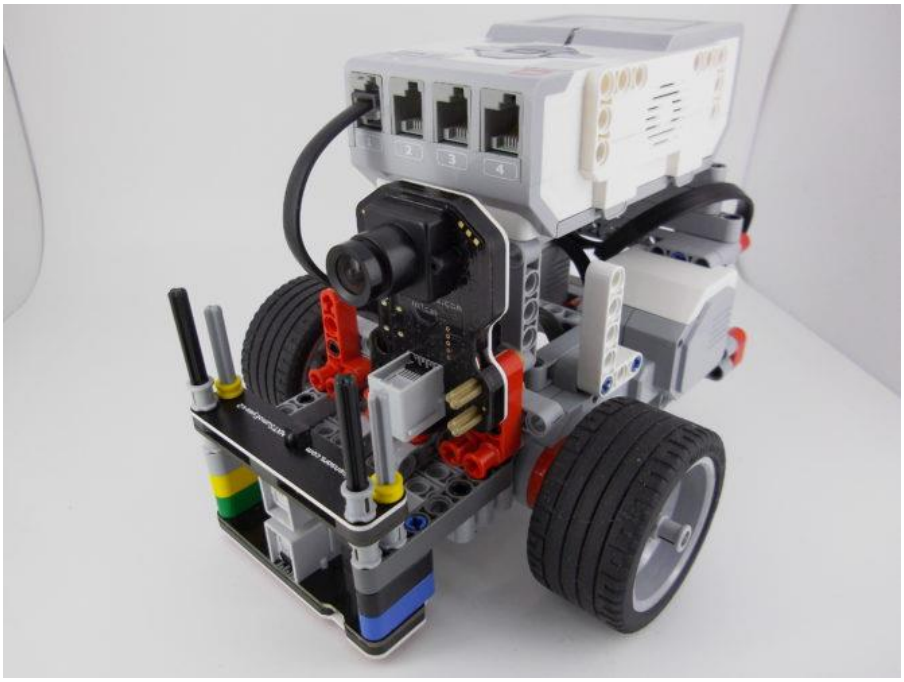
Capteur I2C Lego SumoEyes



DERTLI Osman

BOUJOU Yohan *(référent avant-projet)*

OUADHI Charles

THIMOTHEE Alexis

Sommaire

I Présentation du Lego Mindstorm, Mindsensor et le Capteur SumoEyes.

Lego Mindstorm EV3 est un jeu de construction et de robotique du jeu Lego et de la gamme Mindstorm. La série Mindstorm est la gamme "robotique programmable" de Lego. C'est une brique intelligente programmable sous Linux qui permet, via des lignes de code, de créer des routines et des interactions avec les différentes briques connectées. Se déplacer, attraper des objets, voir et reconnaitre des objets… tout cela de façon très simple et pédagogique.

Mindsensor est une entreprise réalisant des équipements de haute qualité pour le Lego Mindstorm EV3 et NXT. Il se considère comme leader du marché de capteurs ou de contrôleurs sophistiqués pour le Lego Mindstorm EV3 et NXT. Cependant , même si il déclare comme innovateur à bas prix, leurs produits coûtent chers pour ce qui propose et nous pensons avoir une chance de concurrencer leur produits.

Pour cela, notre projet consistera à réaliser un capteur similaire ou même plus performant que celui de Mindsensor tous en proposant un prix abordable. Le capteur choisit sera le SumoEyes.

Le SumoEyes est un capteur d'obstacle adapté pour le Lego Mindstorms. Il peut détecter un obstacle devant, à droite et à gauche de lui dans un rayon d'environ 20 cm. De plus, il utilise une technologique analogique de base pour communiquer avec la brique Lego. On souhaite implanter la communication I2C , compréhensible pour la brique Lego, afin d'avoir une communication plus innovante et plus efficace. Dans ce cas , il faut commencer par effectuer une analyse du cahier des charges .

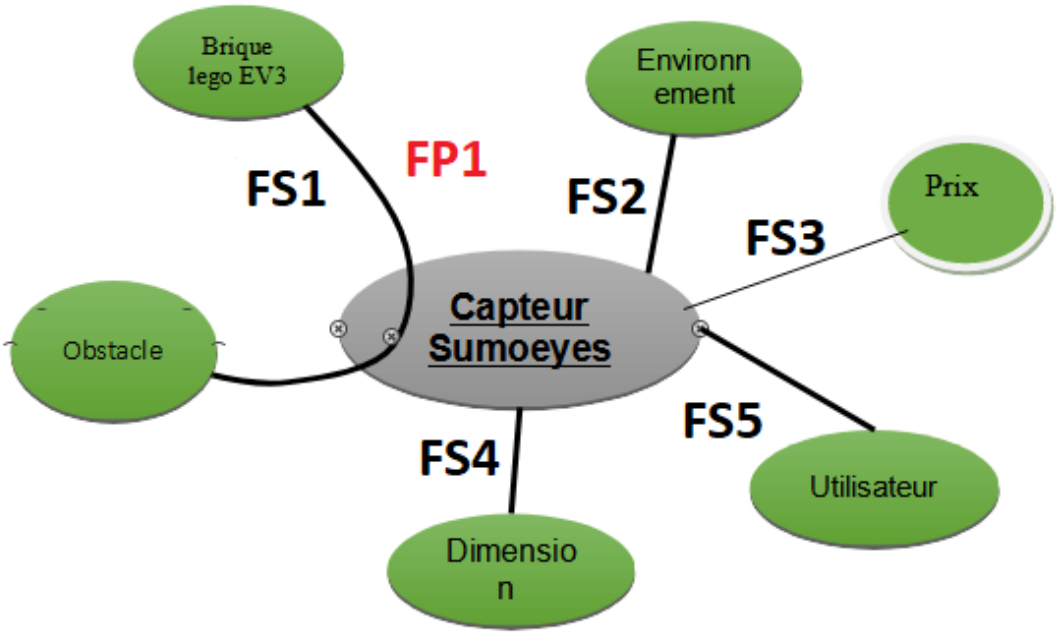
II – Analyse du cahier des charges :

1. Analyse fonctionnelle

Dans un premier temps, pour reproduire un capteur SumoEyes avec des caractéristiques similaires tous en améliorant certains parties, il faut effectuer une analyse fonctionnelle pour savoir quoi faire, comment et pourquoi.

Ce capteur va donc servir à la brique Lego, agira comme détecteur de danger et son but et de remplir les mêmes fonctions que le capteur d'origine mais avec un prix de production inférieur.

Nous continuons l'analyse fonctionnelle en définissant quelles sont les fonctions principales et secondaires de notre capteur. On peut résumer ces fonctions grâce à un diagramme en pieuvre :



Fonction	Formulation	Critères	Niveau
FP1	Détecter les obstacles suivant une direction droite ou gauche.	Technologie Infrarouge	Portée 20 cm
FS1	Communiquer avec la brique Lego	I2C	
FS2	Supporter les chocs	Résiste aux chocs	
FS3	Avoir un prix moins cher que celui vendu par Mindsensors	Prix	Inférieur à 20€ ± 10€
Fs4	Avoir des dimensions similaires que celui du capteur d'origine	Dimensions	70*25*15mm ± 2 mm
FS5	Être ergonomique et esthétique	Couleurs et forme	

2. Analyse du travail à effectuer et la répartition des tâches

Maintenant que l'analyse fonctionnelle a été effectuer, on va commencer à diviser le travail en plusieurs parties :

- **Osman**
  - Choix des composants
  - Choix de la méthode pour le capteur
  - Réalisation de la carte sous KiCad

- **Yohan**
  - Analyse du SumoEyes
  - Analyse des trames des différents capteurs, leur fonctionnement
  - Programmation I2C, Conversion analogique, Allumage des Leds
  - Fonctions RobotC, bibliothèques du capteur

- **Alexis**
  - Analyse des dimensions du capteur
  - Réalisation du Carter sur FreeCad
  - Réalisation de convertisseurs RJ-12 sur KiCad

- **Charles**
  - Fonctionnement complet du capteur infrarouge
  - Réalisation d'outils permettant d'observer les trames

Fabrication du capteur SumoEyes au grand public / production industrielle.

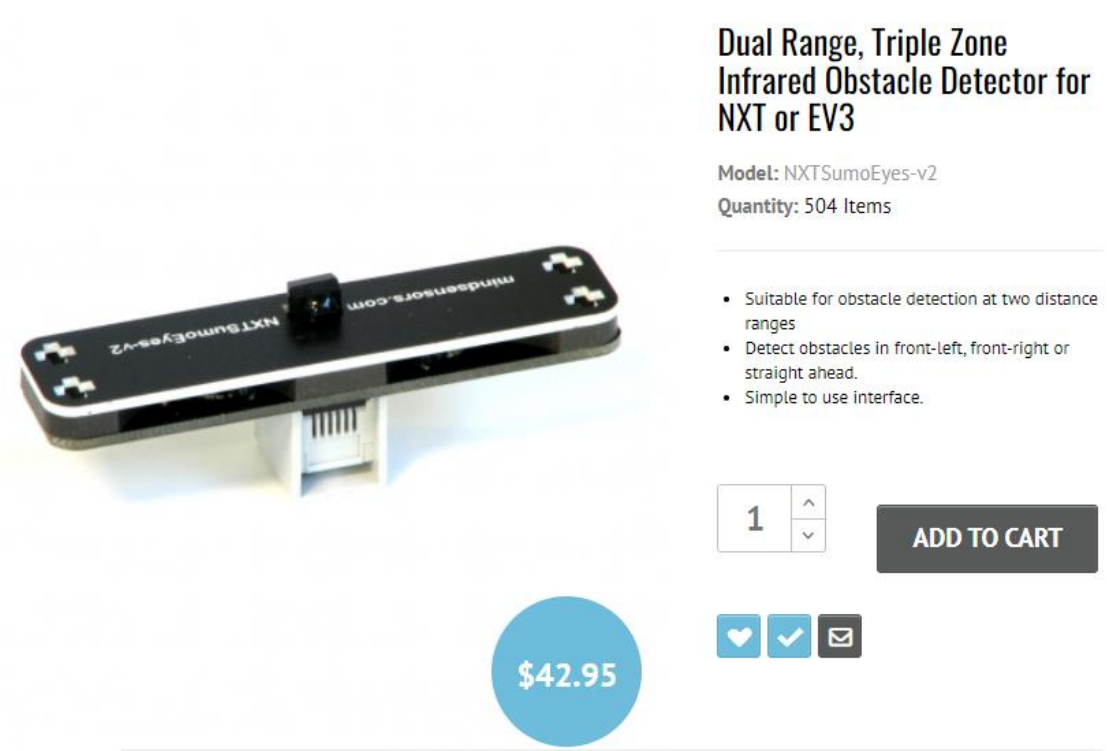
Osman Dertli

Dans cette partie, on parlera la fabrication d'un capteur d'obstacle similaire à celui du SumoEyes pour le grand public. Ce capteur doit avoir les mêmes caractéristiques et les mêmes performances.

1. Analyse du capteur concurrent / Cahier des charges

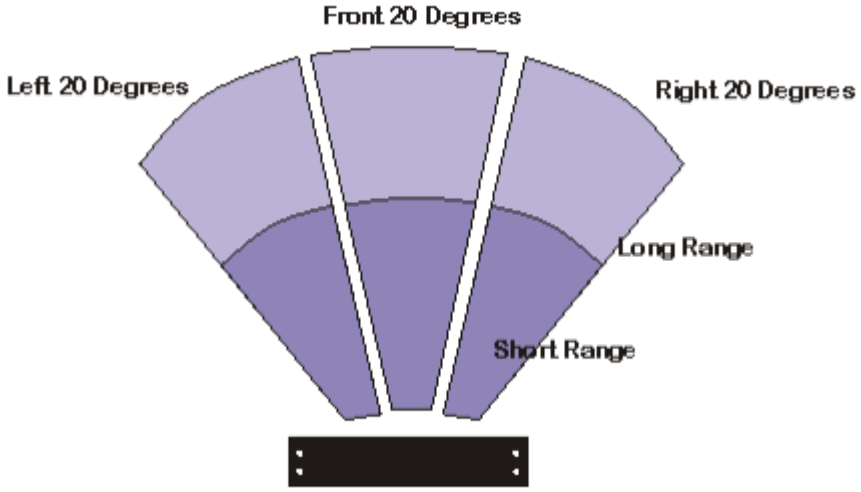
L'objectif est d'analyser le capteur concurrent pour savoir ce qu'il propose aux clients. Ainsi, on pourra faire notre cahier des charges basé sur les caractéristique du capteur concurrent

Voici le capteur à concurrencer :



En analysant le document constructeur, on relève les caractéristiques suivantes :

- Détection d'un obstacle sur une portée de 20 cm avec une couverture de 60°



- Dimensions : 16.4 mm * 72mm avec une hauteur de 25mm

Physical Specs

Weight: 0.39 oz (11.0 grams)
Foot-print: 16.4 mm x 72. mm
Height: 25 mm

- Une consommation de 2.6 mA en continue :

Current Consumption

Average measured current profile is as follows:

Current Consumption	Duration
2.6mA	Continuous

- Coût de vente : 42.95$ soit 39,10€

Ensuite, on regarde la composition du Sumoeyes, pour cela on démonte le carter du capteur :

Le Sumoeyes est constitué de :

- Microcontrôleur : PIC12F629T-I/SN
- 4 Résistances et 2 condensateurs
- 2 LED et 1 récepteur Infrarouge
- Connecteur à boucle droite RJ12 femelle

Maintenant qu'on a les caractéristiques ainsi d'une idée de la constitution du capteur, on peut faire notre cahier des charges pour cette partie. Il doit :

- Détecter un obstacle sur une portée de 20 cm avec une couverture de 60°
- Avoir des dimensions similaires : environ 70*25 mm
- Choisir un microcontrôleur pouvant effectuer le protocole I2C (contrainte dû à notre programmeur)
- Réaliser le capteur avec un coût total inférieur à 40€

Notre cahier des charges est prêt, on peut commencer la réalisation de notre capteur en commençant à choisir les composants.

2. Choix des composants

Avant de choisir les composants, plusieurs contraintes sont à prendre en compte:

• Période de crise des semi-conducteurs , le choix de composant est très restreint

• Utilisation d'un microcontrôleur pouvant être programmable sur Atmel Studio

• On souhaite que le capteur puisse communiquer avec le protocle I2C.

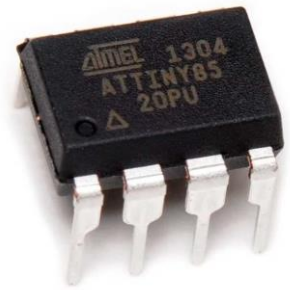
• On souhaite le maximum de composants CMS pour faciliter la production industrielle et d'utiliser le minimum

• La tension d'alimentation sera de 3,3V correspond à l'alimentation générer par le block Lego EV3

a. Microcontrôleur :

On choisit en premier le microcontrôleur pour 2 raisons, premièrement c'est le composant le plus difficile à obtenir pendant la crise, deuxièmement, c'est le coeur de la carte, c'est lui qui gère les LEDs, le récepteur, la communication avec la brique Lego avec le protocole I2C.

Le microcontrôleur choisi sera le ATTINY85-20PU :



Raison du choix :

- L'un des rares disponibles pendant la crise
- Dispose d'un périphérique USI (pour la communication I2C)
- Dispose de 2 sorties PWM pouvant envoyer un signal avec une fréquence précise aux LEDs
- Dispose de ports d'entrées pouvant recevoir le signal du récepteur
- Tension d'alimentation : 1,8 – 5,5 V (dépend des performances souhaitées)

Inconvénient :

- Microcontrôleur traversants ,il existe en version CMS(ATTINY85-20SU) ayant les mêmes caractéristiques et moins cher mais indisponible pendant cette période.

ATTINY85-20PU
MCU 8 bits, AVR ATtiny Family ATtiny85 Series Microcontrollers, 20 MHz, 8 KB, 512 Byte

Ajouter au comparateur

Quantité	Prix (sans TVA)
1+	1,90 €
25+	1,73 €
100+	1,71 €

ATTINY85-20SU
MCU 8 bits, AVR ATtiny Family ATtiny85 Series Microcontrollers, 20 MHz, 8 KB, 512 Byte

Date/Lot Code

Ajouter au comparateur

Quantité	Prix (sans TVA)
1+	1,63 €
25+	1,45 €
100+	1,42 €

b. Récepteur et LEDs :

Maintenant qu'on a choisi le microcontrôleur, une nouvelle contrainte est imposée, le courant maximale supporté par le microcontrôleur :

DC Current per I/O Pin ..... 40.0 mA

DC Current V_{CC} and GND Pins..... 200.0 mA

On choisit le récepteur infrarouge suivant, TSOP4830 :

Raison du choix :

- Tension d'alimentation : 2.5 à 5.5 V donc l'alimentation 3.3V est suffisant
- Courant d'alimentation : entre 0.55 à 0.9 mA, il faudra prévoir une résistance pour adapter le courant.

ELECTRICAL AND OPTICAL CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)						
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
Supply current	E_v = 0, V_S = 5 V	I_{SD}	0.55	0.7	0.9	mA
	E_v = 40 klx, sunlight	I_{SH}	-	0.8	-	mA
Supply voltage		V_S	2.5	-	5.5	V

- Une couverture pouvant recevoir avec un rayon 160°
- Inconvénient : Produit Traversant

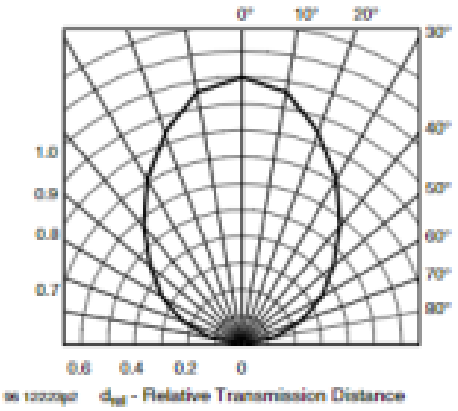


Fig. 11 - Horizontal Directivity

Cependant, on doit prendre en compte de 2 paramètres pour le choix de la LED :

• La fréquence du signal porteuse, le capteur ne peut que capter des signaux avec une fréquence précise . Dans notre cas , une fréquence de 30kHz

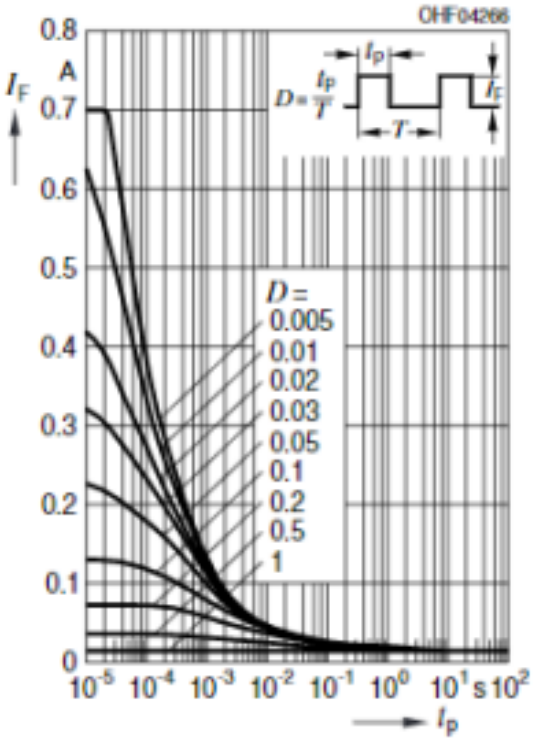
• La longueur d'onde du signal, dans notre cas une longueur d'onde 950 nm

On choisit une LED à infrarouge SFH4045N :

Raison du choix :

- LED CMS
- longueur d'onde de 950 nm
- Taille très petit : 3 mm x 2.65 mm x 1.2mm
- Positionnement au choix (perpendiculaire ou parallèle à la carte).

Selon la fréquence et le rapport cyclique du signal , la LED consomme plus ou moins de courant :



Dans notre cas, on a un signal avec rapport cyclique de 50 % et une fréquence de 30kHz soit une période de 33µs. Le courant est de 15mA par conséquent

suffisamment faible pour le microcontrôleur :

DC Current per I/O Pin ..... 40.0 mA

Il faudra prévoir des résistances pour adapter le courant pour les LEDs.

c. Récapitulatif des composants et coût des composants :

Voici un tableau récapitulatif des composants pour réaliser une carte :

Nom	Référence	Quantité	Prix unitaire	Prix totale	Prix grosse quantité
Microcontrôleur	ATTINY85-20PU	1	1,48	1,48	1,23
Récepteur	TSOP4830	1	1,17	1,17	0,476
LEDs	SFH4045N	2	0,673	1,346	0,45
Connecteur RJ12		1	1,14	1,14	1,14
Résistance 120Ω		3	0,0285	0,0855	0,0285
Résistance 33kΩ		1	0,0249	0,0249	0,0095
Condensateur 0,1µF		1	0,163	0,163	0,08
				5,4094	3,414

On a un capteur qui coûte 5,41€ en composant électronique si on achète 1 par 1 les composants, sinon si on achète en grosse quantité, on arrive à un coût de 3,41€, on dispose déjà d'une grande marge. Maintenant que nous avons tous les composants nécessaires,on prépare le bon de commande pour que les techniciens de l'IUT vérifient la possibilité de l'achat des composants souhaités :

Demande de commande

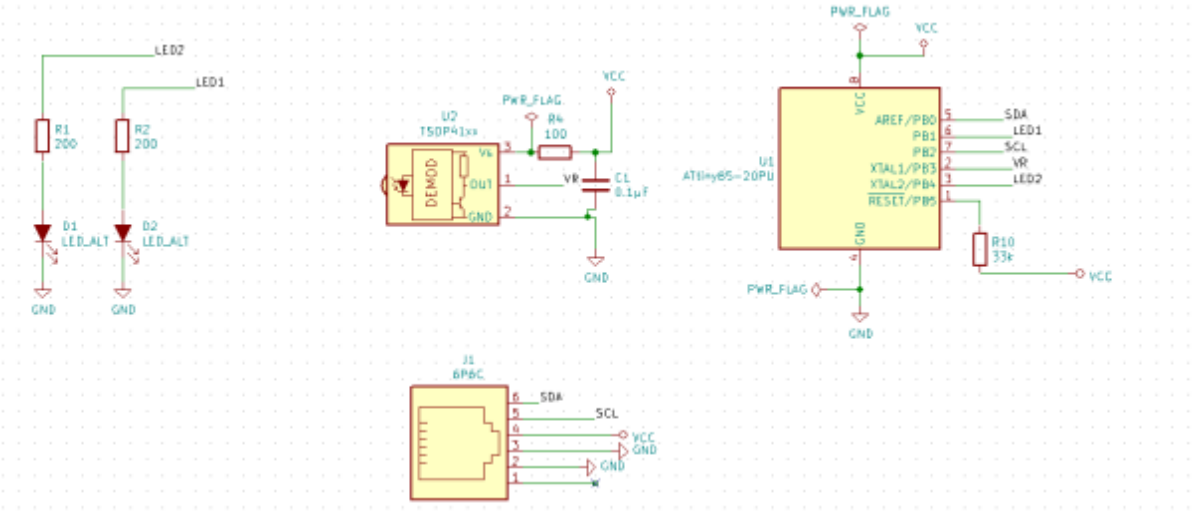
Nom du projet	Projet Capteur LEGO
Mail enseignant encadrant	ovesque@parisnanterre.fr
Mail et nom des étudiants	DERTLI Osman 40010325@parisnanterre.fr
	THIMOTHEE Alexis 40013489@parisnanterre.fr
	OUADHI Charles 40005469@parisnanterre.fr

N°	Désignation	Fournisseur	Référence	Quantité	Prix HT	Prix TTC
1	SFH 4045N	RS	876-8729	25	11,15 €	13,38 €
2	TSOP4830	RS	708-5575	5	1,86 €	2,23 €
3	B78108S1103K000	RS	191-0481	10	3,31 €	3,97 €
4	TSUS4300	RS	708-2835	25	7,70 €	9,25 €
5	ATTINY85-20SU	Farnell	1455164	2	3,38 €	4,06 €
6	ATTINY85-20PU	Farnell	1455162	2	3,28 €	3,94 €
7						
8						
9						
10						
11						
12						
13						
14						
15						
	Total				30,68 €	36,82 €

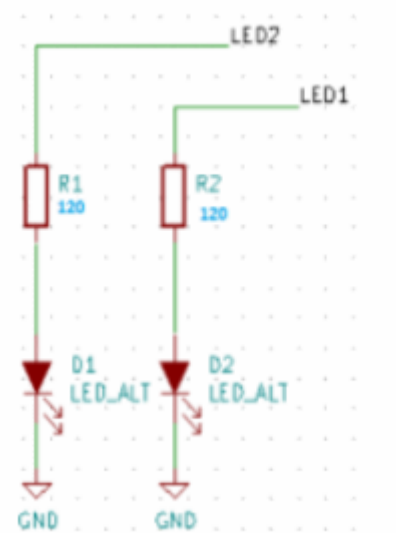
On peut commencer la réalisation du capteur.

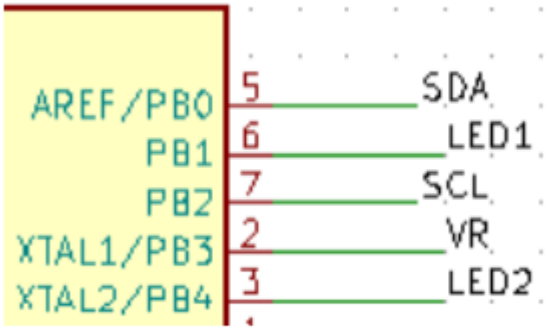
3. KiCad :

Pour réaliser la carte du capteur, on utilisera un logiciel de conception de circuits imprimés qui est KiCad. Premièrement, on effectue le schéma électrique de notre carte :



a. Partie LEDs :





Le montage des LEDs est simple, le microcontrôleur envoie sa commande via les portes PB1 et PB4. Des résistances de 120Ω sont présentes dans le but d'obtenir un courant de 15mA. Pour déterminer la résistance à appliquer il faut prendre en compte la tension de polarisation directe aux bornes de la LED.

