capteur. De plus grâce aux graphiques nous pouvons voir que les valeurs du capteur sont très proches des distances réelles hormis pour des distances très proches (inférieures à 2cm).







Conclusion :

Pour terminer, nous pouvons dire que le projet s'est bien déroulé. Au début du projet, nous avons bien compris l'objectif du projet. Le travail a été réparti de façon qu'on puisse travailler indépendamment tout en étant complémentaire.

Au niveau de la partie hardware, nous aurions pu améliorer la lecture des documentations pour qu'elle soit plus succincte mais aussi une meilleure maitrise des enjeux de la carte afin d'être plus rapide et pouvoir avancer plus loin dans ce projet.

En revanche, il fût compliqué de s'organiser à la fin du projet. Ce manque d'organisation est principalement dû aux délais de livraison des composants mais aussi nos emplois du temps chargés entre les cours et les périodes en entreprise. Nous avons eu du mal à trouver un moment pour se réunir et tester le projet dans sa globalité et tester la carte avec le programme qui était principalement adapté pour la carte *SHIELD* pour *Arduino UNO*.

Perspectives :

Pour la partie hardware il faudrait concevoir une deuxième carte qui serait mise à l'horizontale sur le robot et où la carte capteur serait soudé à 90° dessus. L'objectif de cette deuxième carte est de récupérer les valeurs renvoyées par le capteur et de simuler des signaux logiques à partir d'un microcontrôleur. Cette demande a été faite pour faciliter la pédagogie des premières années. Dans le programme pour coder le robot, il y aura une valeur de distance limite et si elle est atteinte, les moteurs du robot se couperons. Il faudra ensuite multiplier ce fonctionnement sur tous les capteurs du robot et sur tous les robots.

Pour la partie software il serait possible d'améliorer le programme en implémentant des fonctions pour la liaison I2C. Cette liaison permettrait de mettre plusieurs capteurs sur une seul carte Arduino. Cela sera utile pour l'implémentation du système sur un robot du projet SAE1. De plus les soucis de « stabilité » persistent par moments il faudrait donc trouver le paramètre capable de rectifier le problème ou de créer une fonction capable de réinitialiser complètement le capteur afin de le rendre à nouveau opérationnel.

Résumé :

Le projet de capteur ToF est un projet de SAE de deuxième année. Le but de ce projet est de fiabiliser les capteurs embarqués sur les robots du projet *SAE Robot* de première année, en remplaçant les interrupteurs par des télémètres laser.

The ToF sensor project is a second year SAE project. The goal of this project is to improve the reliability of the sensors on the robots of the first year *SAE Robot project* by replacing the switches with laser rangefinders.