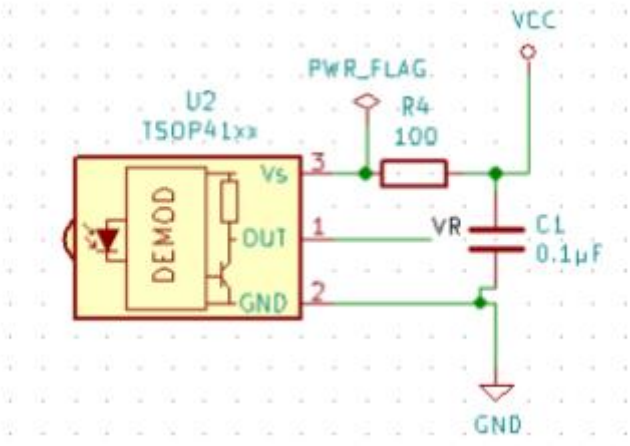
Forward voltage	V_F	typ.	1.6 V

Sachant que la tension d'alimentation est de 3.3V, la résistance à mettre est :

$$R = \frac{3.3 - 1.6}{15 * 10^{-3}} = 113.3\Omega$$

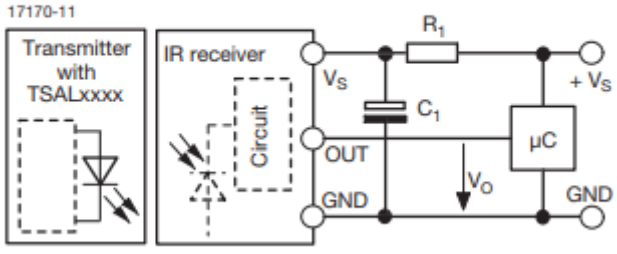
On utilisera les résistances 120Ω car c'est la valeur la plus proche disponible comme résistance normalisé.

b. Partie Récepteur :



Le montage suivant est un montage recommandé par le constructeur pour augmenter la robustesse contre la surcharge électrique. Les valeurs des résistances et du condensateurs sont les valeurs typiques données par le constructeur.

APPLICATION CIRCUIT

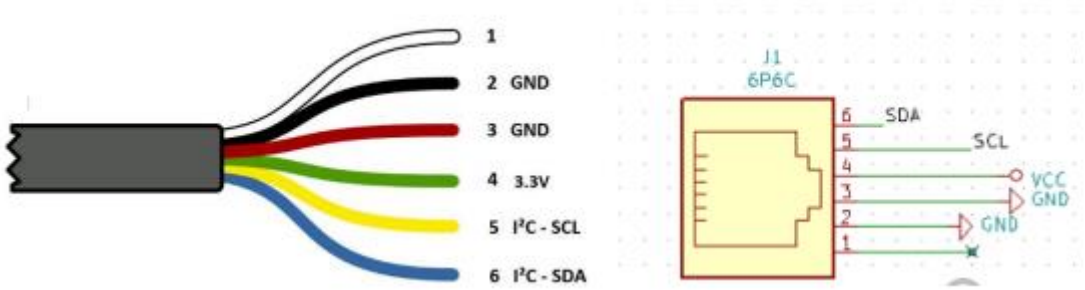


R_1 and C_1 recommended to reduce supply ripple for V_S < 2.8 V

La tension de sortie du récepteur est envoyée par le port PB3 :

c. Partie connecteur:

Voici le câble de communication entre le capteur et le block Lego :

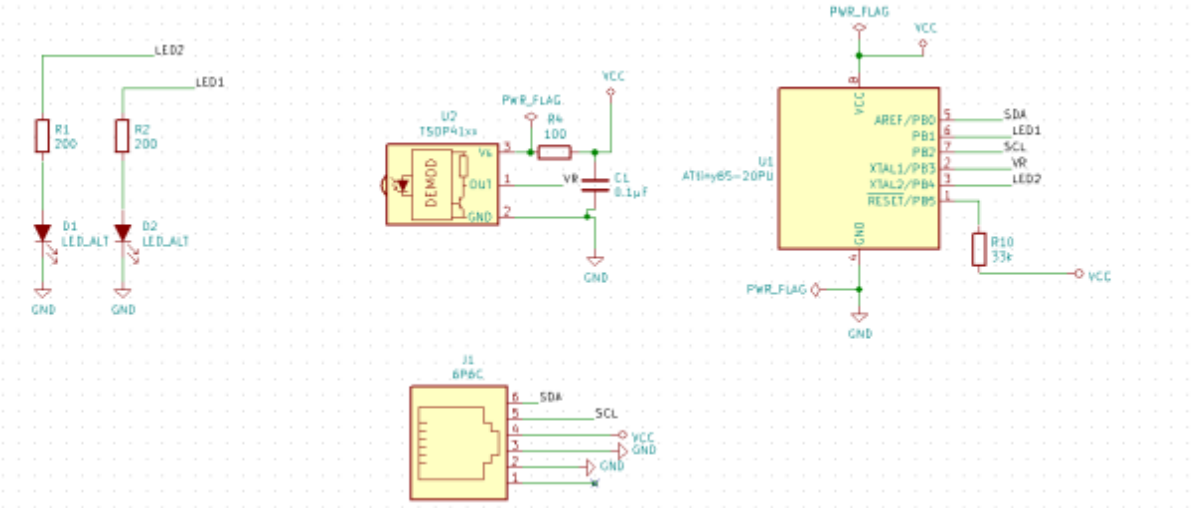


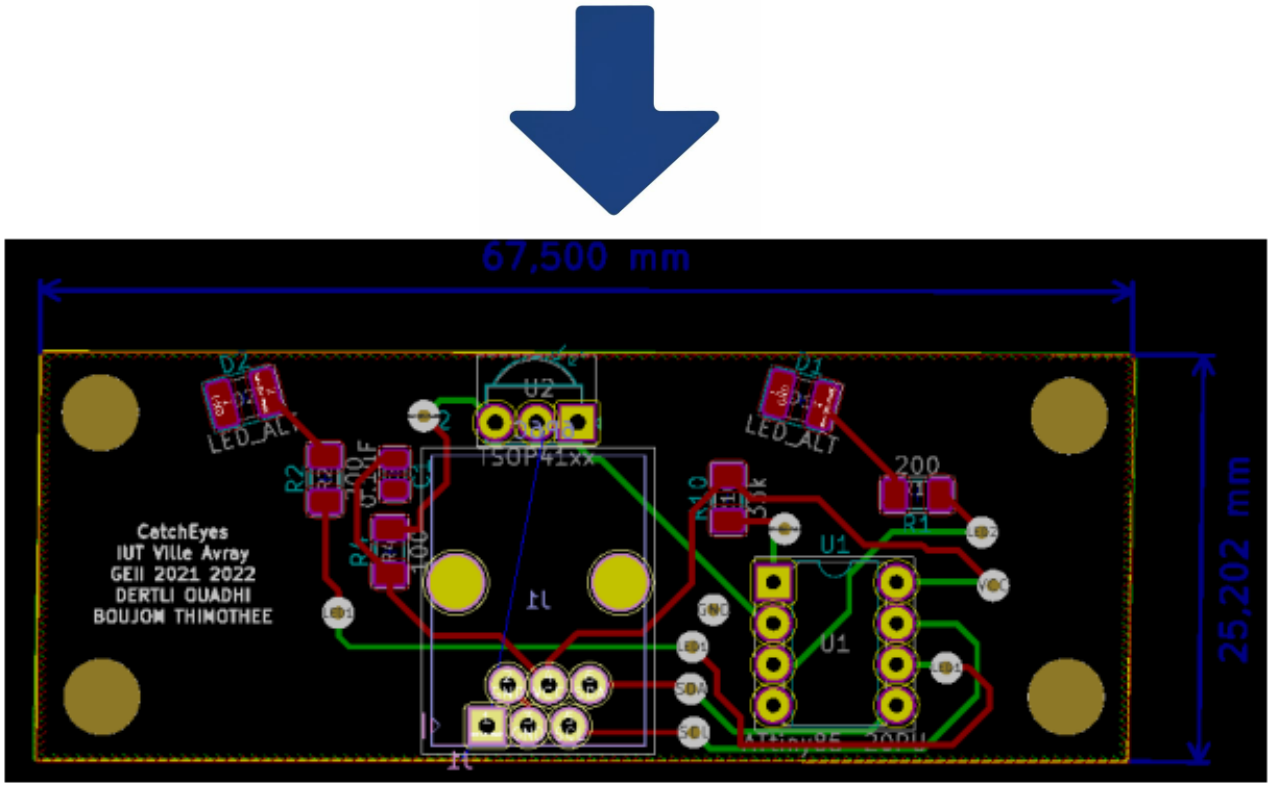
La communication I2C est envoyé sur le microcontrôleur et le câble alimentele capteur.

Maintenant que le schéma électrique est complet, on peut passer au routage.

4. Routage

Étant donné que les empreintes nécessaires sont déjà disponibles dans la bibliothèques de base de KiCad. On a peut directement effectuer le routage sans dessiner de nouvelles pièces.





On génère les fichiers de fabrications pour graver la carte. On prépare une demande de gravure de carte.

DEMANDE DE GRAVURE	
Titre du projet :	Capteur Lego SumoEyes/CatchEyes
Numéro du groupe du projet :	ProjXXX
NOM de l'enseignant encadrant :	Olivier VESQUE
NOMS Prénoms des étudiants :	Osman DERTLI
Emails des étudiants + de l'encadrant séparé par une virgule	40010325@parisnanterre.fr , ovesque@parisnanterre.fr

Date de validation du schéma électrique Par l'enseignant encadrant :	26/11/21
Dimension du circuit à graver ?	67,5mm * 25,202mm
Simple ou double face ?	double face
Les perçages des composants traversants sont-ils tous à 0,8 mm ?	y et 1mm pour les autres com
Avez-vous imprimé le typon à l'échelle 1:1 ?	ualisation à une échelle de 10
Y a t-il assez de cuivre pour souder sur les pastilles ?	oui
Tous les diamètres de pastilles sont-ils égales à 2 mm ?	oui
Toutes les pistes sont-elles de largeur supérieure à 0,4 mm ?	oui
Tous les espacements entre les pistes sont-ils supérieurs à 0,4 mm ?	oui
En double face : y a t-il des soudures impossibles à faire ? *(sous un circuit intégré par exemple)*	non
Avez-vous fait le plan de masse (simple couche) Ou les plans de masse (double couches) ?	oui
Avez-vous généré les fichiers d'extension GERBER ?	oui
Nom du dossier Kicad (type ProjXXX_Kicad)	Sumo_Public
Nombre de circuit à graver	1
Date de validation du typon Par l'enseignant encadrant :	26/11/21

5. Problème rencontrés et solutions :

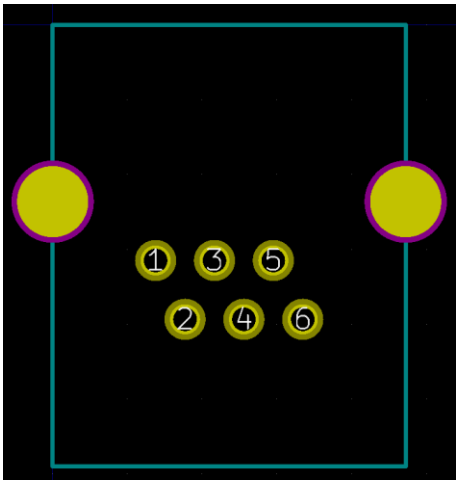
L'empreinte du connecteur RJ12 de Kicad est un connecteur avec une boucle centrale et avec des broches inversés alors que le connecteur du block Lego a une boucle droite :

Boucle centrale

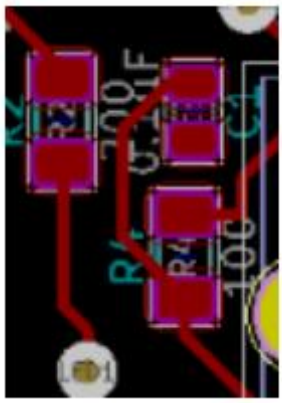
Boucle droite

Solutions :

- Réalisation d'un câble d'adaptation entre une boucle centrale et une boucle droite
- Réalisation d'une empreinte du RJ12 femelle à boucle droite



De plus, lors du soudage des composants CMS, il y a eu une erreur de soudage créant un court-circuit lors des mesures. De plus, certains composants CMS n'étaient pas correctement connectés.



Solutions :

- Réajustement de la technique de soudage CMS

- Pour le prochain routage, espacer les composants CMS proches pour éviter les court-circuits.

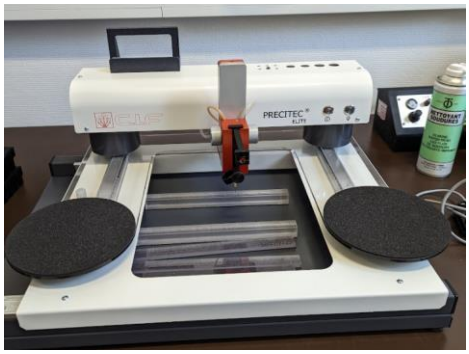
6. Fabrication de la carte V2

a. Technique de soudage

Pour souder les composants CMS, on a utilisé de la pâte conductrice d'étain :

La technique est simple, on place notre pâte sur les emplacements des composants à souder. Ensuite, à l'aide d'une pince, on place le composant en appuyant sur le composant pour que la pâte soit étalée sur la piste. En maintenant le composant, on utilise le fer à souder pour faire fondre la pâte pour connecter le composant à la piste.

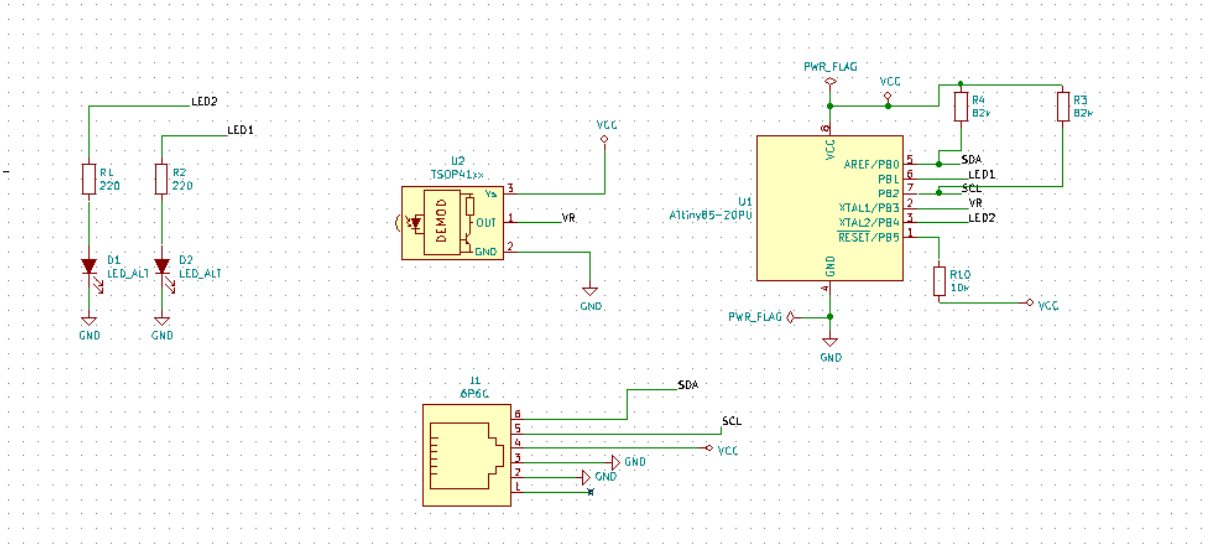
Même si des équipements sont à dispositions pour effectuer le soudage des composants CMS, les composants était assez simple à souder. :

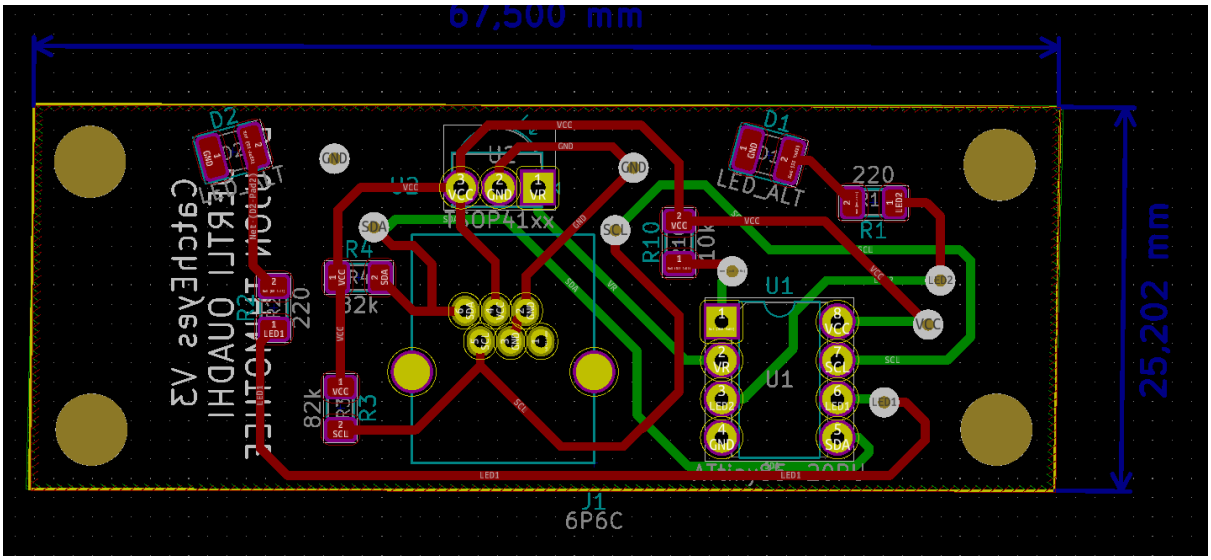


A gauche , c'est un manipulateur de CMS. Il permet de poser des composants CMS avec précision. A droite, c'est un four à refusion, il permet de faire fondre la pâte d'étain.

b. Réalisation de la carte

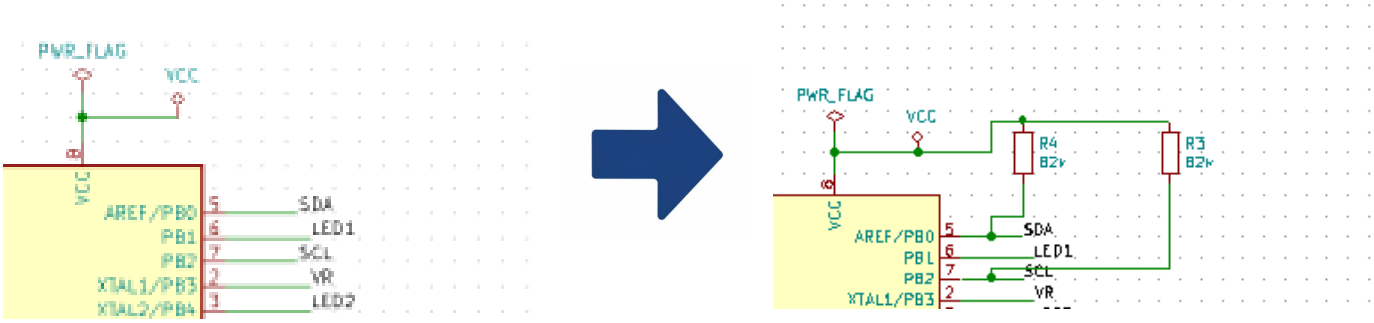
Voici le schéma électrique et le routage de la nouvelle carte :



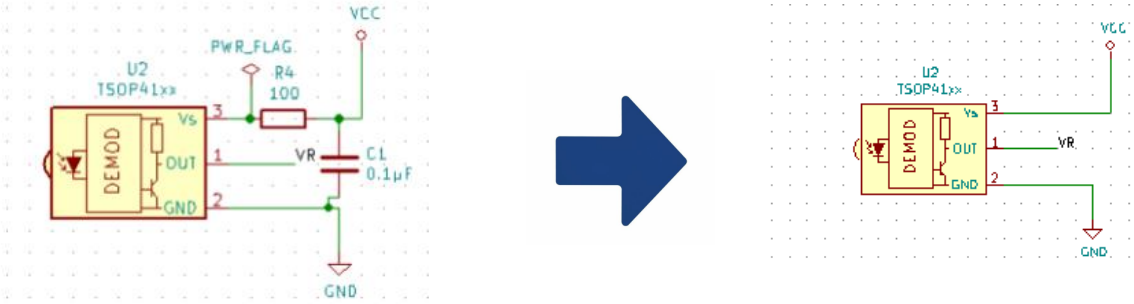


Plusieurs modifications sont présentes :

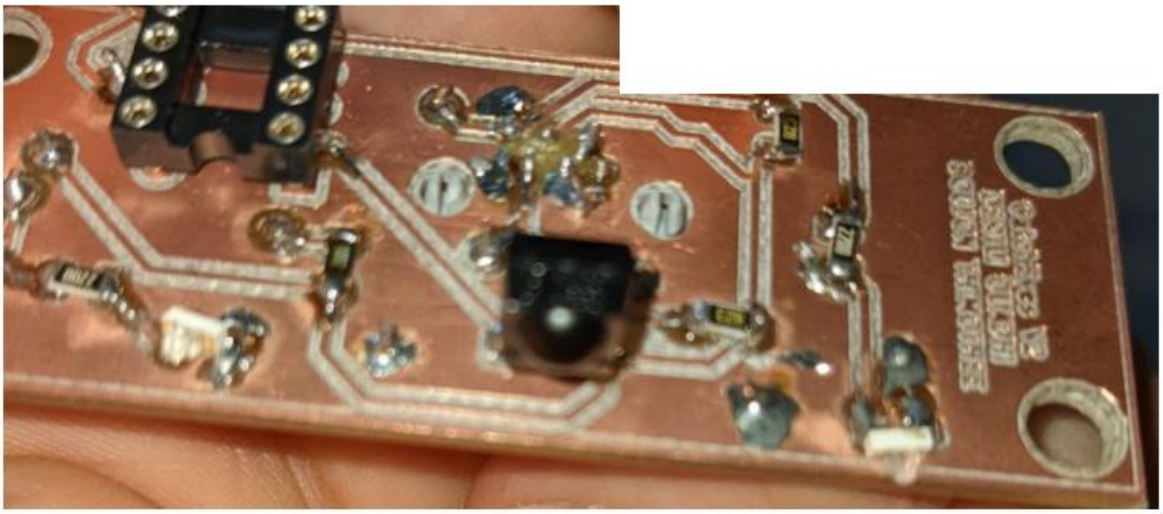
- Ajout de résistance de pull-up pour la communication I2C (Voir la partie de programmation pour savoir le choix des résistances).



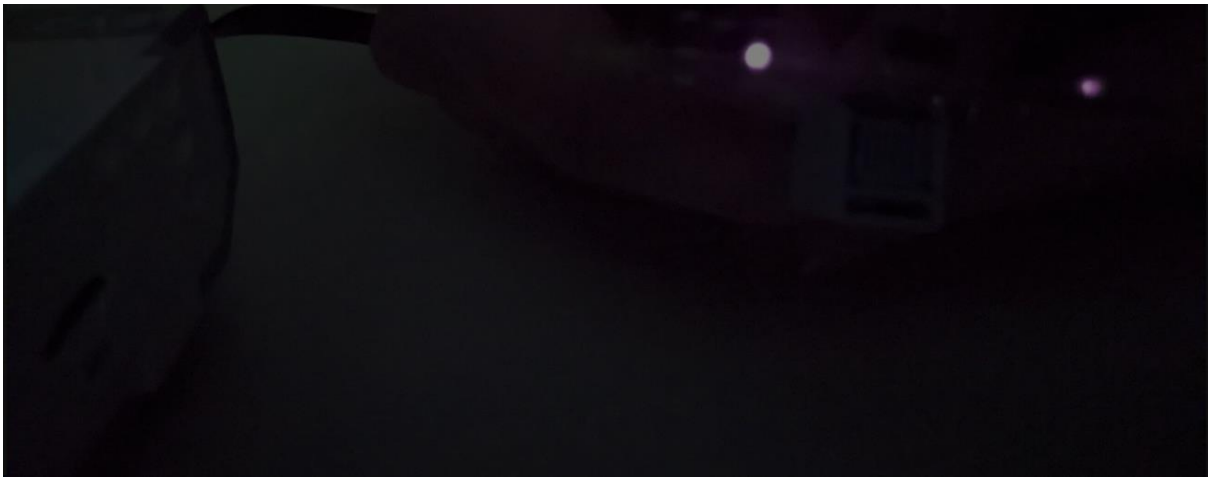
- Pour le récepteur, on avait placé une résistance et un condensateur pour améliorer la robustesse contre la surcharge électrique. Cependant, on avait remarqué que par différents tests que le signal était dégradé. Par conséquent, on enlève la résistance et le condensateur.



Voici la carte avec tous les composants soudés :



On effectue un test simple pour vérifier le fonctionnement des LEDs, on envoie un signal à une fréquence de 30 kHz et on se met dans une pièce sombre pour visualiser plus facilement les infrarouges:



On visualise bien les LEDs allumées et on remarque aussi le clignotement des LEDs ce qui signifie que la LED s'allume puis s'éteint.

7. Problèmes de surchauffe

Malheureusement, on a remarqué que le récepteur IR et le microcontrôleur surchauffent. On vérifie le courant reçu pour le microcontrôleur et le récepteur.

- Pour le récepteur, on mesure un courant de 10mA. Or, dans le document constructeur, le récepteur est conçu pour recevoir au maximum 5mA.

ABSOLUTE MAXIMUM RATINGS				
PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
Supply voltage		V_S	-0.3 to +6	V
Supply current		I_S	5	mA

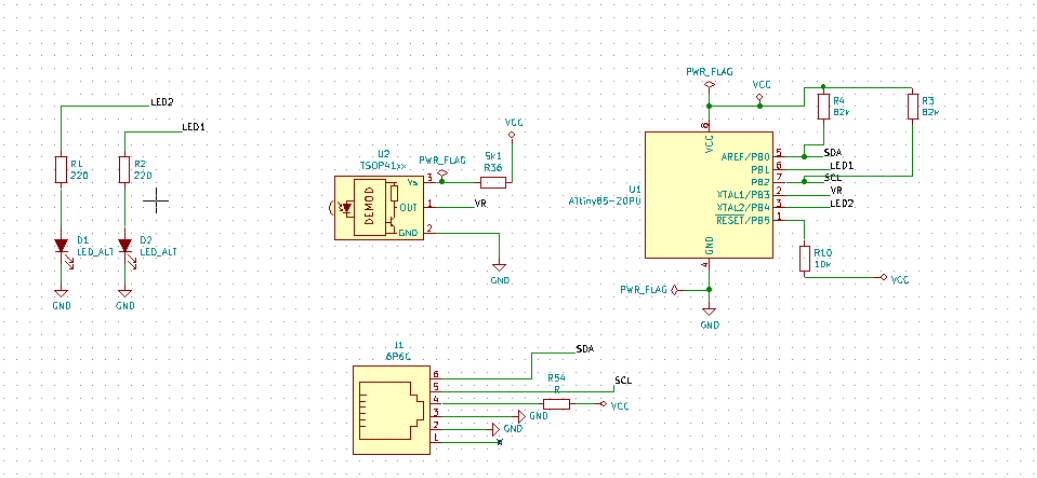
- Pour le microcontrôleur, on mesure un courant de 300mA pour l'alimentation. Or, dans le document constructeur, le microcontrôleur est conçu pour recevoir au maximum 200mA.

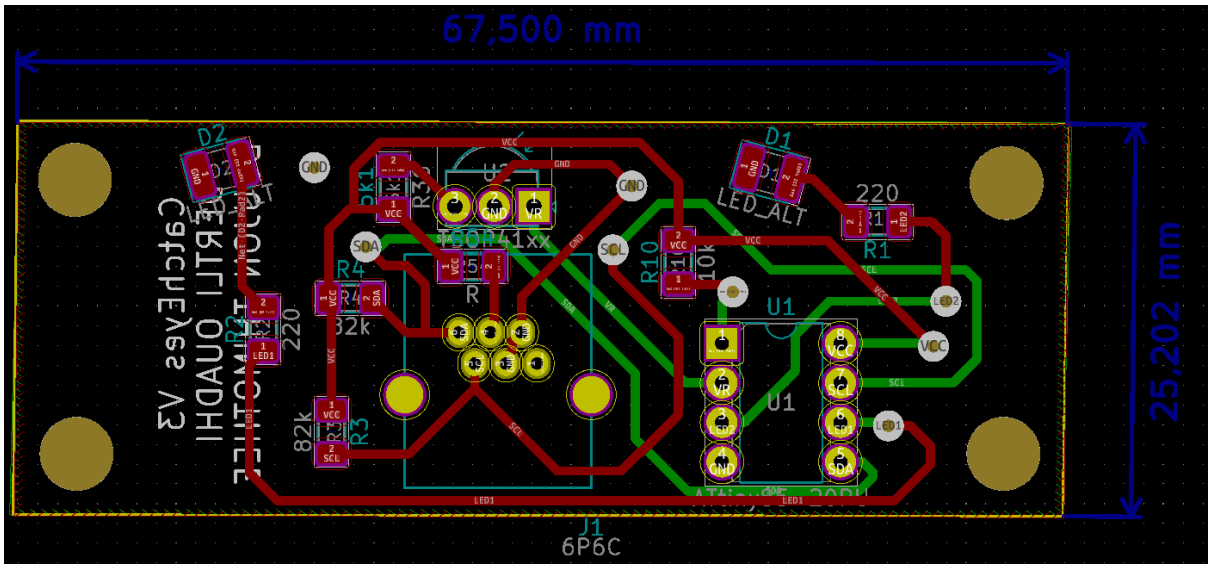
DC Current per I/O Pin ..... 40.0 mA

DC Current V_{CC} and GND Pins ..... 200.0 mA

8. Fabrication de la carte finale

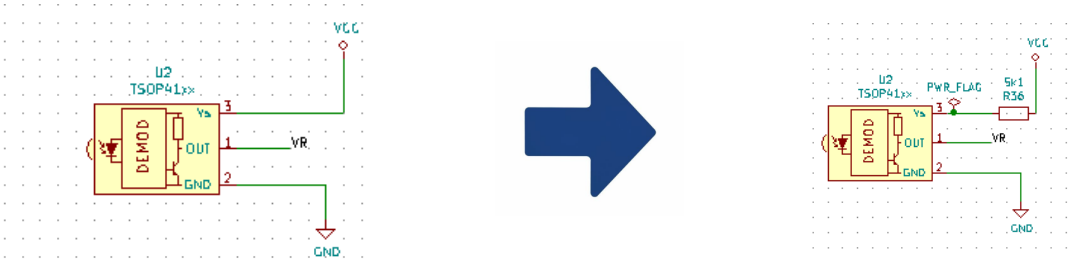
Voici le schéma électrique ainsi que le routage de la dernière carte:





Modifications ajoutés :

- Pour le récepteur, on ajoute une résistance :

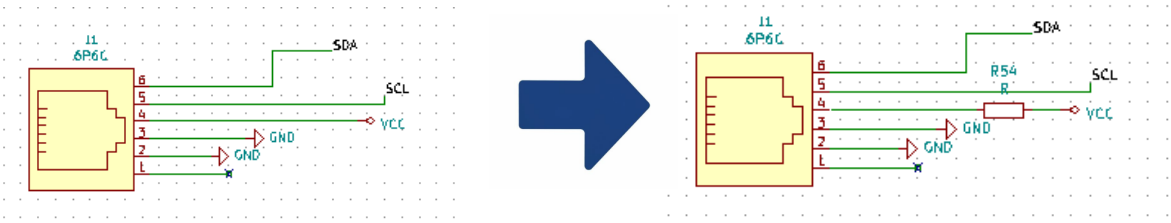


D'après le document technique, la valeur optimale de courant est de 0.8 mA (en considérant la lumière du soleil).

ELECTRICAL AND OPTICAL CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)						
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
Supply current	E_v = 0, V_S = 5 V	I_{SD}	0.55	0.7	0.9	mA
	E_v = 40 klx, sunlight	I_{SH}	-	0.8	-	mA
Supply voltage		V_S	2.5	-	5.5	V

Sachant que la tension est de 3.3V, la valeur de résistance est de 4.125 kΩ, la résistance normalisée la plus proche est 3.9kΩ.

- Pour le microcontrôleur, on ajoute une résistance :



D'après le document technique, la valeur maximale de courant est de 8 mA, on souhaite le pire cas pour être sûr.

I_{CC}	Power Supply Current [7]	Active 1 MHz, V_{CC} = 2V		0.3	0.55	mA
		Active 4 MHz, V_{CC} = 3V		1.5	2.5	mA
		Active 8 MHz, V_{CC} = 5V		5	8	mA
		Idle 1 MHz, V_{CC} = 2V		0.1	0.2	mA
		Idle 4 MHz, V_{CC} = 3V		0.35	0.6	mA
		Idle 8 MHz, V_{CC} = 5V		1.2	2	mA

Sachant que la tension est de 3.3V, la valeur de résistance est de 412.5Ω, la résistance normalisée la plus proche est 390 Ω.

Au moment de la rédaction du rapport, la carte est en production.

9. Conclusion

Ce projet lego m'a permis à obtenir plusieurs compétences différentes :

- Autonomie: le projet a été très peu encadré et l'objectif était de se débrouiller tout seul pour découvrir comment réaliser le capteur.
- Sens des responsabilités: on devait partager les différentes tâches pour réaliser le capteur.
- Techniques: la fabrication de la carte m'a permis de perfectionner le logiciel Kicad ainsi que découvrir les différents équipement pour souder et enfin l'utilisation des composants CMS

Réalisation d'une plaque Labdec prototype

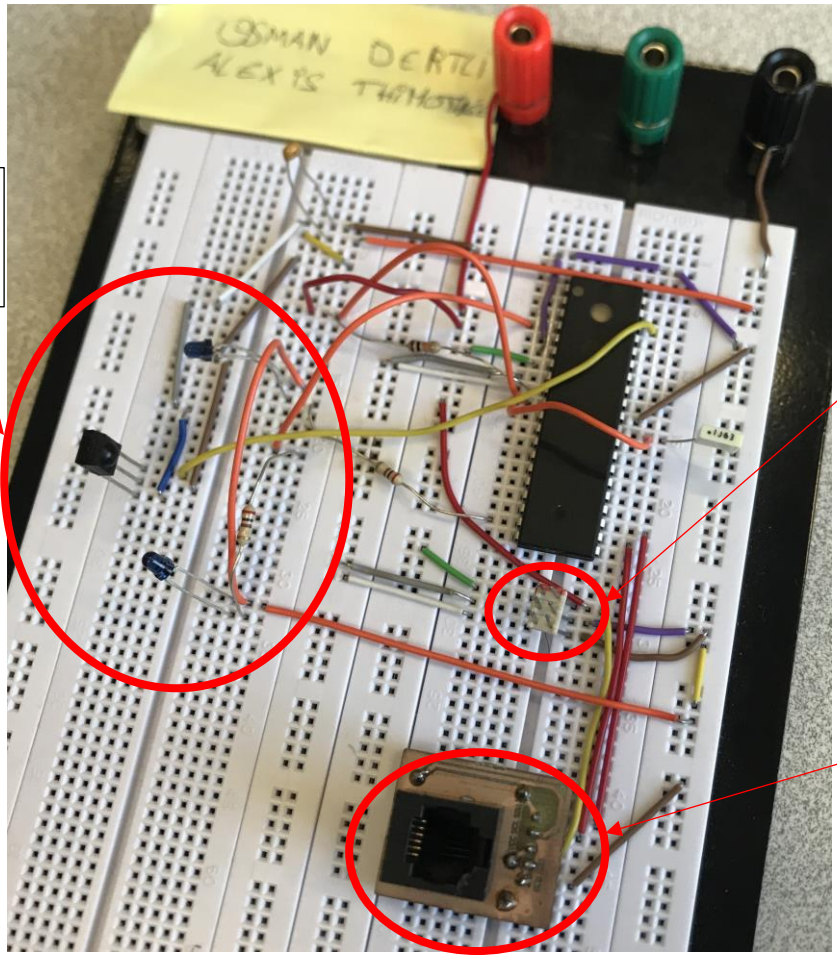
Alexis Thimothée

Afin de vérifier le fonctionnement du montage fait par Osman et moi sur KiCad avec un ATMega 16, j'ai décidé de réaliser la plaque Labdec correspondant au montage afin de tester sans avoir à gaspiller des matériaux ou de souder inutilement.

Le montage à ATMega16 et non un ATTiny85, car pour les premiers tests nous voulions avoir un matériel connu, car souvent utilisé pour de la programmation lors de la première année et donc plus facile à gérer, bien qu'il prenne énormément de place.

Le KiCad lui possédait 2 leds, 2 connecteurs de programmation, l'un pour envoyer un programme vers le microcontrôleur et l'autre pour assurer la communication entre la brick EV3 et le microcontrôleur, un microcontrôleur ATMega16 et le récepteur infrarouge constitué d'un condensateur en parallèle de 100 nF.

J'ai donc réalisé le montage suivant sur la plaque Labdec :



Sur le connecteur pour la Brick, nous avons dû dessouder le connecteur de sa plaque car il n'était pas adapté à notre câble qui lui avait un embout Ethernet classique. J'ai donc ressoudé sur cette même plaque un autre connecteur qui lui est adapté.

1. Apprentissage des bases de FreeCad

Le capteur aura besoin d'un carter en plastique, et pour cela nous devons utiliser un logiciel de CAO.

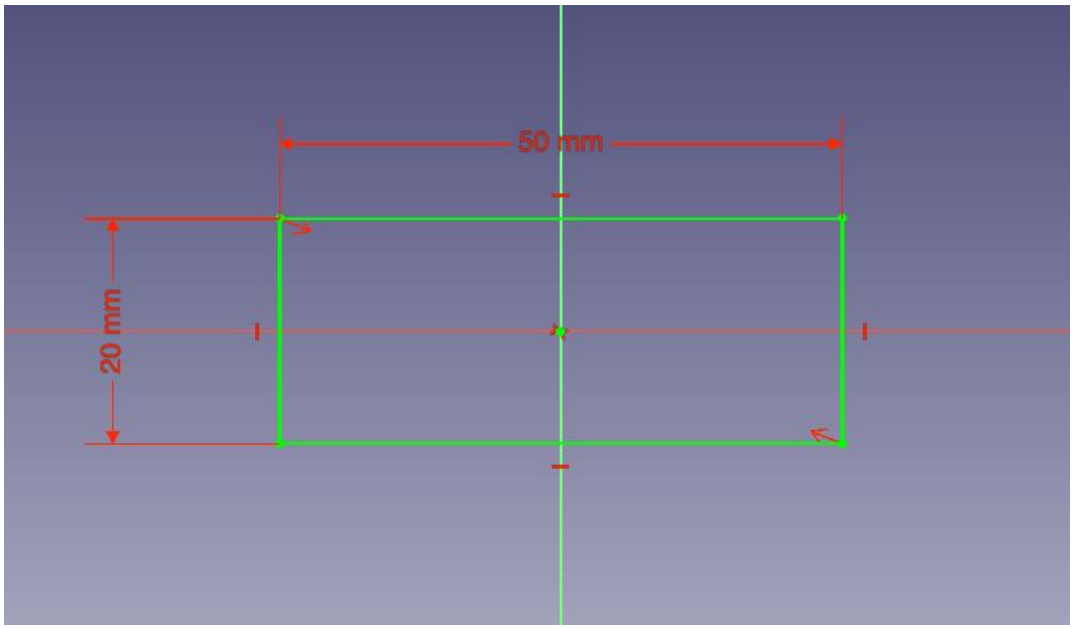
Qu'est-ce qu'un logiciel de CAO ?

La conception assistée par ordinateur, ou CAO, rassemble des outils informatiques (logiciels et matériels) qui permettent de réaliser une modélisation géométrique d'un objet afin de pouvoir simuler des tests en vue d'une fabrication.

Nous avons donc choisi d'utiliser FreeCad. Je me suis entrainé a réalisé les bases :

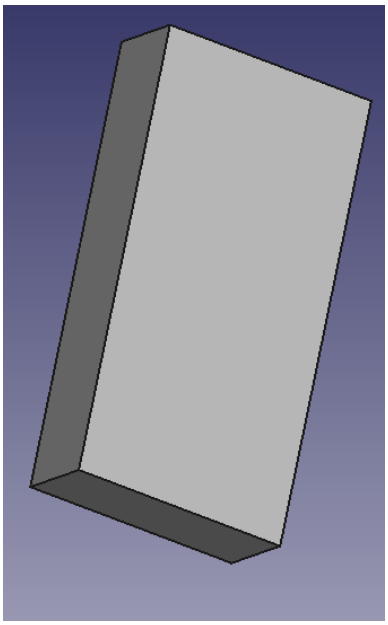
Sketch avec une contrainte de 50 mm en longueur et 20 mm en largeur ainsi qu'un centrage au centre de notre plan :

Le

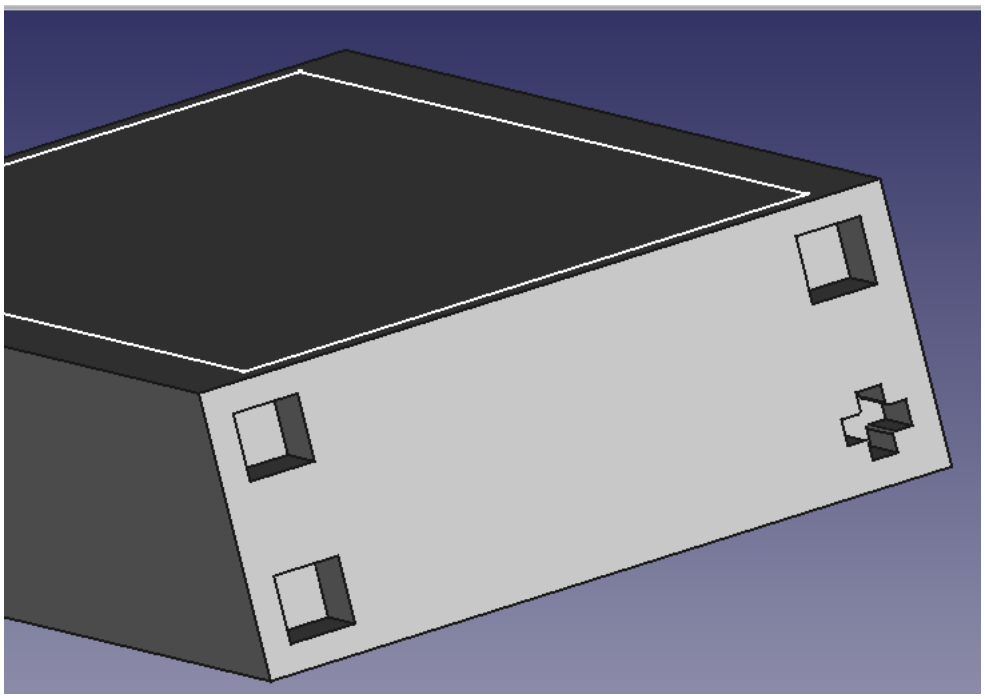


sketch est la pour que l'on réalisé la version 2D comme un dessin et on y ajout des contraintes afin d'avoir de bonne dimension

Création de sa version 3D :



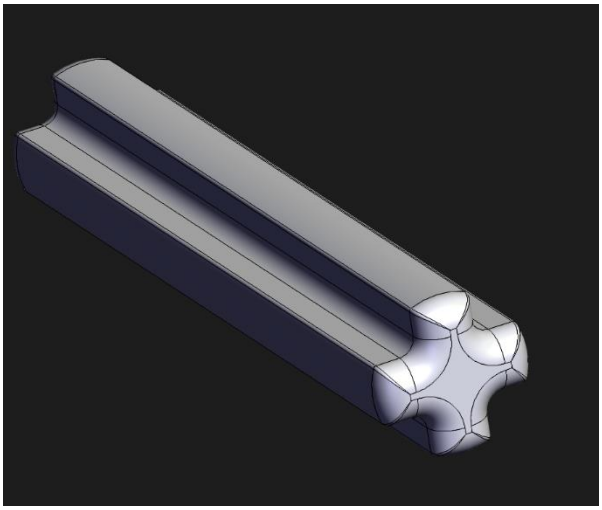
Nous avons donc un bloc en 3D qui est épais de 10 mm. Maintenant nous pourrons réaliser un deuxième sketch par-dessus pour réaliser des trous, ou bien au contraire rajouter de la matière :



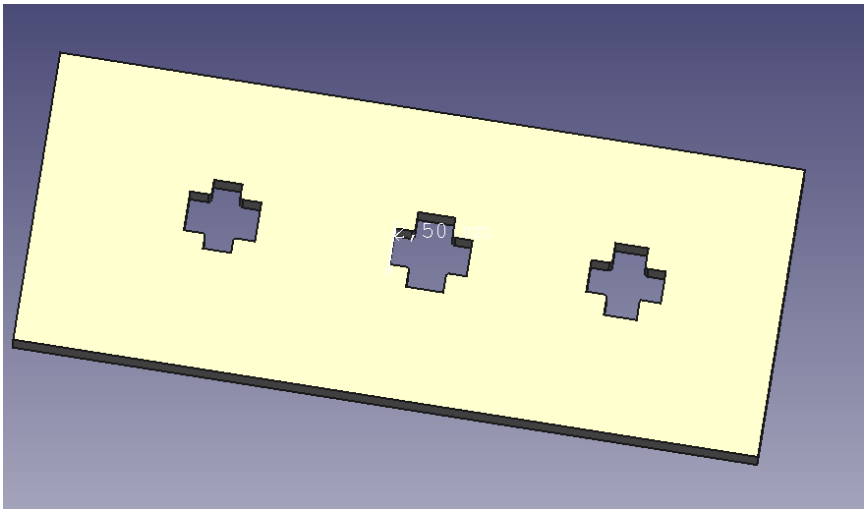
Nous pourrons grâce à cela par exemple créer les trous permettant de mettre les bâtons de chez lego qui permette de faire tenir le capteur sur un robot par exemple.

2. Réalisation d'une plaque d'essai :

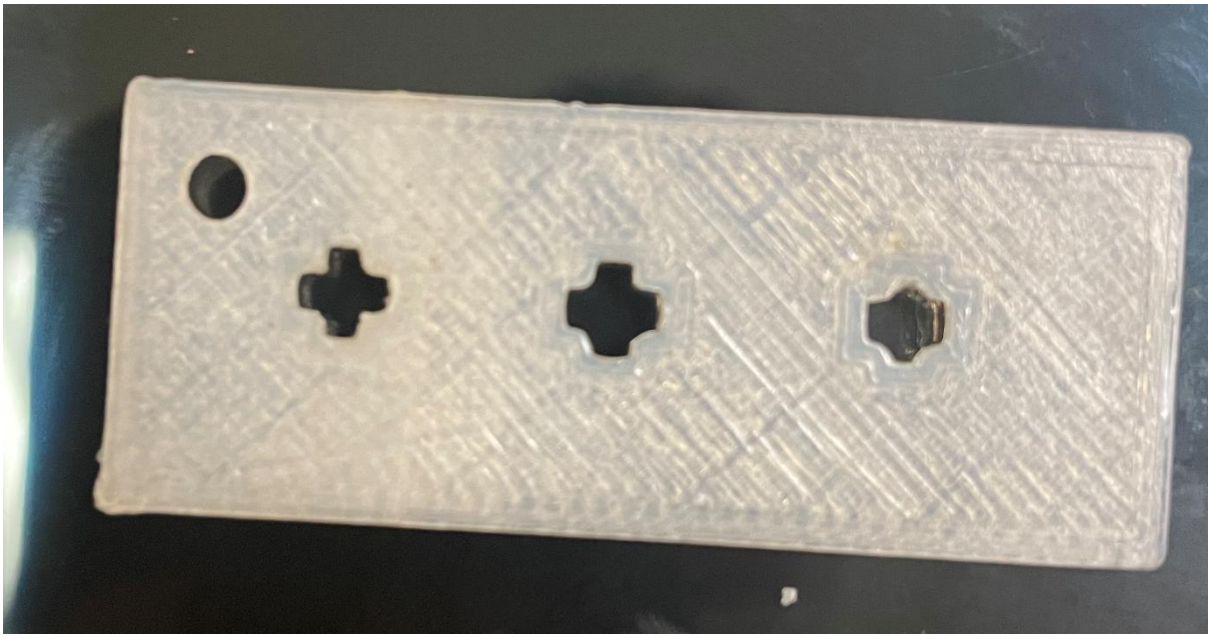
Nous devons réaliser une plaque d'essai avec plusieurs trous en forme d'étoile de différente taille, pour savoir lequel va être compatible avec l'axe en plastique de chez lego. Après quelque recherche j'ai trouvé un fichier 3D sur Grabcad.com avec l'axe voulu :



Nous pourrons voir les tailles qu'elle possède, de ces tailles il va falloir rajouter une marge car l'imprimante 3D de l'IUT est précise à 2 mm près. Nous avons réalisé une première plaque sur FreeCad qui n'était pas assez grande :

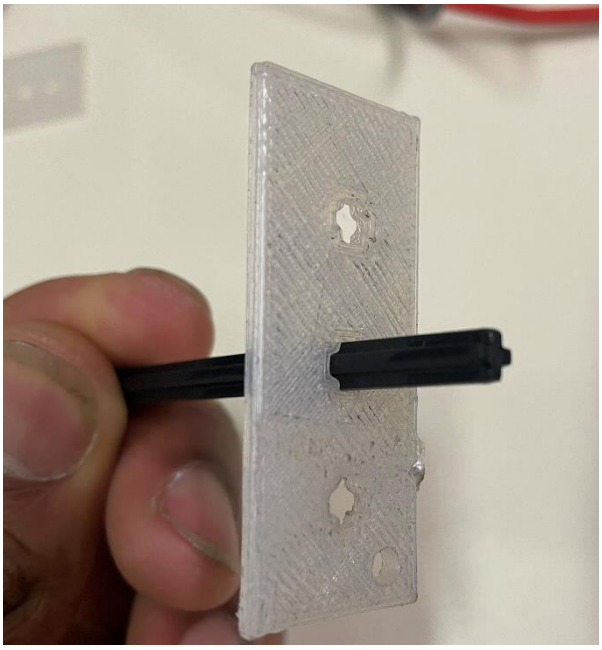


Les trous sont aux alentours de 2,5 mm sur un côté et sur l'autre plus petit côté 1,2 mm, cela n'a pas suffi :



L'axe n'est rentré dans aucun des trous réalisés, j'ai donc du réalisé un deuxième essai. Nous avons ajouté un trou à la plaque et nous avons mis 2mm environ en plus à chaque côté, cela nous donne cette plaque :

Après quelque essai fait sur chaque trou, il y en a 1 seul qui correspond à la bonne taille :

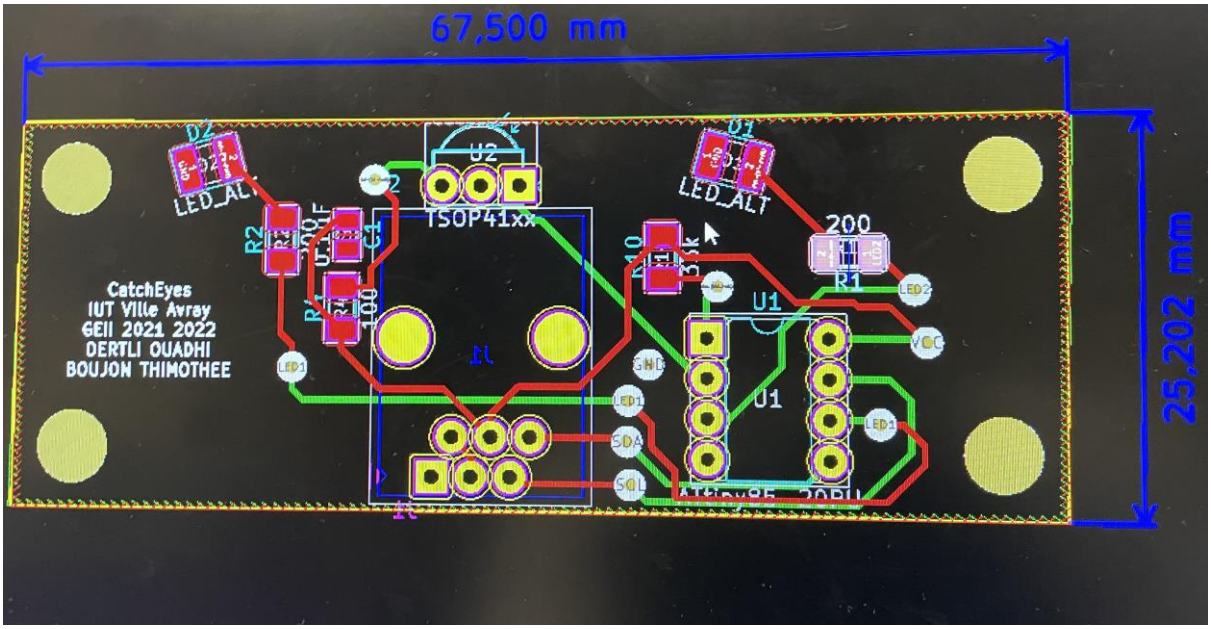


Nous avons donc ce qu'il faut pour l'axe et nous pouvons passer au carter.

3. Réalisation du carter :

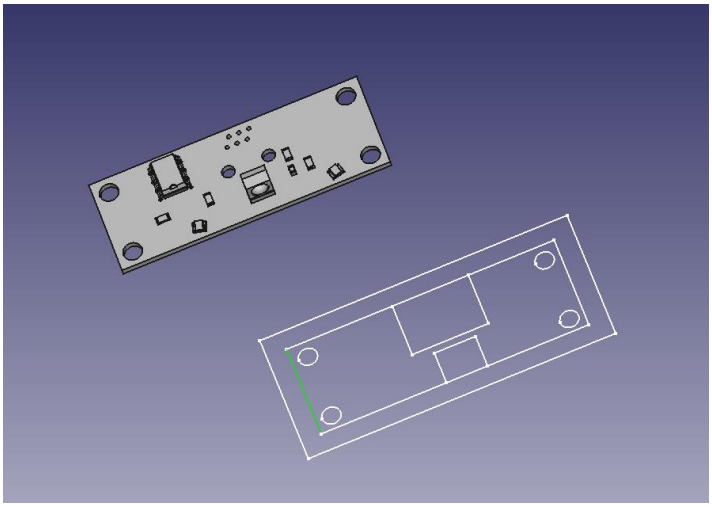
Osman s'occupant du KiCad m'a donc donné les dimensions afin que je puisse réaliser le carter, et j'ai importé ces fichiers sur Freecad afin de l'avoir en 3D et savoir ou placer mes trous.

Kicad du capteur final :



Je n'ai plus cas l'envoyé en fichier 3D sur Freecad en allant dans les paramètres de Kicad et l'important en fichier step. Cela me donne donc en 3D la carte finale que nous aurons, j'ai donc commencé à faire le sketch qui sera son carter.

4. Sketch et carte final en 3D :



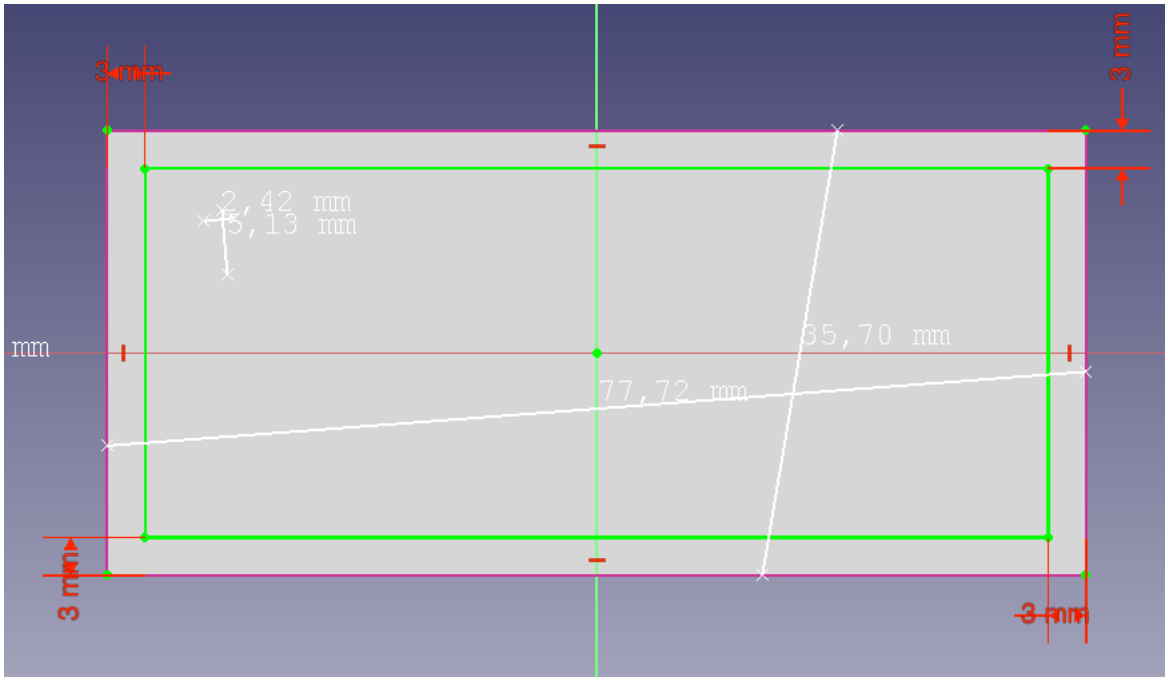
5. Création du carter en 3D :

Nous allons donc créer un carter qui répondra à ce cahier des charges :

- Une longueur minimum de 77,5 cm
- Une largeur minimum de 35,5 cm
- Des rebords d'au moins 3 millimètres afin de coller les deux faces du carter
- Des trous pour les axes lego de 2,4 mm en largeur et 5,13 en longueur
- Créer blocs permettant le blocage de la carter dans le carter
- Création d'arc de cercle permettant l'isolement des deux émetteurs infrarouges de la lumière non voulu
- Création d'un trou pour la connectique vers du capteur vers la brique lego
- Création d'un trou pour le récepteur.
- Plastique du capteur de couleur noir afin de limiter la venue de lumière non voulu

6. Création de la face-arrière du capteur lego :

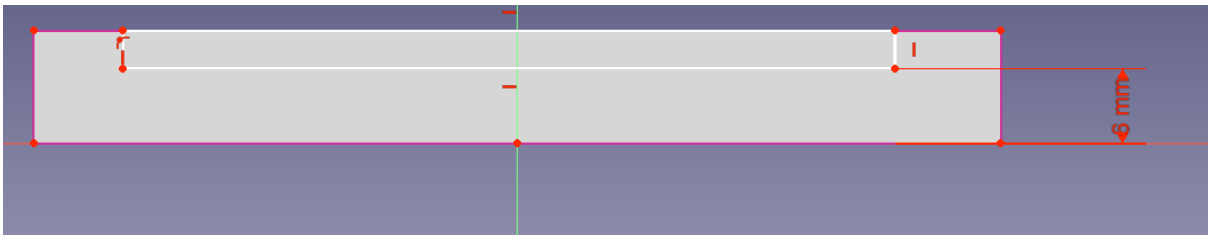
Nous création donc un bloc dans des dimensions respectant le cahier des charges et avec une épaisseur de 9mm, nous créons un sketch rectangulaire à l'intérieur de ce bloc avec une marge de 3mm avec le sketch précédent afin d'avoir notre rebord :



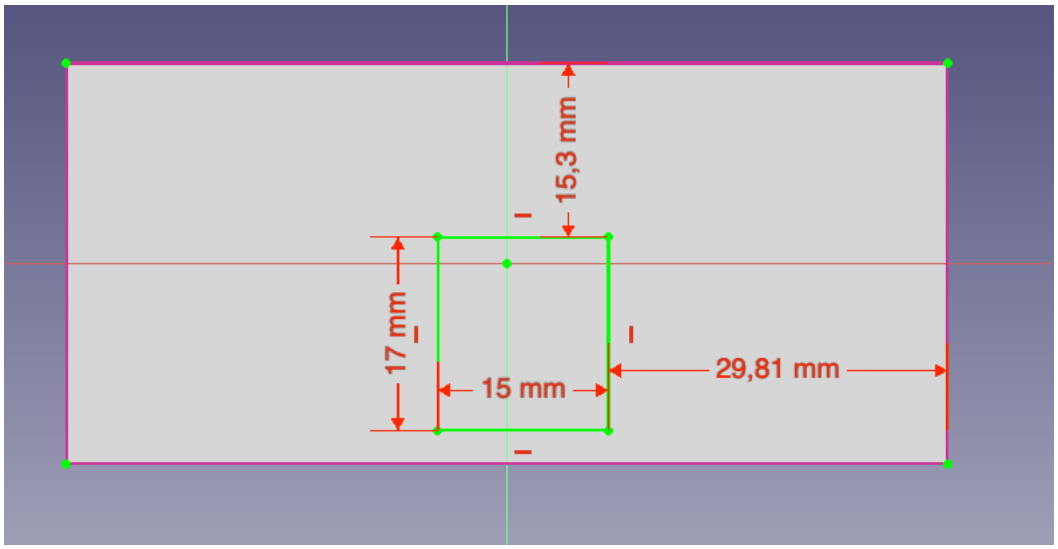
Nous ferons donc créer une cavité de 8mm afin d'avoir une surface assez fine (9mm-8mm = 1mm), afin de pouvoir bien faire ressortir la connectique :

Nous allons ensuite supprimer une partie du rebord car il nous faut un espace pour que les deux émetteurs puissent envoyer leurs signaux :

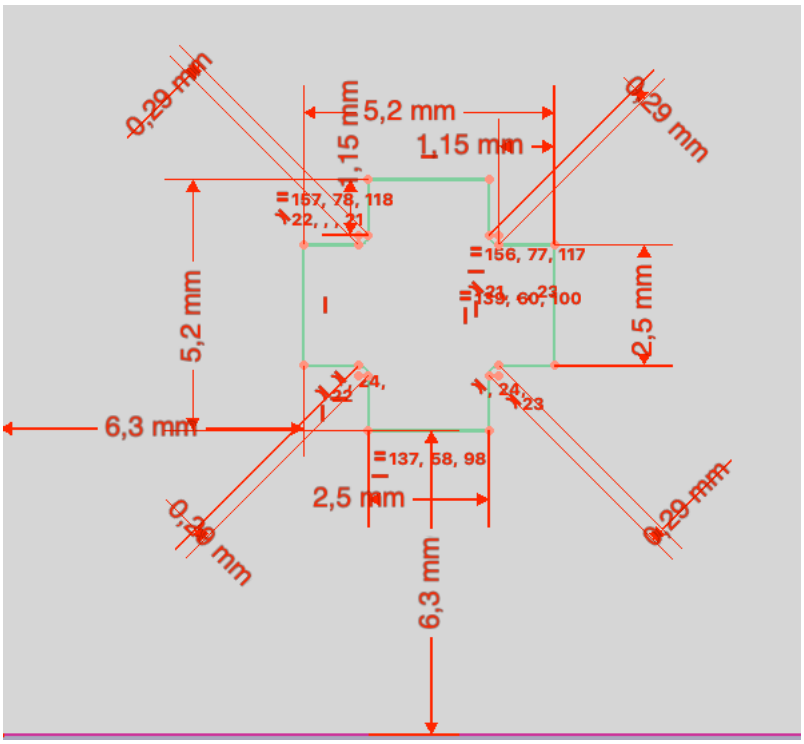
Nous avons un écart de 6 mm sur le sketch du bas du carter au début de l'espacement :



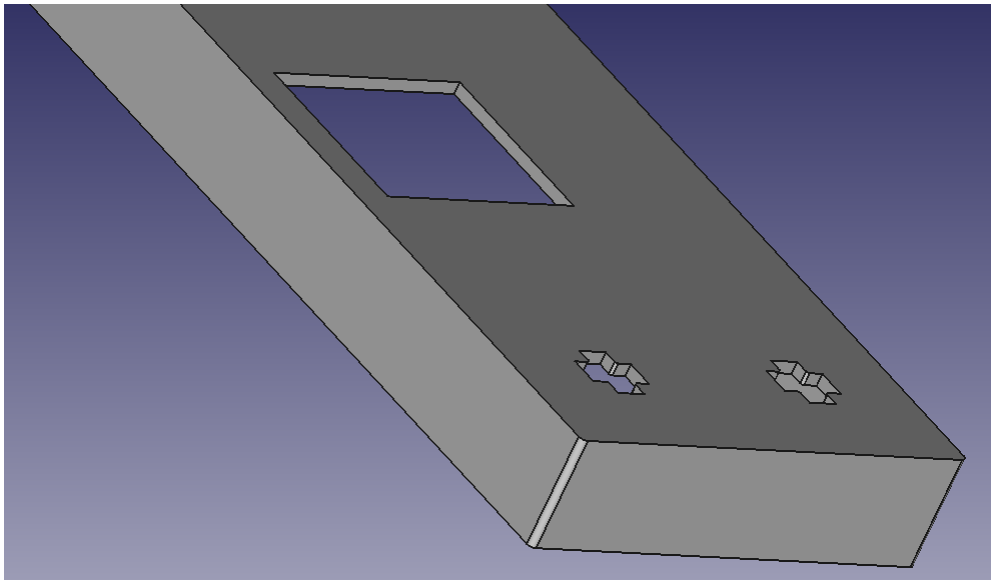
Maintenant nous voulons créer notre trou permettant la connectique, pour cela il m'a fallu mesurer la taille du composant qui ira à cet emplacement, de ce fait nous avons pris une taille de 17mm en longueur et 15mm en largeur, avec le Kicad fabriqué par Osman et que j'ai importé vers Freecad, j'ai pu voir en mettant le capteur dans ma création freecad l'emplacement du trou :



Nous mettons donc les trous créer précédemment sur les 4 extrémités de notre carter en regardant les emplacements des trous du capteur, les trous serons fait à 6,3 mm des côtés de notre carter :



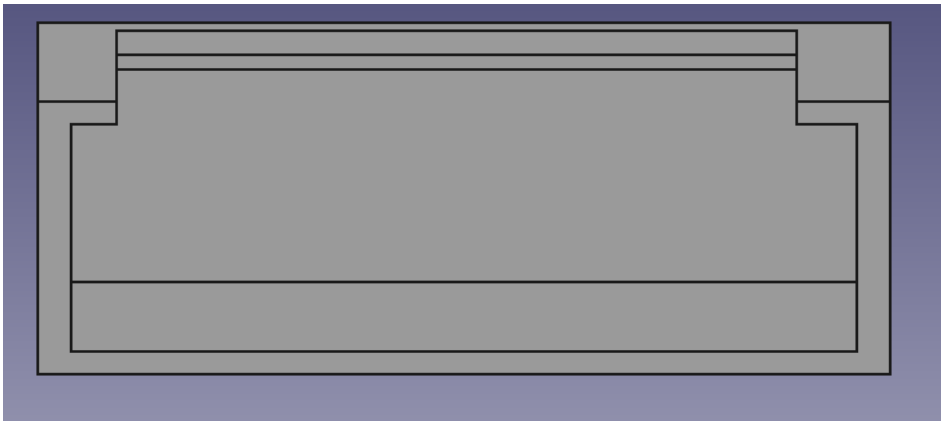
Pour finir nous avons arrondi les bords du carter sur les coté afin de lui donner un style plus épuré :



7. Création de la face-avant du capteur lego :

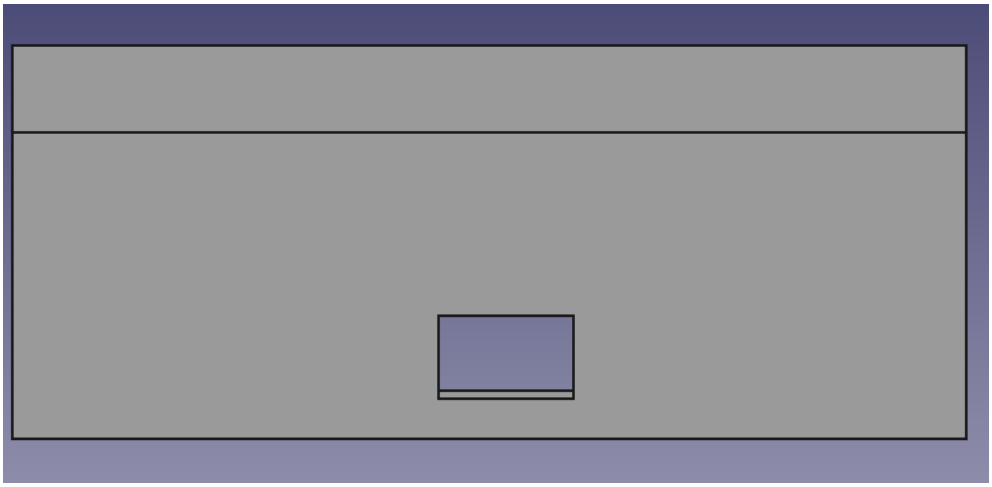
Cette face sera moins simple à réaliser car nous aurons un trou à faire pour le récepteur, ainsi que les arcs de cercle pour les émetteurs, les blocs de blocage pour le capteur et pour finir les trous pour les axes lego.

Nous allons donc créer un second bloc de la même taille que le précédent et nous allons refaire une cavité afin d'avoir une surface sur le dessus assez fine pour que le récepteur ressort bien :

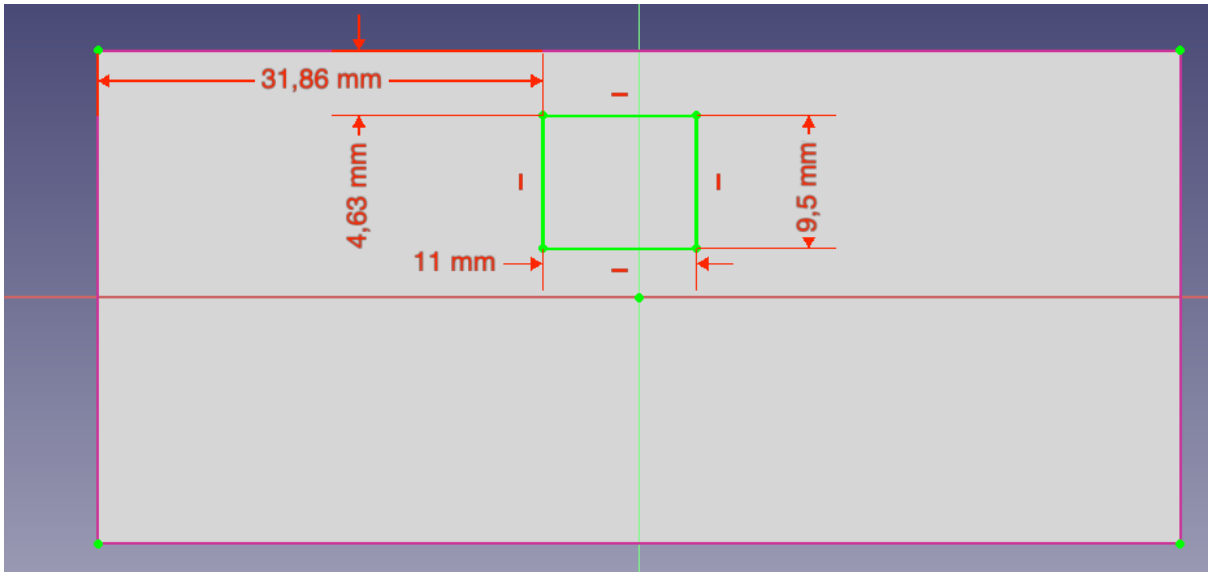


Nous avons un espace au niveau du rebord qui est plus conséquent que le précèdent car c'est ici que nous aurons notre arc de cercle.

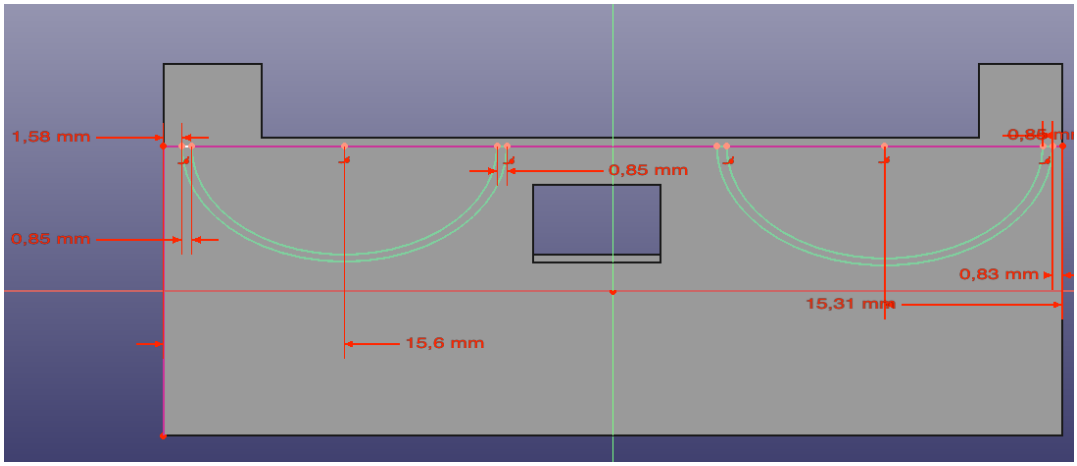
Nous avons par la suite créé le second trou pour le récepteur en ayant pris les dimensions du récepteur ainsi que son emplacement sur le Kicad :



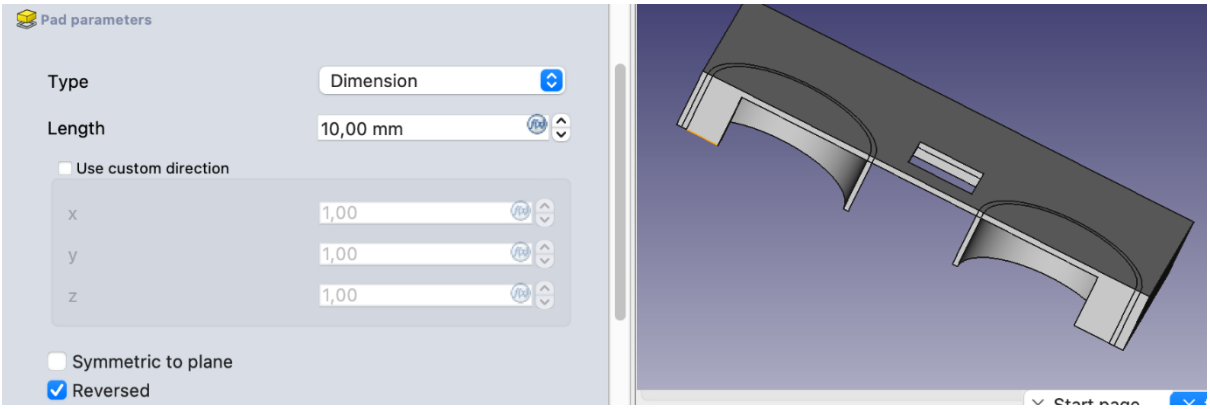
Nous aurons donc un trou rectangulaire de largeur 9,5mm et de longueur 11mm.



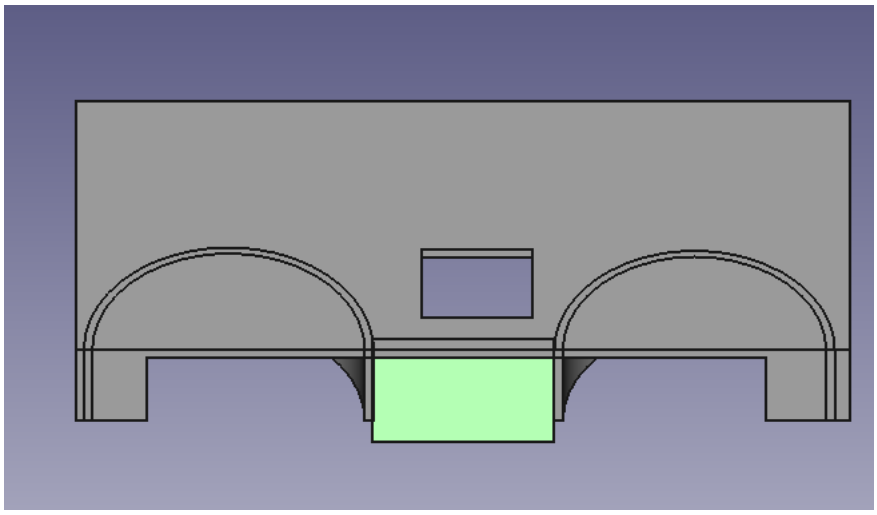
Afin de créer les arcs de cercle qui nous permettrons de diminuer l'entrée de perturbation lumineuse et de caché le reste de la carte, nous devrons créer de de demi-cercle en regardant le positionnement sur le Kicad, les mesures ne seront donc pas forcément les mêmes pour les deux demi-cercles afin de ne pas se poser sur des composant aux alentours des émetteurs.



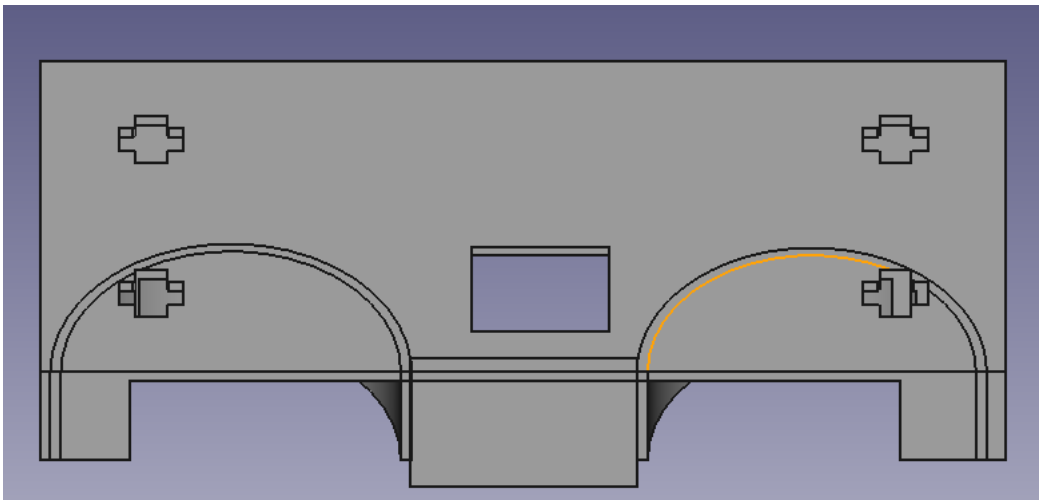
Nous effectuons une protrusion et nous devons l'inverser car sinon, les demi-cercles vont se créer sur la face extérieur et non intérieur comme voulu :



Nous allons rajouter un bloc qui va cacher le milieu afin de ne pas voir le reste de la carte :

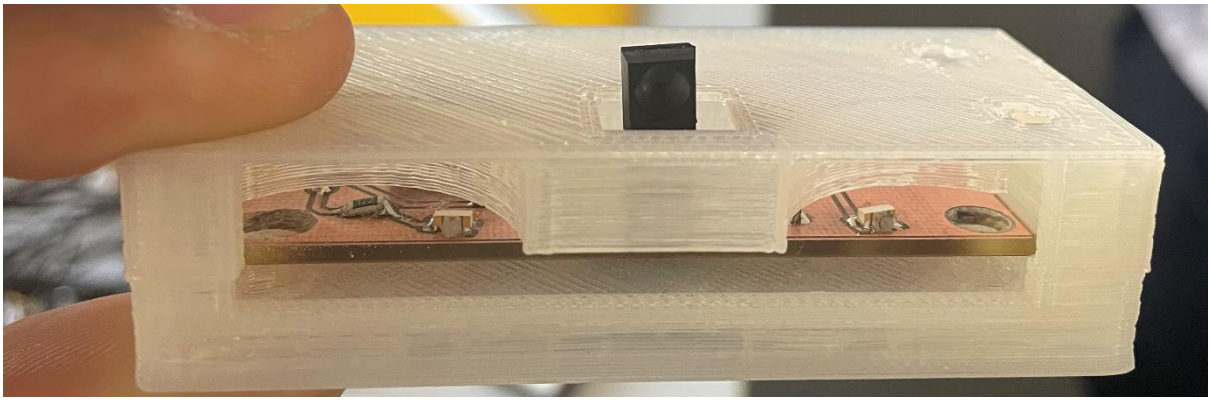


Nous rajoutons les trous pour les axes lego, mais avec une dimension inférieure de 2mm afin de s'assurer qu'il y aura un maintien au moment de passer l'axe dans les 3 trous, car la machine de fabrication 3D à une marge d'erreur non prévisible.



Et pour finir nous arrondissons les bords.

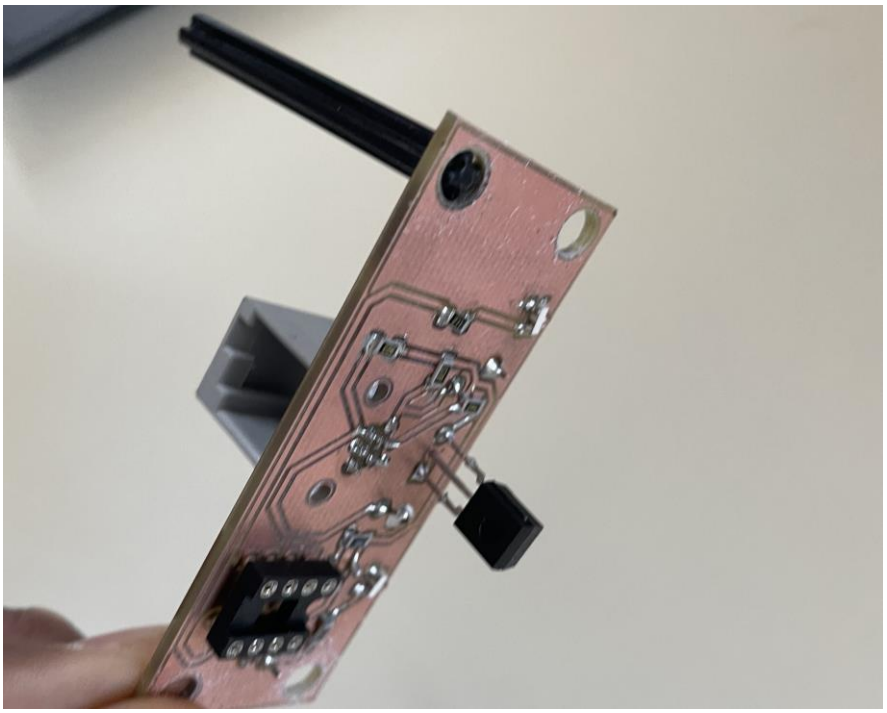
Nous avons donc après impression ce résultat :



Plusieurs problèmes se pose :

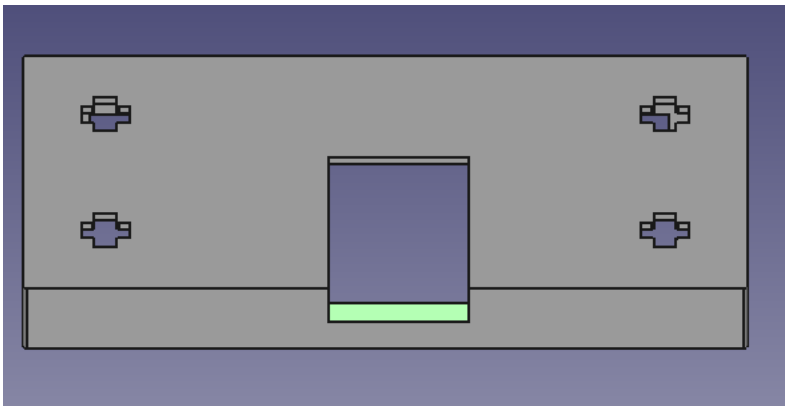
Les arcs de cercles ainsi que le bloc du milieu ne sont pas assez longs, pour cacher la face arrière et le trou de connections n'était pas à la bonne place dû à des modification du capteur qui n'était pas présent sur Freecad, le capteur n'avait pas non plus la couleur voulue.

Les axes lego ne rentrait pas dans le trou du capteur, pour cela j'ai dû percer avec une perceuse des trous légèrement plus gros :

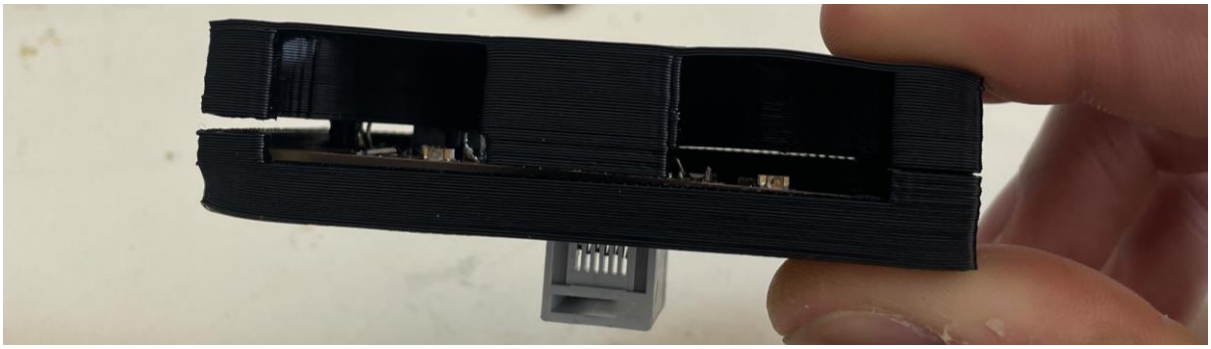


Nous avons utilisé une mèche de 4,9 mm de diamètre.

Nous avons ensuite rectifié les problèmes précédents et nous avons aussi ajouté un espace ouvert pour le branchement de la connectique sur la face arrière :

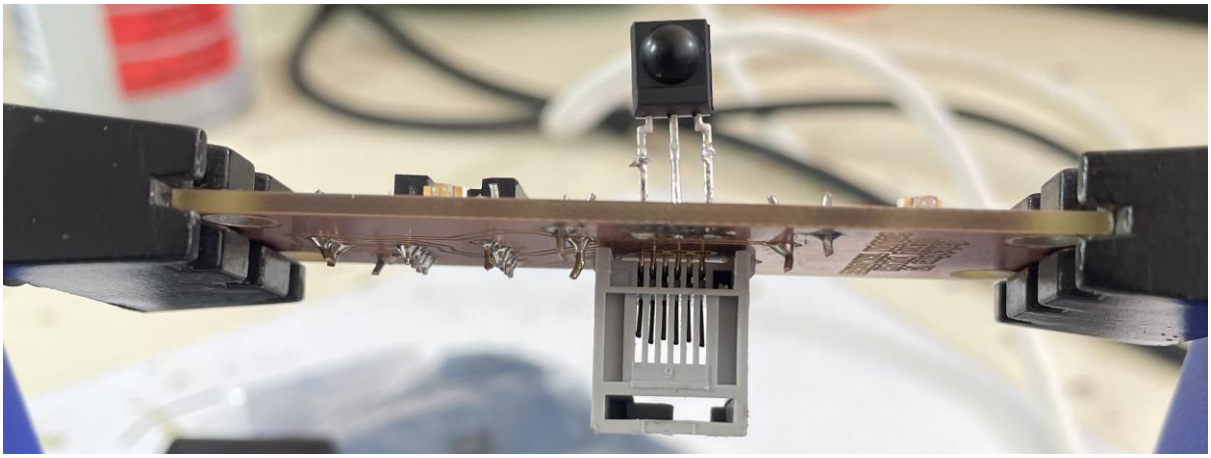


Nous avons donc notre version finale :



Malheureusement, d'autre problème se sont posé :

Dans un premier temps le capteur ne ressort pas, pour y remédier, j'ai dessoudé le capteur et je l'ai fait ressortir un peu plus de la carte :

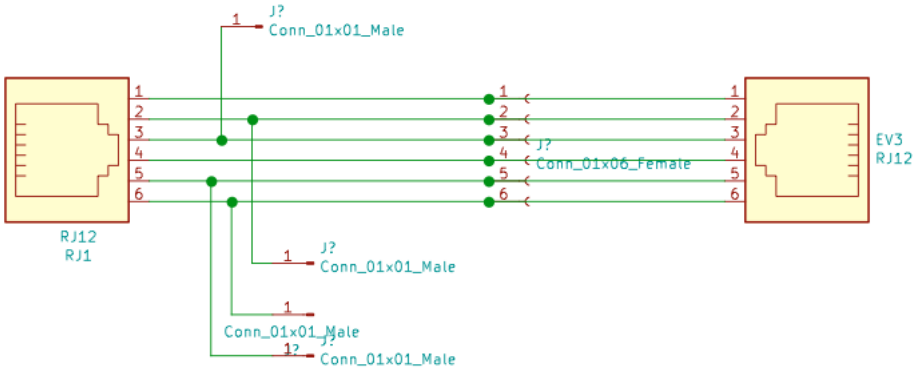


Dans un second temps les blocs de blocage était trop long et ou bien il aurait mieux fallu le mettre sur la face opposée, je l'ai donc limé :

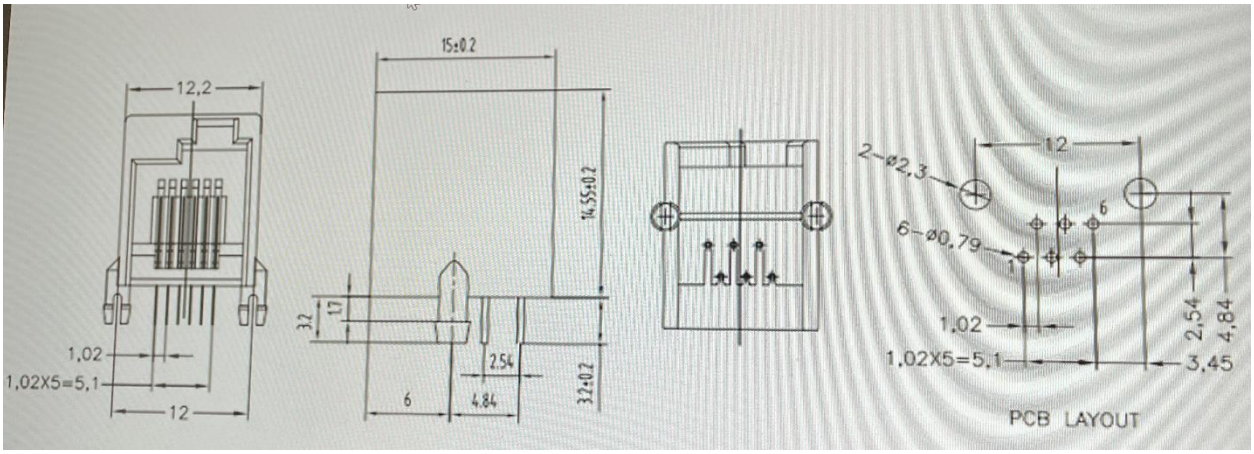
Notre carter est donc maintenant opérationnel.

8. Création d'une carte allant d'un connecteur RJ-12 à un connecteur EV3

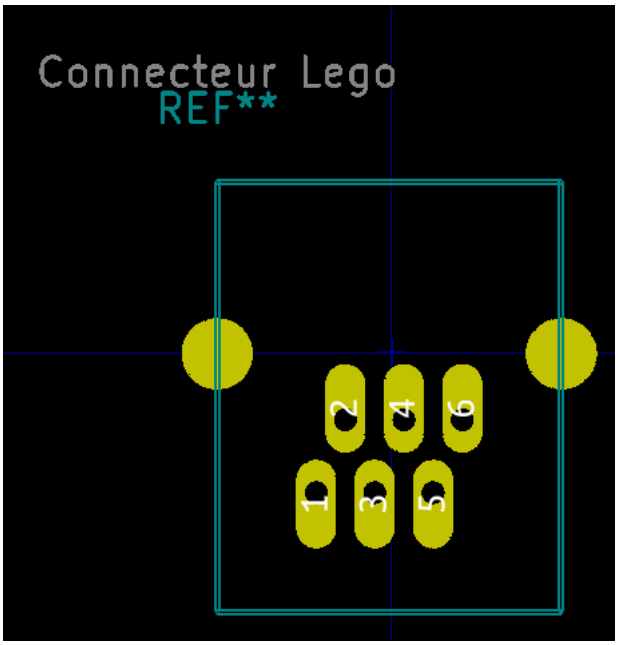
Nous allons créer une carte qui va connecter un connecteur RJ12 et celui des ev3, avec des branches sur les pistes afin de pouvoir faire des mesures :



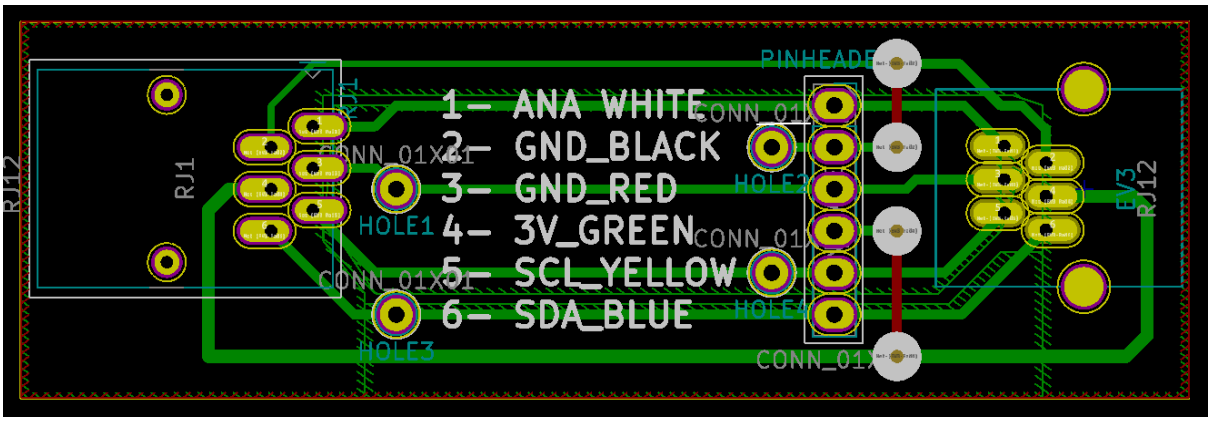
Nous avons pris l'empreinte faite par Osman du connecteur EV3, Malheureusement celle-ci à un défaut, la rangée de pins du bas est inversé avec celle du haut, donc pour palier à cela nous avons dû modifier l'empreinte en s'aidant de la documentation technique suivante :



Après avoir modifié cela on obtient notre empreinte :

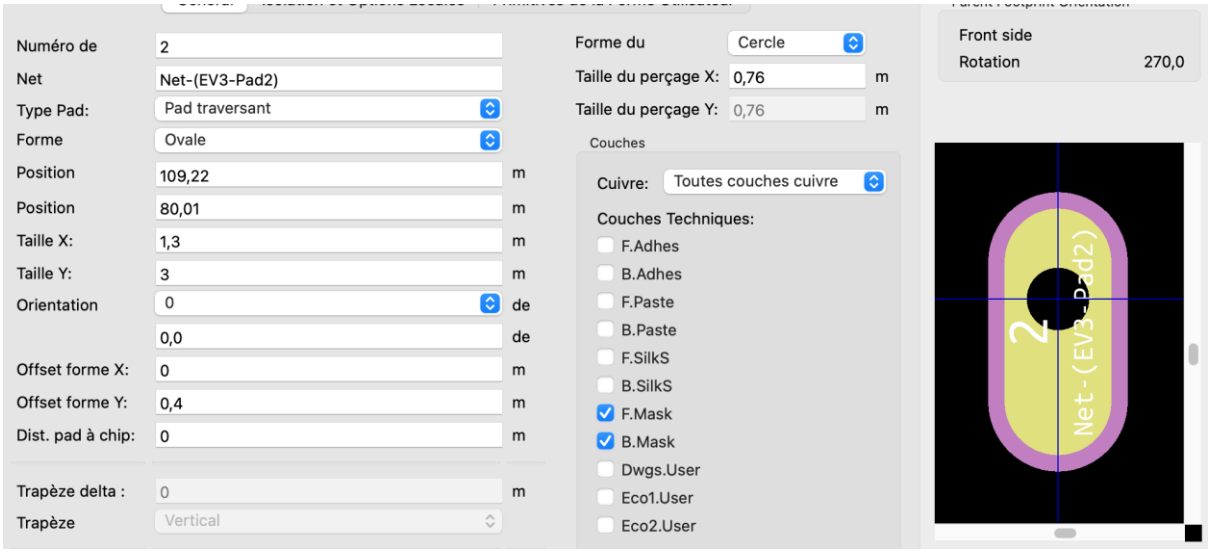


Nous pouvons donc procéder au routage de la carte :



Nous avons mis au milieu de la carte un texte en gravure permettant de se repérer, il y a 4 pins, 1 sur chaque GND et 2 autres sur le SCL et SDA afin de prendre des mesures, nous écartons les pins entre elles afin de ne pas avoir de problème de diaphonie. Les pastilles sont trop proches l'une de l'autre sur le composant EV3, de ce fait nous devons mettre deux via afin de pouvoir faire passer les pistes.

Pour le composant RJ12 et EV3, nous mettons des pads de taille X 1,3mm et de taille Y 3mm afin qu'il n'y ai pas de conflit entre eux et nous leur mettons un offset de plus ou moins 0,4 selon sa rangée :



La carte fera 71mm de longueur pour 22,8mm de largeur. Nous créons pour finir le plan de masse. Afin de l'envoyer au technicien, nous créons les fichiers gerber.

Mise en place de la maquette de test du système

Charles Ouadhi

Afin de câbler les différents éléments électriques sur la plaque, nous allons utiliser les parties précédentes pour réaliser cette plaque. Il y a différentes choses à savoir. La première est le fonctionnement du programme effectuée par Yohan. La seule chose qu'il faut retenir dans cette partie est que l'ATTiny85 enverra deux signaux carrés avec une fréquence de 30kHz sur ses deux ports : PB1 et PB4. Il faut savoir que ces deux signaux ne peuvent pas être actifs en même temps. Quand l'un sera envoyé l'autre sera nul et inversement. Et donc une led sera allumé quand l'autre sera éteinte.

Pourquoi avoir fait ceci ?

La raison est simple, ayant un récepteur celui-ci serait incapable de gérer et de renvoyer une tension en sortie en fonction des deux signaux simplement. Nous cherchons la facilité. De ce fait le récepteur pourra se focaliser sur un signal à la fois et renvoyé les données de celui-ci. (Un signal carré si tout va bien et un signal continu si il y a un obstacle. La valeur moyenne du signal carré est toujours inférieur à la valeur du signal continu) De ce fait Yohan pourra (pas encore mis en place) sauvegarder cette valeur et en comparant celle-ci avec la seconde valeur nous pourrons indiquer où est l'obstacle.

Exemple :

La led droite est contrôlée par le signal 1 venant de la pin PB1 et le second signal lui contrôle la led gauche. Pour rappel, si le signal est continu, sa tension sera de 3.8V et la moyenne du signal carré est de 1.9V

Réception 1	Carré	Carré	Continue	Continue
Réception 2	Carré	Continue	Carré	Continue
1<2	Non	Non	Oui	Non
2<1	Non	Oui	Non	Non
Moy1=Moy2	Oui	Non	Non	Oui
Moy1=Moy2=3.8V	Non	Non	Non	Oui
Résultat	Il n'y a aucun obstacle.	Il y a un obstacle situé à gauche	Il y a un obstacle situé à droite	Il y a un obstacle situé devant.

Maintenant parlons de la réalisation de la plaque et plus de son fonctionnement.

Je me suis basé sur le kicad de Osman afin d'effectuer cette partie. Voici une photo du premier montage. Nous pouvons observer les différentes parties de celui-ci. Vous pourrez constater une filtre passe RC qui était indiqué sur la doc si nous utilisions une alimentation peu stable et qui monte dans de plus haute tension. Ceci a était mis par erreur et fût rectifiés par la suite mais je n'ai pas de photo du nouveau montage.

Si nous oublions ce problème, le système fonctionne correctement. La partie la plus dure ici fût de comprendre comment le récepteur était censé marché. Effectivement le filtre RC mit devant le récepteur qui avait pour but de protéger celui-ci l'a tout simplement rendu inactif. C'est lors de mon second montage en faisant des tests à chaque étape que j'ai découvert que la mise en place de ce filtre désactivé notre récepteur. Après l'avoir enlevé, j'ai dû faire des tests car la fiche d'information du récepteur était incomplète et nous voyons des signaux carrés, continu sans comprendre pourquoi.

Par suite de ces test, j'ai pu en déduire comment marche le récepteur, les informations ont été mise plus haut. Le plus long et dur dans cette partie fût de comprendre pourquoi le premier montage ne fonctionné pas et ensuite comprendre le fonctionnement du récepteur. Le bilan de cette partie est positif, le système marche entièrement.

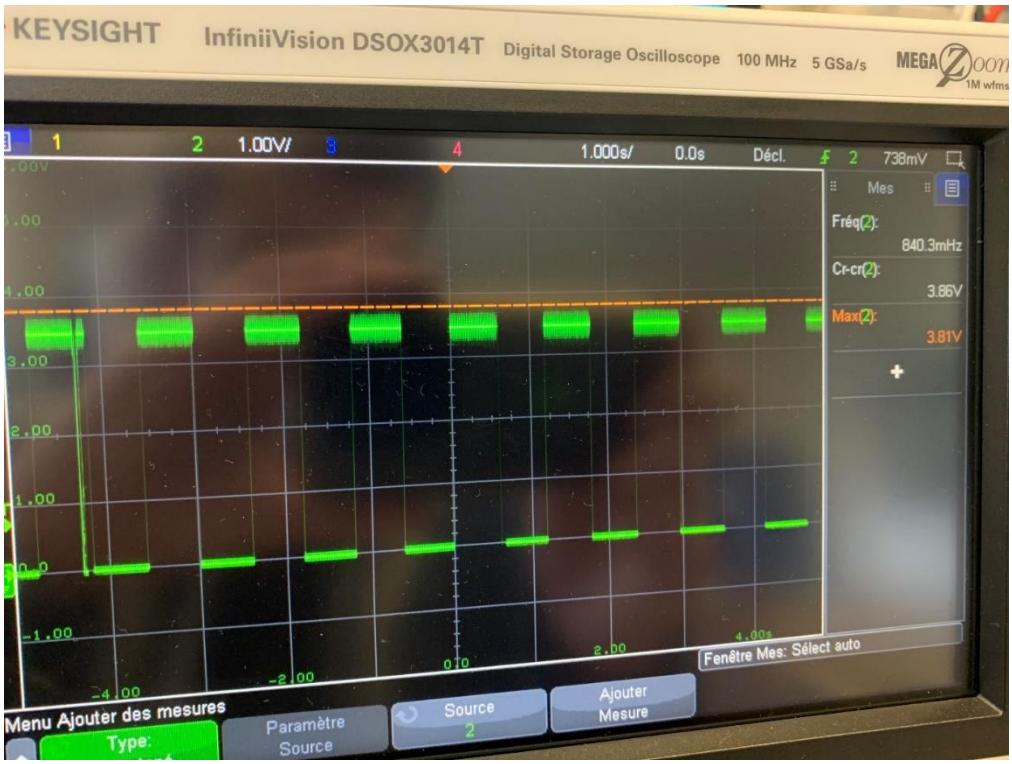
La seconde partie fût d'effectuer les tests sur la carte Kicad produite par Osman. Nous avons eu un problème lors de cette partie. Avant d'alimenter la carte, j'ai pris l'ohmmètre et j'ai regardé si toutes les pistes étaient raccordées comme il le fallait et ce fût le cas.

Après avoir vu que les signaux devraient passer là où il faut j'ai décidé d'allumer l'alimentation et de prendre des captures des signaux à l'oscilloscope. Lors du premier démarrage, il y avait une odeur de brulé donc j'arrête l'alimentation et je cherche d'où vient la fumée. N'ayant pas trouvé je décide de retenter a l'ohmetre si il n'y aurait pas de problème.Et je n'en trouve pas. Je rallume une seconde fois et j'ai eu la mauvaise idée de me dire « je vais chercher ou est le problème au doigt ». Je me suis brulé le doigt mais maintenant nous savions que la résistance de la led située à gauche est en plein court-circuit ce qu'il l'a faite brulée. La carte ne marche pas du côté gauche mais nous savons pourquoi et nous avons déjà effectué les changements sur kicad pour éviter ce genre d'incident.

J'ai aussi commencé RobotC. Actuellement je ne l'ai pas avancé énormément mais je suis capable d'afficher des données et de par exemple faire tourner un composant relié à la brique légo que celui-ci soit considéré comme un moteur ou bien un capteur. Ayant finit ma première partie, je pense que je vais effectuer les tests sur la nouvelle carte et ensuite je m'occuperai de la bibliothèque RoboC.

Pour être franc, je pense que nous commençons à toucher à la fin du projet et la seule partie ou nous n'avons pas beaucoup avancée est la mise en place du cache de protection. Mais Alexis est rentrée et tout devrait bien se remettre dans l'ordre.

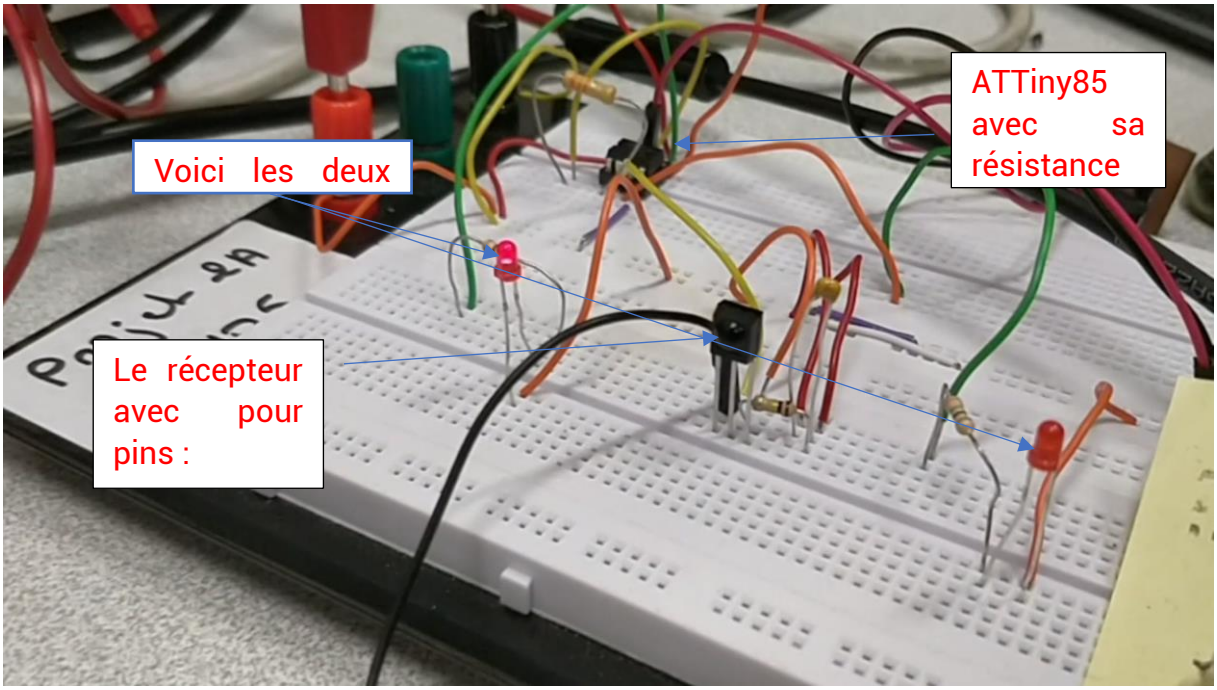
Photo du signal carré émis par le récepteur quand il n'y a pas d'obstacle.



Vidéo du test des leds (Dans le dossier).

(Le bruit négatif que vous pouvez entendre n'est pas dû au système qui ne marche pas mais à mon appareil photo qui ne capte pas la lumière émise).

Photo du premier montage avec le filtre RC.



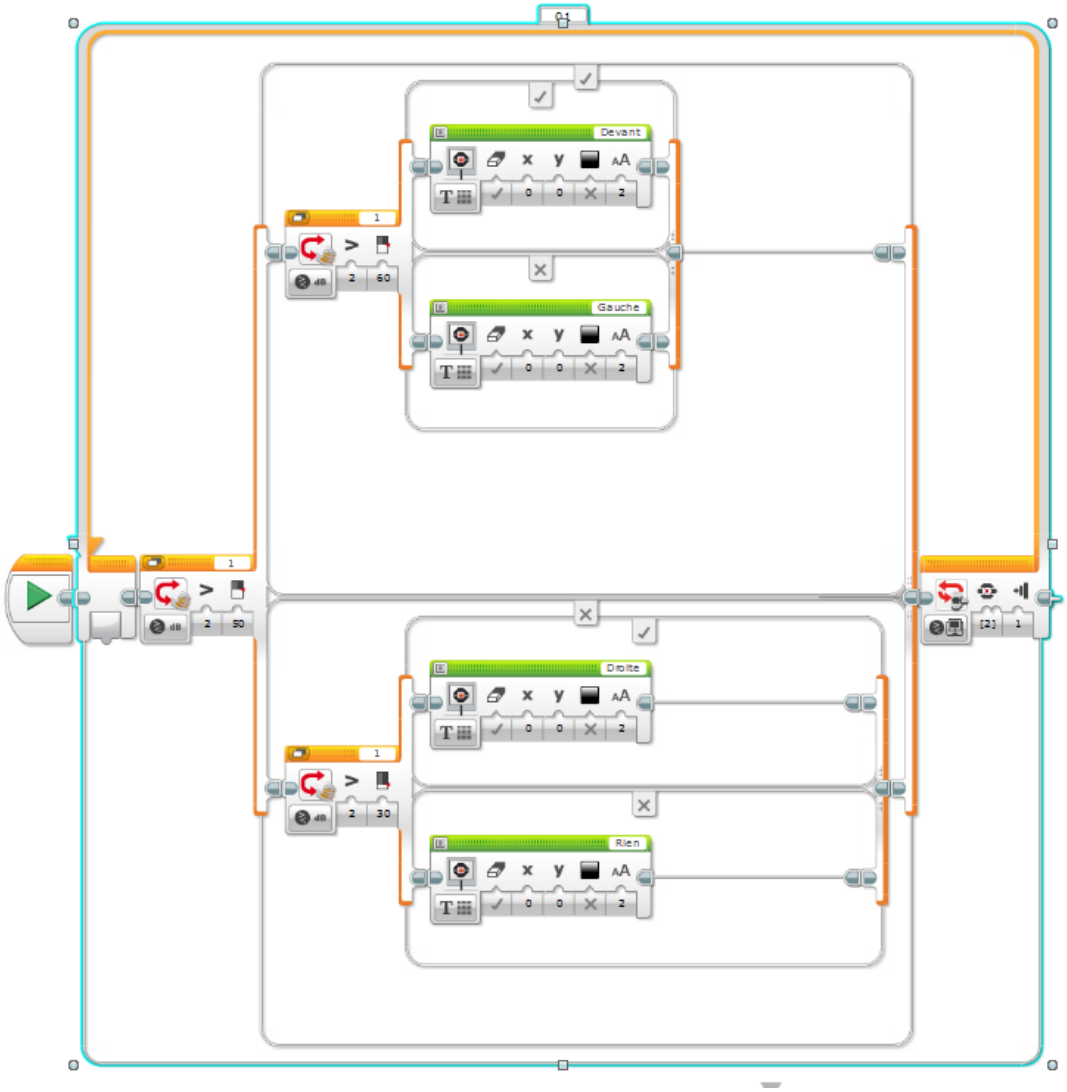
Analyse et Programmation I2C

Yohan Boujon

1. Analyse du capteur SumoEyes

La mission qui m'a été donné durant ce projet est de faire tout le côté microcontrôleur et communication entre la brique EV3 et le capteur. Pour cela j'ai décidé de connecter le capteur SumoEyes et de découvrir comment il fonctionnait. D'après la datasheet il est censé réagir quand il détecte un obstacle, à gauche, à droite ou bien devant.

Avec le logiciel donné par l'IUT, Lego Mindstorm EV3 Home. J'ai branché le capteur à la brique pour observer les informations que nous donnait le capteur.



Comme nous pouvons le voir, la brique reçoit une tension en « dB », comme le capteur n'est pas fait par Lego directement, il faut installer des blocks EV3 réalisés par Mindsensor. Nous nous retrouvons avec les valeurs suivantes qui peuvent être directement lues sur l'application EV3 :

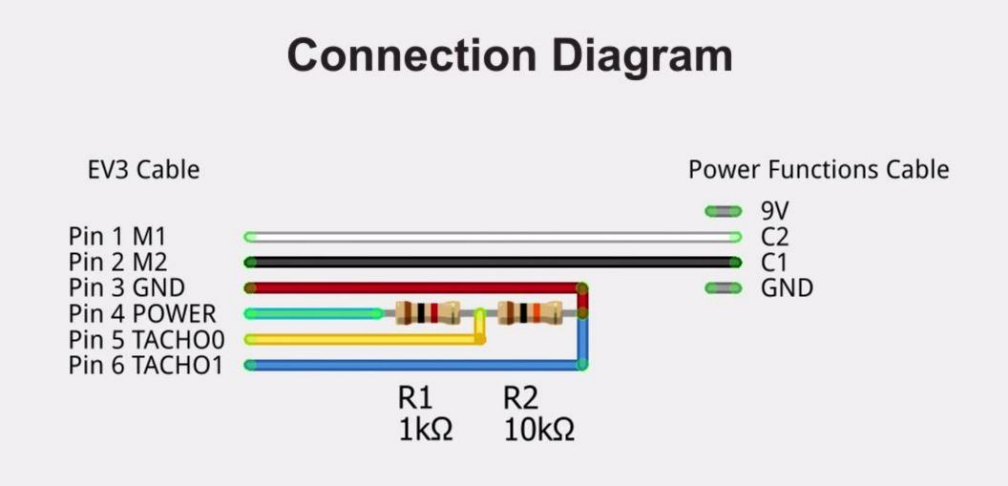
- Obstacle à droite = 35-36 (dB)
- Obstacle à gauche = 58-59 (dB)
- Obstacle devant = 66-65 (dB)
- Rien = 0-0.1 (dB)

Ces valeurs ont été déterminés en bougeant le capteur vers différents obstacles, nous pouvons récupérer ces valeurs analogiques et réaliser un petit programme permettant d'afficher où se situe l'obstacle en fonction de ceci.

2. Comment observer la communication

Le problème avec ce capteur, c'est qu'il faut envoyer un certain octet pour activer le mode I2C. Et comme nous ne savons pas encore comment faire nous avons d'abord regarder avec le capteur de l'autre groupe « Compass » et celui de l'ancien projet « Capteur de ligne ».

Pour cela, avec l'aide de mon camarade Charles nous avons modifié un câble, utilisé par le groupe de l'année dernière car certains pins étaient inversés. Nous avons remarqué avec une datasheet en ligne qu'il y avait un GND pour le capteur et l'autre pour la brique, sur l'ancienne version les deux GND ont été inversés et donc la polarité des tensions aussi, soit :



Source : Bricks Stack Exchange

Nous pouvons voir ici que le GND rouge est le vrai GND du capteur, et donc nous avons dû changer comment fonctionnait le câble.

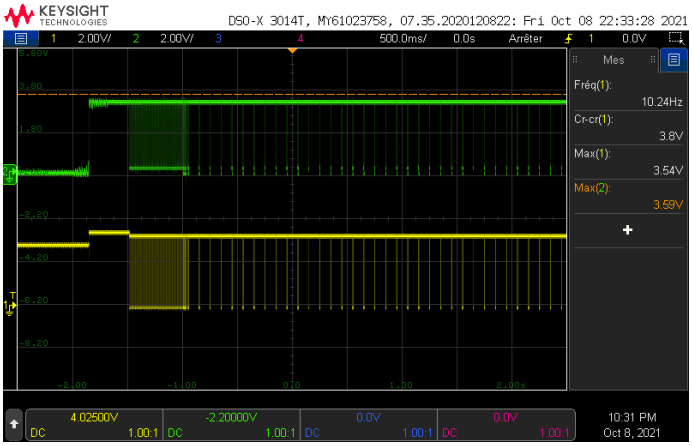
Après l'intervention de mon camarade, il a fallu connecter le SDA et le SCL à l'oscilloscope pour pourvoir observer les trames.

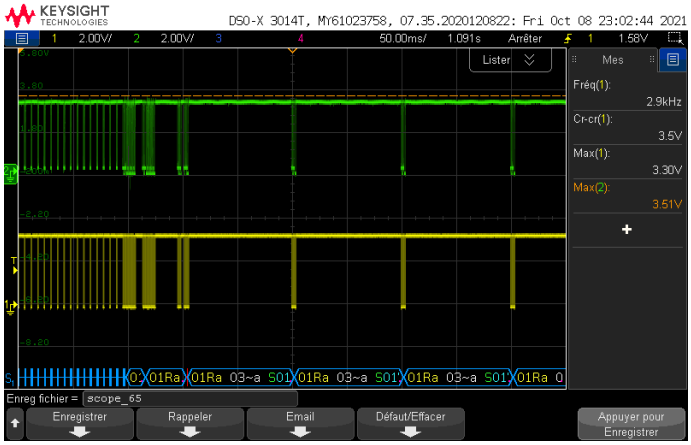
3. Analyse des trames

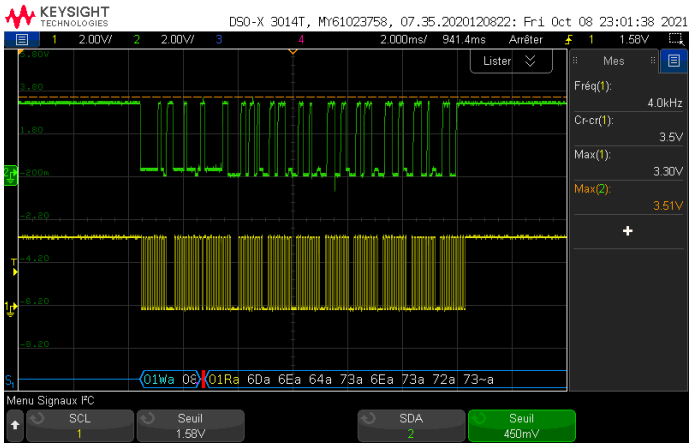
Le capteur SumoEyes étant en analogique il nous était impossible d'observer ces trames, nous avons donc regardé pour le capteur LineLdr ainsi que la boussole.

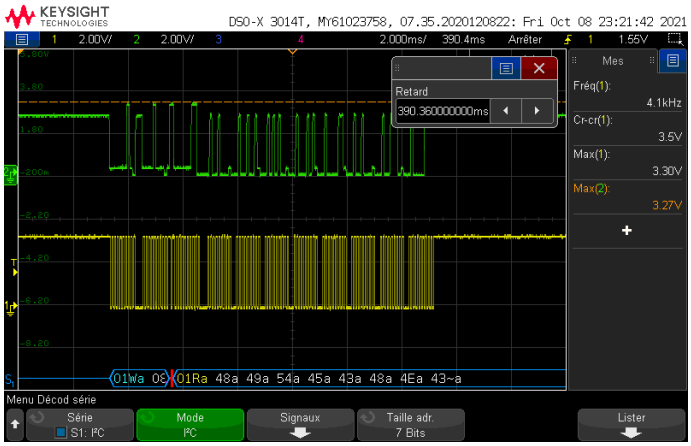
Pour analyser les trames il faut mettre 2V/décade de manière à observer les deux signaux. Dans le mode sérial j'ai mis le mode I2C et les seuils des signaux SDA (données) et SCL (horloge) à leur milieu pour que l'oscilloscope observe correctement les changements de tension.

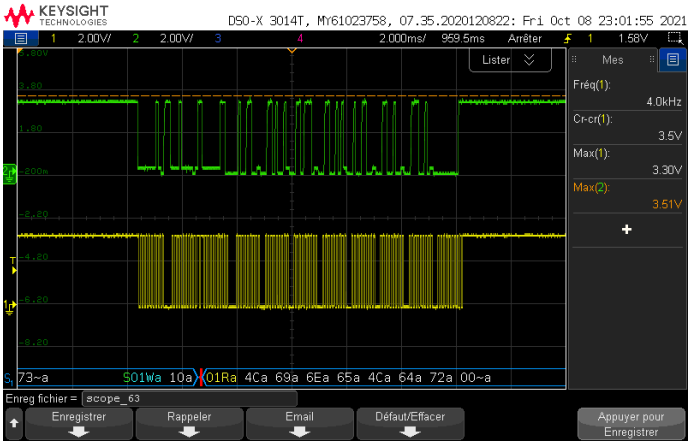
J'ai réglé un retard et mis en mode single l'oscilloscope. De cette manière, quand on branche le capteur et la brique nous recevrons plusieurs bits que nous pourrons analyser par la suite, comme montré ci-dessous :

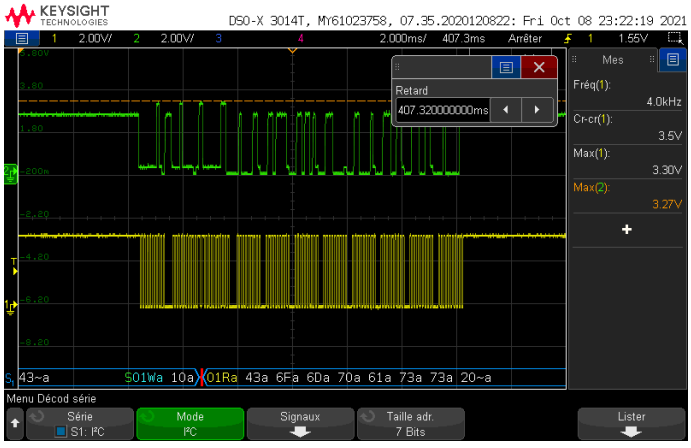


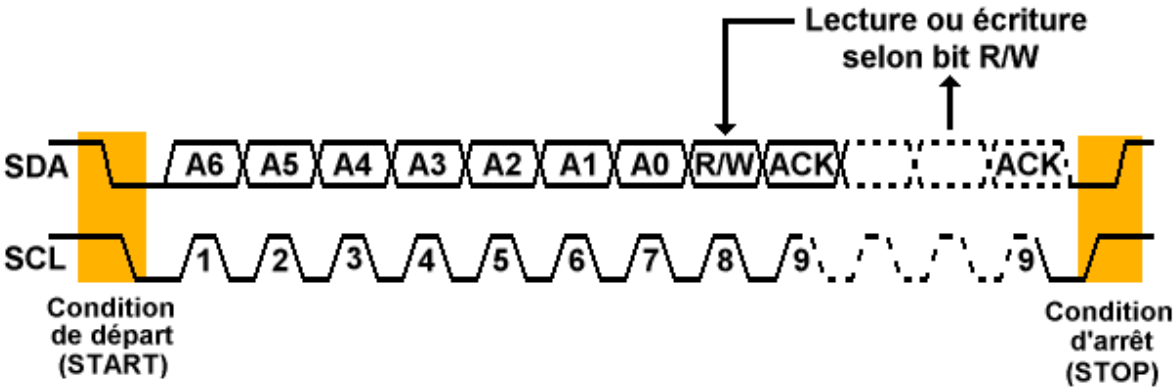


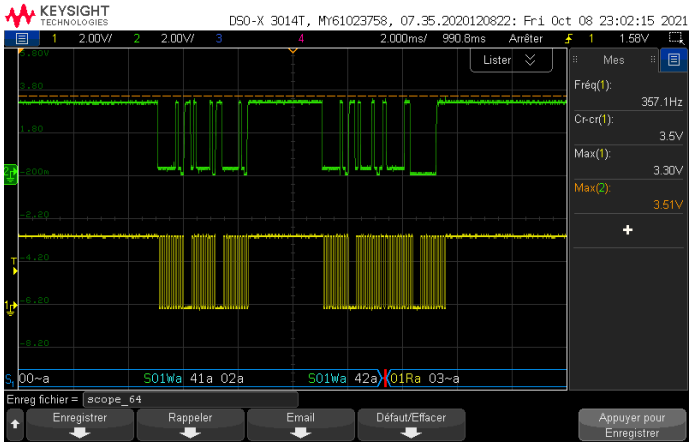


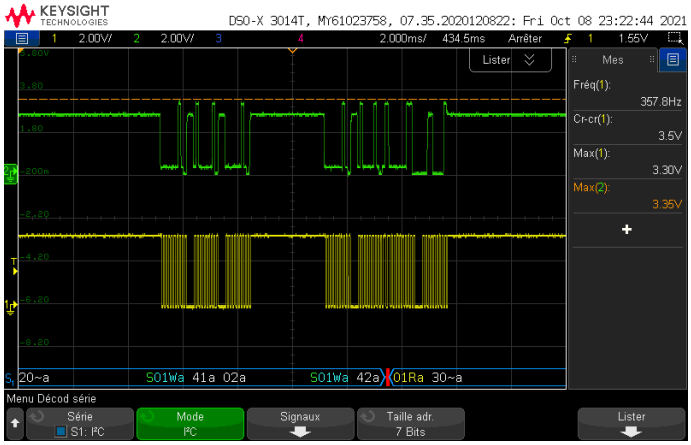












On peut voir que :

Pour le capteur ligne :

0x08 -> 6D 6E 64 73 6E 73 72 73 -> mndsnsrs (nom du constructeur)

0x10 -> 4C 69 6E 65 4C 64 72 00 -> LineLdr (nom du capteur)

0x41 0x02 (mode configuration de la Brique)

0x42 -> 0x03 (donnée du capteur)

Pour le capteur « Compass »

0x08 -> 48 49 54 45 43 48 4E 43 -> HITECHNC (nom du constructeur)

0x10 -> 43 6F 6D 70 61 73 73 20 -> Compass (nom du capteur)

0x41 0x02 (mode configuration de la Brique)

0x42 -> 0x30 (donnée du capteur)

Et avec cela nous pouvons voir dans les deux datasheets que nous avons exactement les mêmes bits d'adressage, nous pouvons donc commencer la programmation :

S'il est à 0, le maître signale qu'il va envoyer des octets, et donc que l'esclave doit les lire,

S'il est à 1, le maître indique qu'il veut recevoir des octets, et donc que l'esclave doit les fournir.

Address	Type	Contents
00 – 07H	chars	Sensor version number
08 – 0FH	chars	Manufacturer
10 – 17H	chars	Sensor type
18 – 3DH	bytes	Not used
3E, 3FH	chars	Reserved
40H	byte	Not used
41H	byte	Mode control
42H	byte	Heading } two degree heading
43H	byte	Heading } one degree adder
44, 45H	word	Heading (low byte, high byte)
46 – 7FH	bytes	Not used

0x00-0x07	Firmware version - *Vxxxx*	-
0x08-0x0f	Vendor Id - *mndsnsrs*	-
0x10-0x17	Device ID - LineLdr	-
0x41	-	Command
Register	Read	Write
0x49 - 0x50	Calibrated Sensor reading - Reading measured for each of the sensor in the array. Higher value indicates brighter object. (one byte for each sensor - 8 bytes)	-
0x51 - 0x58	White Reading Limit - Calibration value for White color for each sensor. (one byte for each sensor - 8 bytes)	-

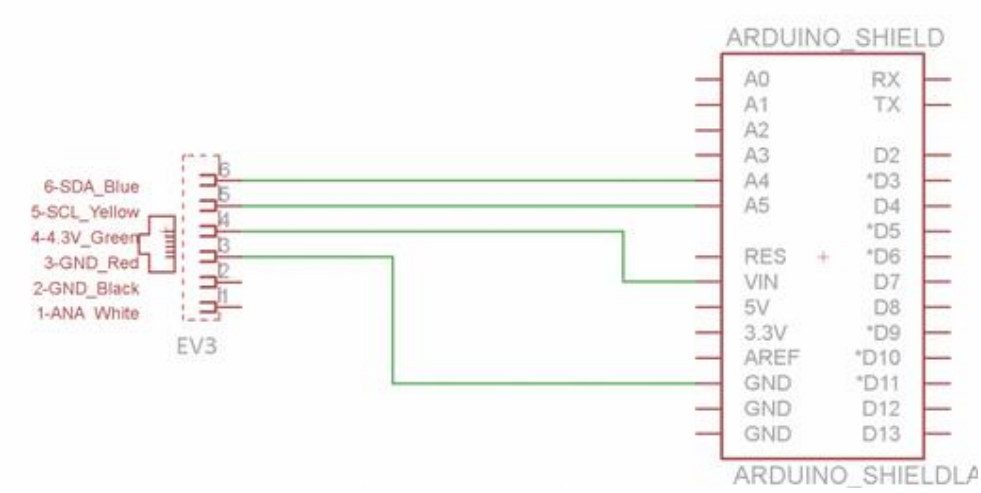
https://www.robotshop.com/media/files/pdf/introduction-nmc1034.pdf

http://www.mindsensors.com/pdfs/LineLeader-v2-User-Guide.pdf

4. Test avec Arduino

Avant de pouvoir programmer sur l'ATMEGA 16, je voulais être sûr de comment fonctionnait l'I2C avec la Brique, pour cela, j'ai programmé sur la Brique un programme qui récupère la donnée I2C et l'affiche sur l'écran.

Après avoir fait cela, à l'aide du site web www.dexterindustries.com, j'ai pu connecter correctement l'Arduino ainsi que la brique pour pouvoir envoyer un bit, voici le câblage :



Et voici le programme avec Arduino IDE :

```
#include <Wire.h>
#define SLAVE_ADDRESS 0x01

void setup()
{
    Serial.begin(9600); // start serial for output
    Wire.begin(SLAVE_ADDRESS);
    Wire.onReceive(receiveData);
    Wire.onRequest(sendData);
    Serial.println("Ready!");
}
int val,flag=0;
void loop()
{
  if(flag==1)
   {
     Serial.print(val);
     flag=0;
   }
}
void receiveData(int byteCount)
{
    while(Wire.available()>0)
    {
      val=Wire.read();
      flag=1;
    }
}
// callback for sending data
void sendData()
{
  Wire.write(0x45);
}
```

Dans void setup() :

- A l'aide de la librairie wire on établit la communication I2C via SCL et SDA
- On défini l'adresse pour la communication I2C

Un flag nous permet de savoir s'il est possible d'envoyer une donnée. Quand positionné à 1 nous renvoyons la valeur lue

Pour le positionner à 1 la fonction wire.avaliable() nous renvoi une valeur différente de 0 quand le maitre n'est pas occupé

Enfin nous envoyons le bit vers la brique.

En observant dans les ports de la brique, elle a bien reçu la valeur 0x45 ou 69.

5. Test avec ATMEGA 16

Mes camarades ayant travaillé sur un prototype de carte, il a fallu que je fasse un prototype fonctionnel pour pouvoir tester avec la carte, pour cela je me suis aidé du site web ElectroWings qui explique comment fonctionne les fonctions I2C avec les librairies basiques données par l'ATMEGA 16, nous allons tester si cela fonctionne correctement :

```
int main(void)
{
    DDRA = 0xFF;                        /* DDRA est une sortie */
    uint8_t slave_adress = 0x01;
    unsigned char nom_capteur[9]={0x49,0x55,0x53,0x56,0x69,0x64,0x61,0x76};
    unsigned char sensor_type[9]={0x43,0x6F,0x6D,0x70,0x61,0x73,0x73,0x20};
    int8_t ack_status=0;
    I2C_slave_init(slave_adress);       /* Initialisation de l'I2C en slave*/
    while (1)
    {
        unsigned char read_value = I2C_listen();
        int i;
        if((read_value >= (0x08)) && (read_value <= (0x0F)))
        {
            for(i=0 ; i<8 ; i++)
            {
                do{
                    ack_status=I2C_write(nom_capteur[i]);
                }while(ack_status == 0);
            }
        }
        if((read_value >= (0x10)) && (read_value <= (0x17)))
        {
            for(i=0 ; i<8 ; i++)
            {
                do{
                    ack_status=I2C_write(sensor_type[i]);
                }while(ack_status == 0);
            }
        }
        if(read_value == (0x42))
        {
            do{
                ack_status=I2C_write(0x45);
            }while(ack_status == 0);
        };
    };
};
```

Avec les prototypes :

```
#include <avr/io.h>
#define F_CPU 3686400
#include <util/delay.h>
void I2C_slave_init(uint8_t slave_adress);
int8_t I2C_write(char data);
int8_t I2C_listen();
```

Et les fonctions :

```
void I2C_slave_init(uint8_t slave_adress)
{
    TWAR = slave_adress;
    TWCR = (1<<TWEN)|(1<<TWEA)|(1<<TWINT);  /* on active l'I2C */
};

int8_t I2C_write(char data)
{
    uint8_t status;
    TWDR=data;                          /* Write data to TWDR to be transmitted */
    TWCR=(1<<TWEN)|(1<<TWINT)|(1<<TWEA);/* Mode I2C activ? et interruption*/
    while(!(TWCR&(1<<TWINT)));          /* on attend d'etre appel? */
    status=TWSR&0xF8;                   /* on lit le registre TWI */
    if(status==0xA0)                    /* Check for STOP/REPEATED START received */
    {
        TWCR|=(1<<TWINT);               /* Clear interrupt flag & return -1 */
        return -1;                      /* bit re?u avec succes */
    }
    if(status==0xB8)                    /* Check for data transmitted &ack received */
    {
        return 0;                       /* bit non re?u*/
    }
    if(status==0xC0)                    /* Check for data transmitted &nack received */
    {
        TWCR|=(1<<TWINT);               /* Clear interrupt flag & return -2 */
        return -2;                      /* bit re?u avec succes */
    }
    if(status==0xC8)                    /* Last byte transmitted with ack received */
        return -3;                      /* bit re?u avec succes */
    else
        return -4;                      /* bit re?u avec succes */
}
```

```
int8_t I2C_listen()
{
    while(1)
    {
        uint8_t status;
        while(!(TWCR&(1<<TWINT)));      /* on attend d'etre appel? */
        status=TWSR&0xF8;               /* on lit le registre TWI */
        if(status==0x60||status==0x68)
        return 0;                       /* Return 0 -> Slave en mode re?ois */
        if(status==0xA8||status==0xB0)
        return 1;                       /* Return 1 -> Slave en mode envoi */
        if(status==0x70||status==0x78)
        return 2;                       /* Return 2 -> Fin de la communication */
        else
        continue;                       /* Else continue */
    }
}
```

Avec la datasheet :

TWI Control Register – TWCR

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TWINT : Flag d'interruption, s'active à chaque fois que la donnée a été reçue ou envoyée.

TWEA : permet de générer le bit d'Acknowledge.

TWSTA : à 1, met le microcontrôleur en mode Master.

TWSTO : permet, quand en mode master de générer un bit de stop.

TWWC : Flag qui est mis à 1 quand le registre de donnée est écrit.

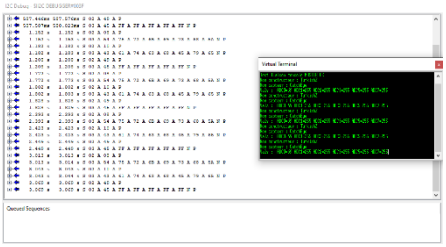
TWEN : active la liaison I2C.

TWIE : Mode Interrupt activé.

TWAR : adresse du Slave

TWDR : Registre des données (lecture)

Sur Proteus les tests paraissent corrects, je reçois bien des données, mais il n'en est rien du cas réel, cela doit être dû au STK500 ou simplement aux fonctions qui ne prennent pas en compte certains paramètres :



6. Utilisation d'un STK500

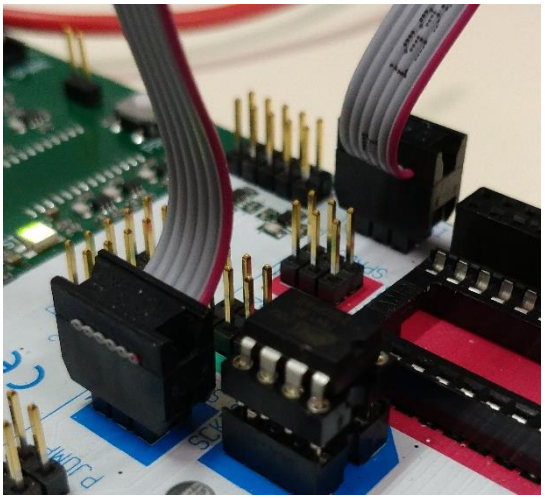
Après avoir testé mon programme pour l'ATMega16 avec la brique j'ai observé quelques problèmes. Bien que la simulation fonctionnât, la brique, elle, n'affichait qu'un problème de batterie à chaque connexion. De plus, la carte STK500 avait des résistances internes et d'autres connectiques.

Je me suis donc dis qu'il fallait avancer sur le projet final, c'est alors que j'ai voulu commencer la programmation sur l'ATTiny85. Bien qu'au premier abord cela pouvait paraître simple, il fallait penser à pleins de détails sur Atmel et la carte STK500. Premièrement il faut choisir l'emplacement. Le premier problème était qu'elle ne supportait pas l'ATTiny85 directement :

- ATtiny11
- ATtiny12
- ATtiny15
- ATtiny22
- ATtiny28
- AT90S1200
- AT90S2313
- AT90S2323
- AT90S2333
- AT90S2343
- AT90S4414
- AT90S4433
- AT90S4434
- AT90S8515
- AT90S8535
- ATmega8
- ATmega16
- ATmega161
- ATmega163
- ATmega323
- ATmega103(1)
- ATmega128(1)

Source :

Heureusement, cela n'est pas un problème car l'architecture de l'ATTiny85 correspond à celle de l'ATTiny15 (le MISO et le MOSI étant sur les mêmes pins).

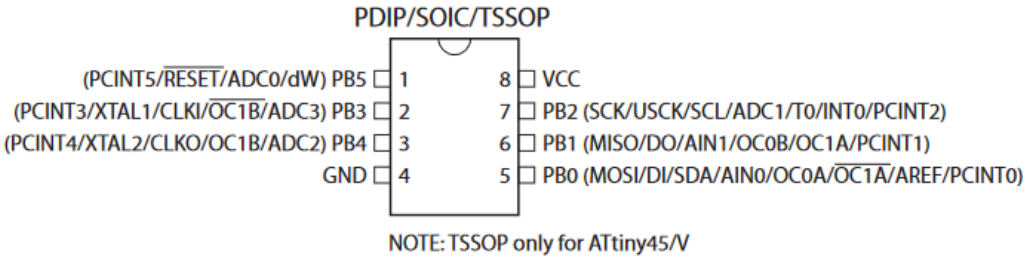


Comme nous pouvons le voir, il a aussi fallu brancher la connexion MOSI et MISO sur l'espace bleu, pour que les données s'envoient sur l'ATTiny85 et non l'ATMega16. On

pourrait penser que ça va marcher comme par miracle, mais non ! D'après certains forums il faut brancher deux pins du Port B à deux autres pins bien distincts.

Bien que dans le cas d'un ATMega16 il y a beaucoup de pins et donc beaucoup de place pour les mettre, ce n'est pas le cas avec notre ATTiny85 et ses pauvres 8 pins. Il faut donc faire attention à brancher la résistance de pull-up du RESET, ainsi que l'horloge à la première utilisation.

Figure 1-1. Pinout ATtiny25/45/85

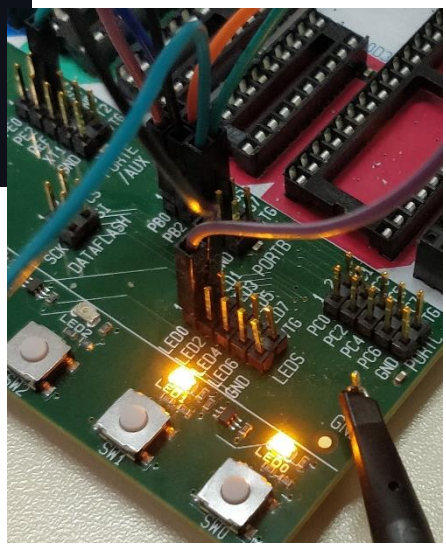


Il faut donc brancher les pins suivantes :

- **PB5** à **RST** *(pin Résistance de pull-up pour RESET)*
- **PB3** à **XT1** *(pin pour l'horloge externe)*

Après cela, nous pouvons faire nos premiers tests pour voir si L'ATTiny85 se programme correctement. Pour cela j'ai fait un programme simple qui va allumer les Leds de manière cyclique :

```c
#include <avr/io.h>
#define F_CPU 1000000
#include <util/delay.h>

int main(void)
{
    DDRB |= (1<<PB0);
    DDRB |= (1<<PB2);          //port b 0 et 2 en sortie
    while (1)
    {
        PORTB |= (1<<PINB0);    //les deux sont eteintes
        PORTB |= (1<<PINB2);
        _delay_ms(1000);
        PORTB &= ~(1<<PINB0);   //les deux sont allumees
        PORTB &= ~(1<<PINB2);
    };
}
```



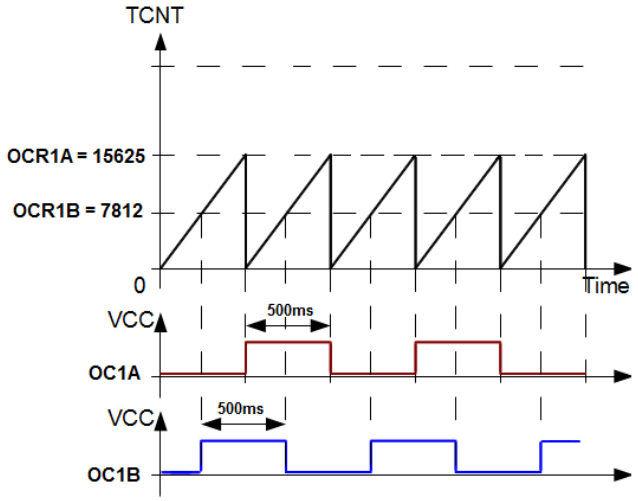
7. Création de timers pour les Leds

Comme Osman l'a précisé dans ses parties, nous connecterons les Leds sur les ports PB1 et PB4, et cela n'est pas dû au hasard, nous aurons deux timers, soit sur PB1 et PB4 le Timer 1. Comme vu sur la datasheet, nous avons sur PB1 le choix entre OC1A et OC0B, et pour PB4 OC1B, nous allons donc choisir la valeur de OCR1C pour comparer ces derniers.

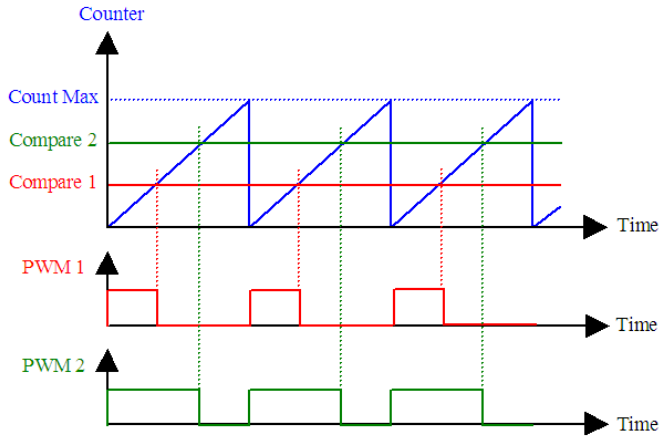
Tout d'abord il faut savoir quel bit nous allons utiliser pour faire une comparaison. Il est sûr que nous allons utiliser le mode CTC *(Clear Timer on Compare)* car nous n'aurons besoin que de 30 KHz, ce qui est très peu par rapport au 1 MHz donné par l'horloge. Le mode PWM est beaucoup plus intéressant si nous voulions une fréquence plus élevée.

Dans la documentation nous pouvons donc lire que OCR1A et OCR1B sont deux registres qui correspondent à l'incrémentation du Timer 1. OCR1C, dans notre cas avec le CTC, va être la valeur max de ces derniers. La grande différence entre les deux modes est la façon dont la sortie va être alimentée. Soit :

- Pour le mode CTC :



- Pour le mode PWM :



Comme nous pouvons le voir, dans le mode CTC nous avons la position haut après une comparaison entière, alors que le mode PWM lui met la position haut à chaque comparaison. De plus, le fait que le mode PWM ne nous permet pas d'avoir un rapport cyclique de 50% nous force à utiliser le mode CTC.

Pour trouver la formule la documentation ne nous aide guère, en effet elles ne sont que pour le mode PWM, soit :

> The frequency of the PWM will be Timer Clock 1 Frequency divided by (OCR1C value + 1). See the following equation:
>
> $$f_{PWM} = \frac{f_{TCK1}}{(OCR1C + 1)}$$

Mais ne nous laissons pas abattre ! En rétro-ingéniérant la formule nous pouvons récupérer la fréquence pour le mode CTC. Comme nous le savons c'est toutes les 2 comparaisons que notre période se fait. La pré-division, elle, est directement basée sur le processeur. Nous obtenons donc :

$$\frac{f_{CPU}}{2 \cdot prediv \cdot (OCR1C + 1)} = f_{CTC}$$

Encore une fois, d'après Osman, les Leds que nous utilisons doivent être alimentées avec une fréquence de 30 KHz, nous cherchons donc tout d'abord le prediv le plus précis, soit (en faisant varier OCR1C de 1 à 255) :

- Prediv de 1 -> entre 1.95 kHz et 250 KHz
- Prediv de 2 -> entre 977 Hz et 125 KHz

Bon, nous pouvons déjà arrêter là car le prediv 2 est bien moins précis pour des fréquences élevées. Nous allons donc rester sur un prediv de 1, et pour calculer OCR1C il faut simplement remplacer :

$$OCR1C = \frac{f_{CPU}}{2 \cdot prediv \cdot f_{CTC}} - 1$$

$$AN : \frac{1 \times 10^6}{30 \times 10^3 \times 2} - 1 = 15.6$$

Nous allons donc prendre OCR1C = 16.

Maintenant que nous avons toute la théorie de faite, nous pouvons enfin passer à la programmation des registres et au test pratique.

D'après la documentation, nous avons un registre commun à OC1B et OC1A, et un registre qui va nous servir pour la configuration de OC1B. Ils se nomment TCCR1 et GTCCR. Bien évidemment avant de paramétrer ces derniers, il faut mettre PB1 et PB4 en sortie, nous les mettons donc à 1.

TCCR1 – Timer/Counter1 Control Register

Bit	7	6	5	4	3	2	1	0	
0x30	CTC1	PWM1A	COM1A1	COM1A0	CS13	CS12	CS11	CS10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

- COM1A[1;0]

Ces bits nous permettent de choisir si nous voulons que la sortie PB1/PB4 soit mise à 1, mise à 0 ou change d'état entre chaque comparaison. D'après la datasheet nous avons :

- 00 = pin désactivée
- 01 = change d'état
- 10 = niveau bas
- 11 = niveau haut

Bien évidemment ce qui nous intéresse c'est le deuxième cas pour avoir un rapport cyclique de 50%.

- CS1[3;0]

Ces bits nous permettent de choisir la pré-division, d'après la datasheet voici les possibilités :

Table 12-5. Timer/Counter1 Prescale Select

CS13	CS12	CS11	CS10	Asynchronous Clocking Mode	Synchronous Clocking Mode
0	0	0	0	T/C1 stopped	T/C1 stopped
0	0	0	1	PCK	CK
0	0	1	0	PCK/2	CK/2
0	0	1	1	PCK/4	CK/4
0	1	0	0	PCK/8	CK/8
0	1	0	1	PCK/16	CK/16
0	1	1	0	PCK/32	CK/32

(Cela s'étend jusqu'à CK/16384)

Nous n'allons pas entrer dans les détails du mode PCK (Horloge Asynchrone) et de toute manière elle ne nous sera pas utile. Mais pour l'activer il aurait fallu changer un bit nommé PCKE après 100 us du mode PLL. Comme nous pouvons le voir il faut simplement mettre CS1[3;0] à 1 pour pouvoir activer l'horloge avec un prediv de 1.

GTCCR – General Timer/Counter1 Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C	TSM	PWM1B	COM1B1	COM1B0	FOC1B	FOC1A	PSR1	PSR0	GTCCR
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

- COM1B[1;0]

Elle fonctionne exactement comme COM1A[1;0] mais pour PB4. Nous allons donc mettre ce registre à 01.

Voici les fonctions associées :

```c
#include "led_gen.h"

void led1_init(void)
{
    DDRB |= (1<<PB1);
    TCCR1|=(1<<CTC1)|(1<<COM1A0);       //on active le mode CTC en comparant OCR1C, COM1A0 : mode toggle
    OCR1C=16;                           //d'apres retroingenieurie OCR1C=16
    TCCR1|=(1<<CS10);                   //on active la clock avec CS10 = 1, pas de prediv (plus precis pour haute freq)
}

void led2_init(void)
{
    DDRB |= (1<<PB4);
    GTCCR|=(1<<COM1B0);                 //COM1B0 : mode toggle
    OCR1C=16;                           //d'apres retroingenieurie OCR1C=16
    TCCR1|=(1<<CS10);                   //on active la clock avec CS10 = 1, pas de prediv (plus precis pour haute freq)
}

void led1_stop(void)
{
    DDRB &= ~(1<<PB1);
    TCCR1 &= ~(1<<COM1A0);
}

void led2_stop(void)
{
    DDRB &= ~(1<<PB4);
    GTCCR &= ~(1<<COM1B0);
}
```

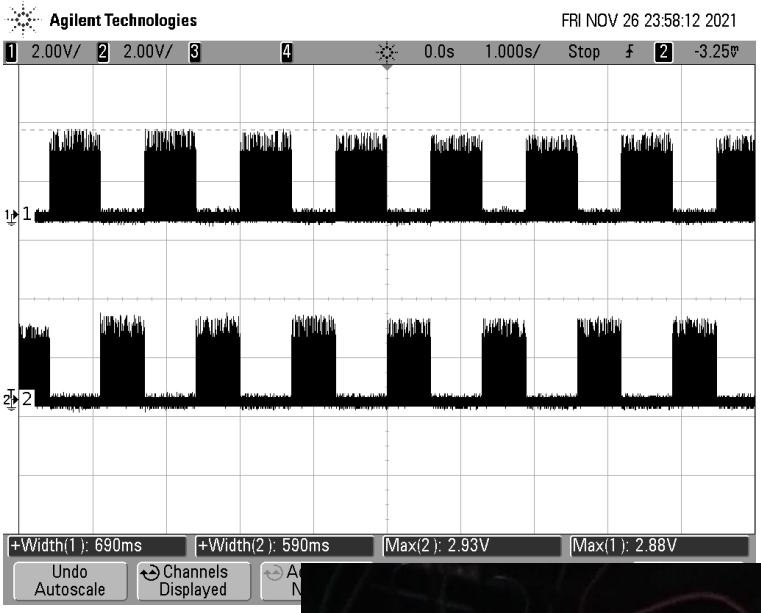
8. Délai et test

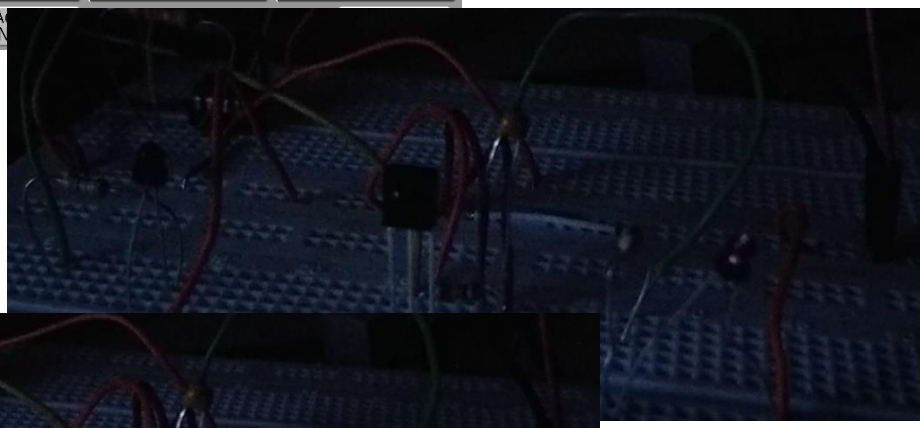
Pour pouvoir récupérer l'information si un objet est à droite ou à gauche, ou même au centre il faut deux Leds. Le but ici sera d'allumer une led, de récupérer la tension, et ainsi de suite. Pour faire cela il nous faut donc des délais, et pour le test nous n'allons pas utiliser de grands délais. Soit le programme suivant :

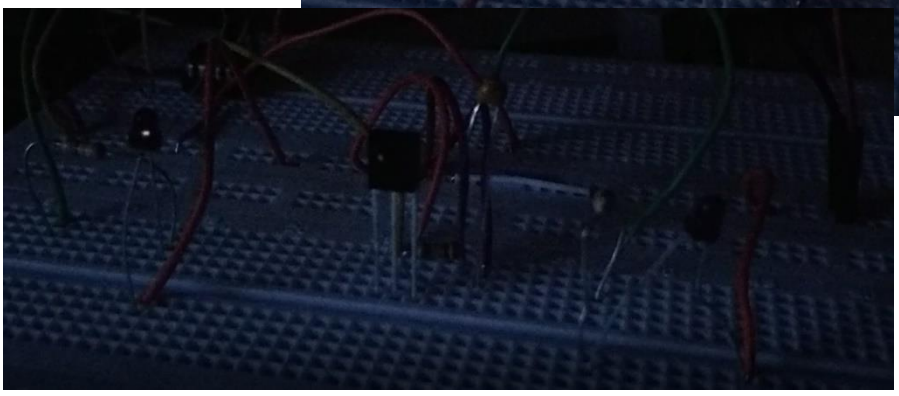
```c
void measureLed1(void)
{
    led1_init();
    _delay_ms(TIMING);
    led1_stop();
};
```

Avec ***TIMING*** une variable que nous changerons par la suite.

Pour les tests nous avons mis 500 ms pour pouvoir l'observer correctement sur l'oscilloscope ainsi que sur les Leds IR. A l'aide du montage que Charles a fait nous nous retrouvons avec les chronogrammes suivants :







Comme nous pouvons le voir les Leds s'allument bien correctement, et d'après le chronogramme avec un intervalle de 500 ms.

9. Conversion Analogique Numérique

Les Leds ne font pas tout, nous possédons aussi par la suite un capteur infrarouge qui nous enverra différentes tensions. N'ayant pas encore testé le fonctionnement de ce récepteur nous sommes partis du principe qu'elle nous envoie une tension quand la Led est allumée et sinon 0V. Bien que cela peut être faux, ce n'est pas très grave car l'algorithme sera déjà à peu près fonctionnel à quelques détails près.

J'ai tout d'abord écrit mon algorithme sur Code : Blocks. On récupère donc deux tensions différentes et on les compare pour savoir si il y a un obstacle ou non, si cet obstacle est à droite, gauche ou devant.

Soit le programme suivant :

```
int main()
{
    float adcLed1, adcLed2;
    while(1)
    {
    cout << "adcLed1 = ";
    cin >> adcLed1;
    cout << "adcLed2 = ";
    cin >> adcLed2;
    float adcCompare = adcLed1 - adcLed2;
    cout << "adcCompare = " << adcCompare << endl;
    if (adcCompare < -VOLTMIN)  // adcLed2 > adcLed1
    {
        cout << "LEFT" << endl;
    }
    else if (adcCompare > VOLTMIN)
    {
        cout << "RIGHT" << endl;
    }
    else
    {
        if((adcLed1 > VOLTMIN) && (adcLed2 > VOLTMIN))
        {
            cout << "FRONT" << endl;
        }
        else
        {
            cout << "NOTHING" << endl;
        };
    };
    };
    return 0;
}
```

```
D:\Alzyohan\Documents\
adcLed1 = 1
adcLed2 = 1
adcCompare = 0
NOTHING
adcLed1 = 1.5
adcLed2 = 1.5
adcCompare = 0
FRONT
adcLed1 = 2
adcLed2 = 0
adcCompare = 2
RIGHT
adcLed1 = 0
adcLed2 = 2
adcCompare = -2
LEFT
adcLed1 =
```

Dans l'algorithme que j'ai créer j'ai laissé une légère erreur de 1 V. Comme ça j'évite d'avoir des problèmes de mesure dans le cas ou la tension est supérieure à 0 alors qu'il n'y a rien. Il ne manque plus qu'à convertir ce programme en C++ en C, et à faire la conversion analogique.

Nous allons donc utiliser PB3 pour la conversion analogique numérique. Il faut donc la mettre en entrée. Nous aurons besoin ici de 3 fonctions pour que cette conversion fonctionne. La première est évidente, c'est l'initialisation, la seconde est pour lancer une conversion et enfin une pour lire la valeur renvoyée par le microcontrôleur.

Tout d'abord il nous faut choisir le port de sortie du CAN, pour cela nous pouvons voir sur la documentation les bits MUX[3;0] du registre :

Table 17-4. Input Channel Selection

MUX[3:0]	Single Ended Input
0000	ADC0 (PB5)
0001	ADC1 (PB2)
0010	ADC2 (PB4)
0011	ADC3 (PB3)

(Les autres modes ne nous intéressent pas car nos tensions n'ont pas de différentiel et son directement liés au GND)

Nous prendrons donc MUX[3;0]=0011 car nous voulons PB3 comme entrée. D'ailleurs MUX ne provient pas de nulle part, il vient du registre ADMUX, que voici :

17.13.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
0x07	REFS1	REFS0	ADLAR	REFS2	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Il y a aussi un autre bit que nous devons changer dans ce registre et c'est ADLAR, c'est un pur choix d'optimisation de mémoire car la tension que nous allons récupérer n'a pas besoin d'être très précise. En effet, ADLAR possède deux modes :

- ADLAR = 0 -> ADCL possède les bits de 0 à 7, ADCH possède les bits de 8 à 9.
- ADLAR = 1 -> ADCL possède les bits de 0 à 1, ADCH possède les bits de 2 à 9.

Comme cela nous n'aurons qu'à relever ADCH dans un char (8 bits donc).

La prochaine étape est de décider la fréquence à laquelle nous allons récupérer cette tension pour la convertir en valeur numérique. Il est indiqué dans la datasheet que la fréquence de celle-ci doit être inférieure à 200 KHz. En regardant le registre et les bits associés nous avons donc cela :

17.13.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
0x06	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Dans ADCSRA les bits qui nous intéresse pour l'instant sont ADPS[2;0], avec la table suivante :

Table 17-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Nous possédons un processeur avec une fréquence d'horloge de 1 MHz, il nous faut donc une division de **16** pour avoir **62.5 kHz**, comme ça nous sommes inférieur à 200 KHz. *(J'aurais pu prendre 8, soit 125 kHz mais je voulais être sûr)*

Nous aurons donc finalement ADPS[2;0] = (100)

Les deux bits que nous mettrons à 1 par la suite sont ADEN et ADATE. ADEN doit obligatoirement être à 1 pour activer le module de conversion analogique numérique. ADATE permet, lui, de faire une conversion automatiquement à partir d'un certain bit. Comme nous voulons utiliser les interruptions nous en aurons besoin.

Mais justement, en parlant d'interruption, pourquoi nous voulons les activer ? Il aurait été bien plus simple de ne faire qu'une conversion au moment voulu. Avec la liaison I2C qui va être réalisé après. Nous ne pouvons pas nous autoriser à laisser le microcontrôleur attendre pendant quelques millisecondes, c'est pour cela que nous mettons une interruption. Pour pouvoir faire la conversion même quand le microcontrôleur est en train de communiquer avec la brique.

ADIE est donc le prochain bit que nous allons mettre à 1, et cela pour activer l'interruption sur la conversion analogique numérique.

17.13.4 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
0x03	BIN	ACME	IPR	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R/W	R/W	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADCSRB est le prochain registre qui va nous intéresser car nous voulons être en « Free-running mode », ce mode de CAN nous permet de faire une conversion à tout moment selon l'interruption, il faut donc être bien sûr que les bits ADTS[2;0] sont correctement paramétrés pour pouvoir faire une interruption.

Nous avons donc le tableau suivant :

Table 17-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter0 Compare Match B
1	1	0	Pin Change Interrupt Request

Il faut bien mettre ADTS[2;0] = (000).

```
void ADC_init(void)
{
    DDRB &=~(1<<PB3);                          //Pin B3 en entree
    ADMUX |= (1<<MUX1)|(1<<MUX0)|(1<<ADLAR);   //MUX = 0011 donc sortie PB3 et ADLAR a 1 pour avoir les bits de poids fort en premier
    ADCSRA|= (1<<ADEN)|(1<<ADPS2)|(1<<ADIE)|(1<<ADATE);   //ADEN conversion autorisee
                                               //On divise 1Mhz par 16 (62.5 KHz) < 200 KHz
                                               //Interruptions autorisees
                                               //Auto-Trigger autorise
    ADCSRB &= ~(0b111<<ADTS0);                 //free-running mode
    sei();
};
```

Sans oublier sei() qui va nous permettre de lancer l'interruption selon la fonction ISR.

D'après la documentation, le bit **ADSC** est celui qui va nous permettre de lancer une conversion, mais il a une particularité en free-running. En effet, il faut d'abord mettre ce bit à 1 au tout début du programme. Il se mettra automatiquement à 0 pour finir la conversion. Par la suite, dans l'interruption nous relancerons cette conversion.

Pour récupérer cette valeur, ce bit nous permettra de savoir quand la conversion est finie et donc comment récupérer cette valeur. Nous avons donc finalement, deux fonctions :

```
void ADC_start_conversion(void)
{
    ADCSRA|=(1<<ADEN)|(1<<ADIE)|(1<<ADSC);
};

char ADC_read_value(void)
{
    ADCSRA|=(1<<ADSC);
    while(ADCSRA & (1<<ADSC));
    return ADCH;
};
```

Pour finir, nous avons les fonctions suivantes :

```
float measureLed1(void)
{
    float adcLed;
    led1_init();
    _delay_ms(TIMING);
    adcLed=ADC_averaging(ADCH);
    _delay_ms(TIMING);
    led1_stop();
    return adcLed;
};

float measureLed2(void)
{
    float adcLed;
    led2_init();
    _delay_ms(TIMING);
    adcLed=ADC_averaging(ADCH);
    _delay_ms(TIMING);
    led2_stop();
    return adcLed;
};
```

```
unsigned char compareLed(float led1,float led2)
{
    float adcLed1 = led1;
    float adcLed2 = led2;
    float adcCompare = led1 - led2;
    if (adcCompare < -VOLTMIN)  // adcLed2 > adcLed1
    {
        return LEFT;
    }
    else if (adcCompare > VOLTMIN)
    {
        return RIGHT;
    }
    else
    {
        if((adcLed1 > VOLTMIN) && (adcLed2 > VOLTMIN))
        {
            return FRONT;
        }
        else
        {
            return NOTHING;
        };
    };
};
```

Dans le main.c :

```
volatile unsigned char rawVal;
volatile float adcled1;
volatile float adcled2;
volatile unsigned char recVal;

ISR (ADC_vect){
    adcled1=measureLed1();
    adcled2=measureLed2();
    recVal=compareLed(adcled1,adcled2);
}
```

```
sei();
ADC_init();
ADC_start_conversion();
```

Le test fut très concluant, en changeant la tension d'entrée nous avions différentes valeurs qui s'affichait sur la brique, soit 255 pou 5V, 128 pour 2.5V et 0 pour 0V. En fonction des tensions que nous recevrons nous pourrons donc faire un algorithme qui affichera des valeurs prédéfinie en fonction de la position de l'obstacle.

10. Liaison I2C

J'ai déjà expliqué précédemment comment fonctionnait cette liaison, malheureusement sur l'ATTiny85 il n'est pas possible de faire directement une liaison I2C, nous avons un module dit USI (Universal Serial Interface) qui va nous permettre cependant, de simuler cette liaison.

A l'aide d'une librairie nommée « USI_TWI_Slave »

En ouvrant cette librairie j'ai remarqué quelques coquilles qu'il faut régler pour que la liaison fonctionne sans accroc. Tout d'abord le nom des registres n'était pas bien renseigné pour l'ATTiny85, j'ai donc réglé le tir :

```
111  #if defined( __AVR_ATtiny25__ ) | \
112  defined( __AVR_ATtiny45__ ) | \
113  defined( __AVR_ATtiny85__ )
114      #define DDR_USI             DDRB
115      #define PORT_USI            PORTB
116      #define PIN_USI             PINB
117      #define PORT_USI_SDA        PB0
118      #define PORT_USI_SCL        PB2
119      #define PIN_USI_SDA         PINB0
120      #define PIN_USI_SCL         PINB2
121      #define USI_START_COND_INT  USISIF
122      #define USI_START_VECTOR    USI_START_vect
123      #define USI_OVERFLOW_VECTOR USI_OVF_vect
124  #endif
```

- Notamment PB0 et PB2 qui étaient nommés PORTB0 et PORTB2
- USISIF, un bit du registre USISIR qui est un flag d'interruption pour le démarrage d'une liaisons TWI (Two Wire Interface, l'autre nom de l'I2C)

Après cela, les erreurs d'Atmel ont disparues, il est donc maintenant temps d'écrire le main.c pour communiquer avec la brique !

- USI_TWI_Slave_Initialise(slaveAdress);

Permet d'initialiser l'adresse de l'esclave, comme son nom l'indique, avec un unsigned_char.

- USI_TWI_Data_In_Receive_Buffer();

Une function qui renvoi 1 quand la liaison TWI n'est pas occupée, sinon 0.

- USI_TWI_Receive_Byte() ;

Renvoie la donnée en char.

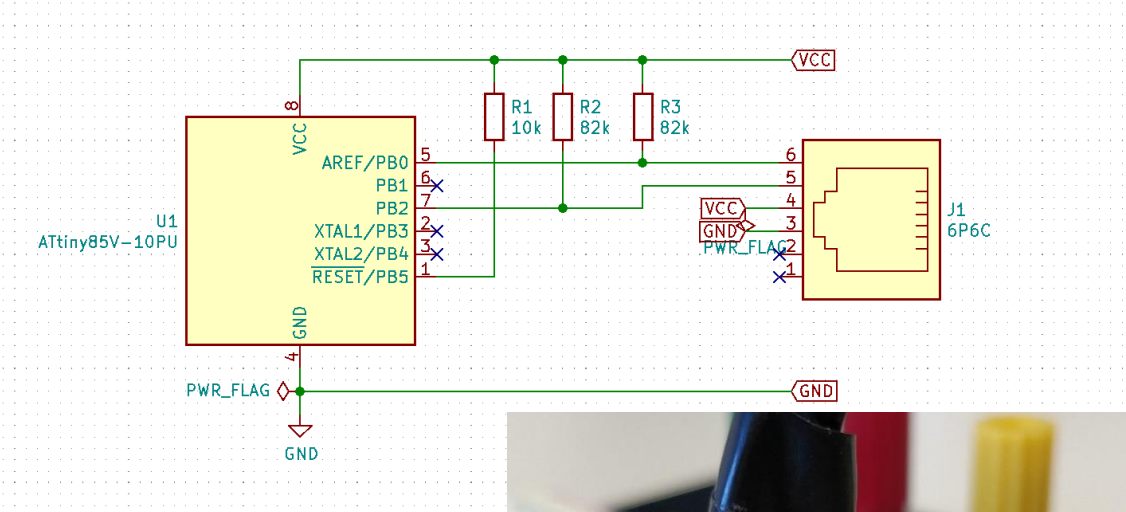
- USI_TWI_Transmit_Byte() ;

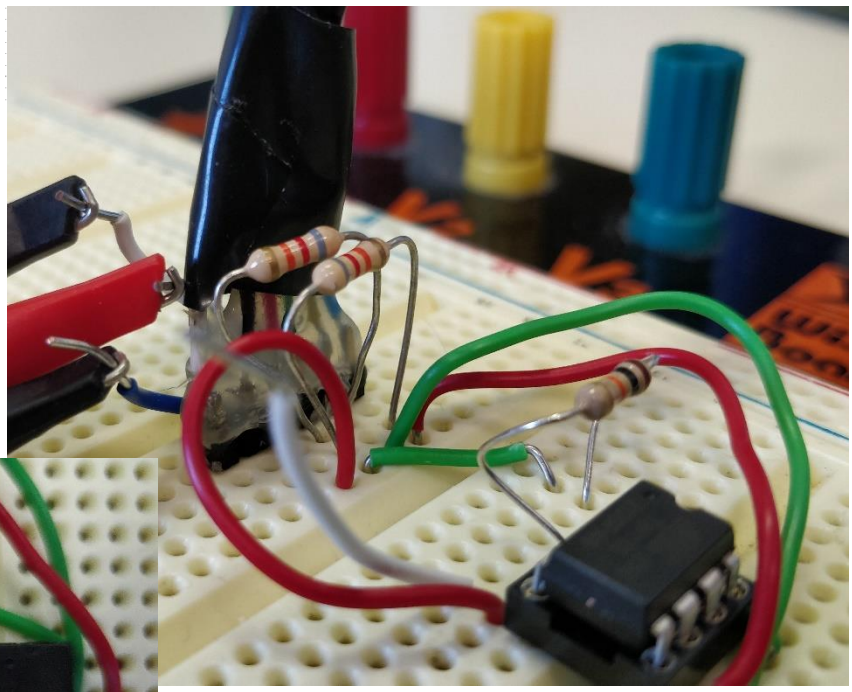
Envoie une donnée en char.

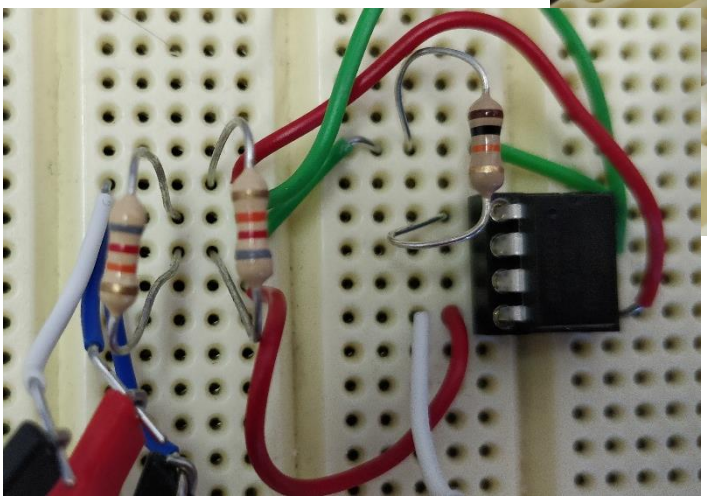
```
#define NOM_CONSTRUC    0x08
#define NOM_CAPTEUR     0x10
#define VALEUR_DEMAN    0x49
#define BASIC_MODE      0x42
```

```
int main( void )
{
    unsigned char slaveAdress, cmd,i=0;
    unsigned char nom_capteur[8]= {'T','u','r','k','i','s','h','Z'};
    unsigned char sensor_type[8]= {'C','a','t','c','h','E','y','e'};
    slaveAdress = 0x01;
    MCUCR |= (1<<PUD);

    USI_TWI_Slave_Initialise(slaveAdress);

    sei();
    ADC_init();
    ADC_start_conversion();
        while(1)
        {
            if( USI_TWI_Data_In_Receive_Buffer() )
            {
                cmd = USI_TWI_Receive_Byte();
                switch(cmd)
                {
                case NOM_CONSTRUC :
                    for (i=0; i<8; i++)
                    {
                        USI_TWI_Transmit_Byte(nom_capteur[i]);
                    }
                    break;

                case NOM_CAPTEUR :
                    for (i=0; i<8; i++)
                    {
                        USI_TWI_Transmit_Byte(sensor_type[i]);
                    }
                    break;

                case VALEUR_DEMAN :
                    USI_TWI_Transmit_Byte(0x45/*recVal*/);
                    break;

                case BASIC_MODE :
                    USI_TWI_Transmit_Byte(0x46);
                    break;
                }
            }
        }
        return 0;
}
```

Ensuite à l'aide d'un montage, ci-dessous, je peux lire correctement le dialogue entre la brique lego et l'ATTiny85.

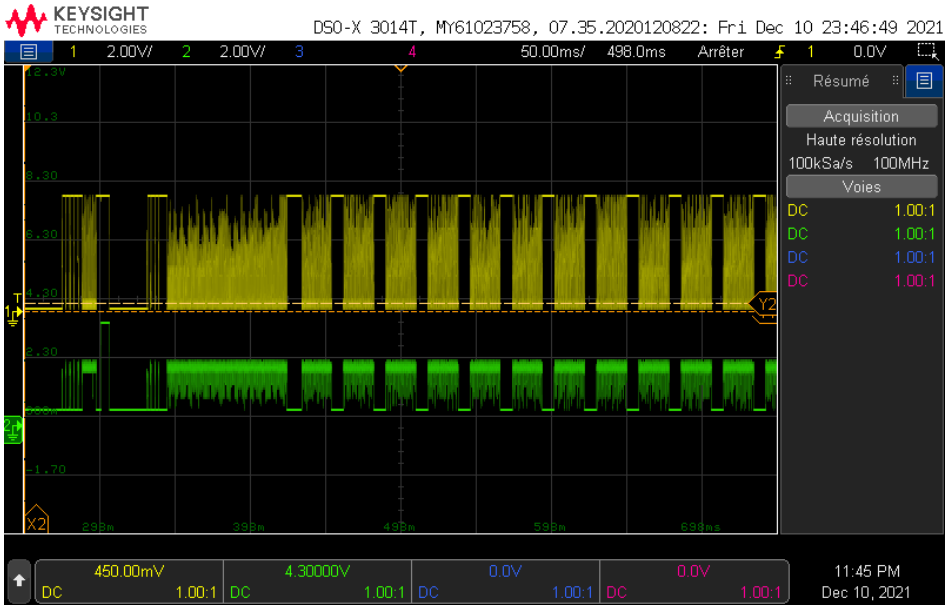




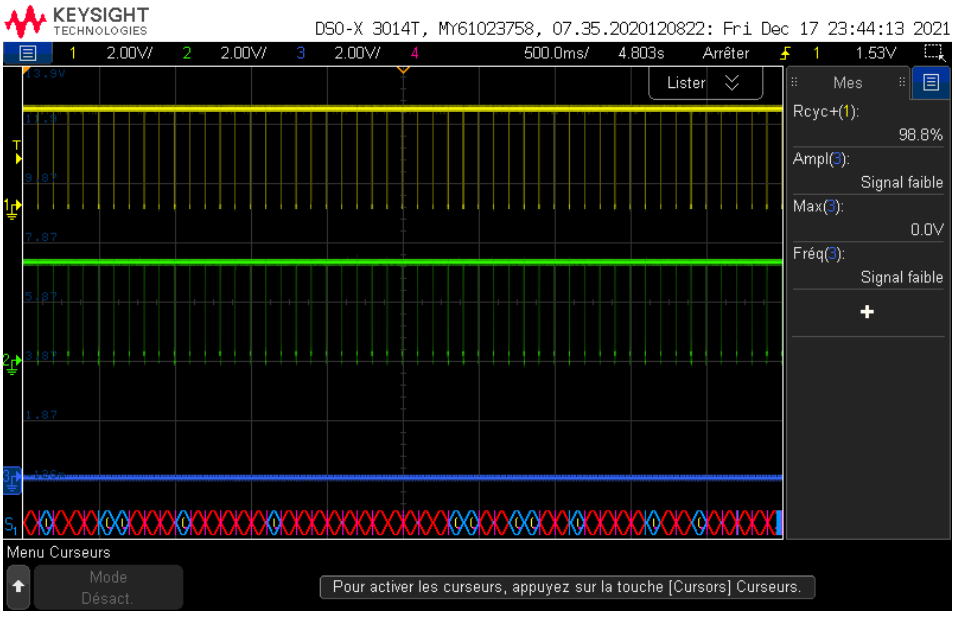


Il est donc temps maintenant de regarder sur l'oscilloscope ce que nous avons, en espérant que ce soit magnifique !

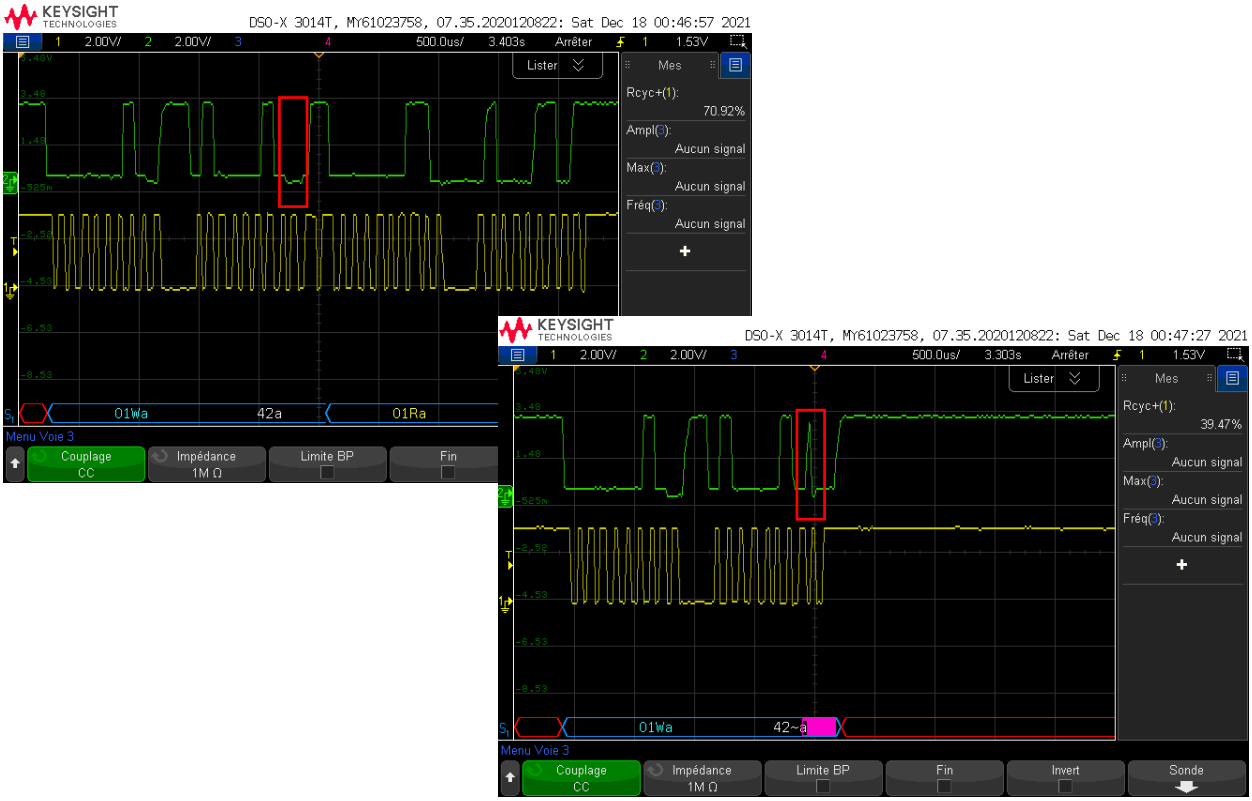
J'ai vraiment hâte !



… Disons que ce n'est pas exactement ce que je voulais. Le problème provient clairement des résistances de pull-up. Au moment de ce screen je n'avais pas encore mis celles-ci. Et après analyse sur différentes résistances et avec l'aide de mes camarades de l'autre groupe, les résistances de 82 kΩ paraissaient comme une évidence. L'autre problème était la résistance de pull-up interne de l'ATTiny85, mais en regardant la datasheet, un bit nommé PUD du registre MCUCR nous permet de la désactiver, vous pouvez revoir cela dans le main.c ci-dessus.

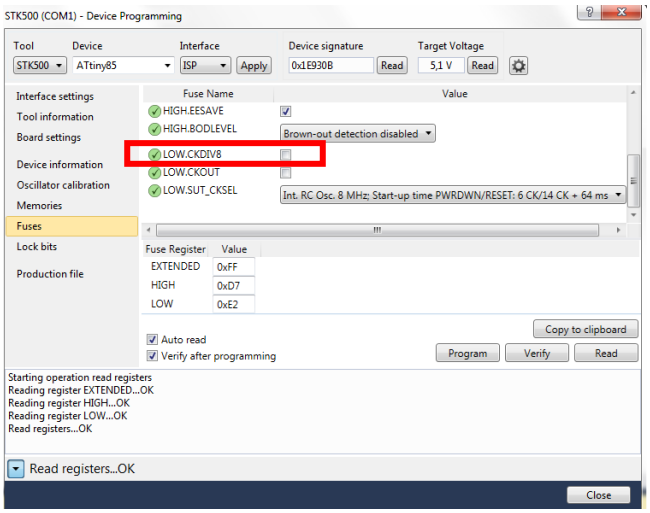


Nous possédons clairement quelque chose ! Ce n'est pas parfait en effet mais la brique nous affiche bien un bit que nous voulions recevoir (0x46 -> 70).



Comme nous pouvons le voir, un bit du SDA n'est pas correct, il a une forme de pic. C'est l'acknowledge, qui devrait être à 0. Après quelques minutes de réflexion, des recherches sur les résistances, sur le code. J'en suis venu à une conclusion : l'horloge du microcontrôleur n'est pas assez rapide. D'après certaines documentations en ligne l'ATTiny85 peut aller jusqu'à 8 MHz, mais comme vous l'avez vu sur tous nos calculs. Il est à 1 MHz.

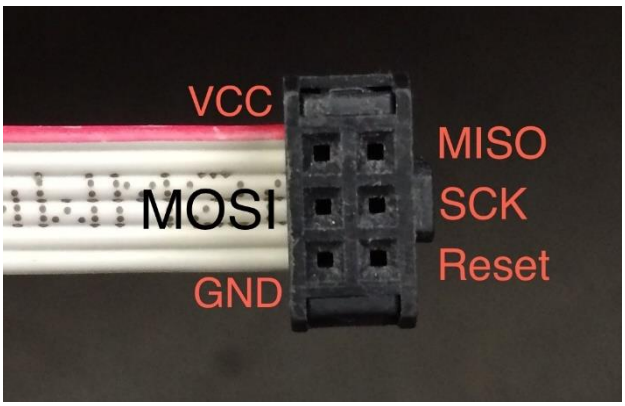
Un simple paramètre dans Atmel Studio nous permet de changer cela. Une fois dans « Device Programming », il suffit de décocher « LOW.CKDIV8 », un fusible qui divise automatiquement la fréquence par 8. Nous sommes maintenant à 8 MHz, et la brique Lego nous affiche constamment 70 (0x46).

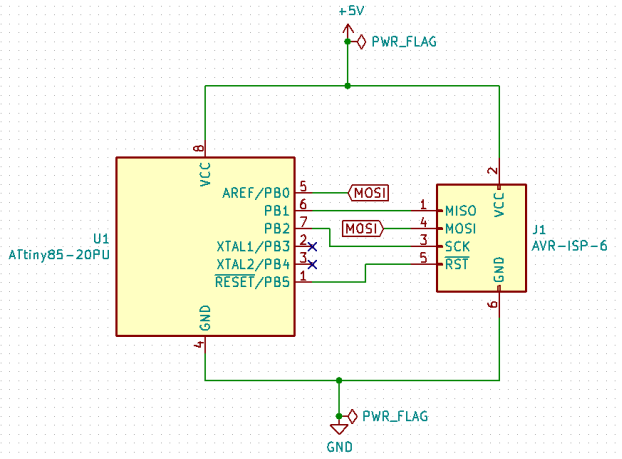


11. Programmation avec l'AVRISP MK2

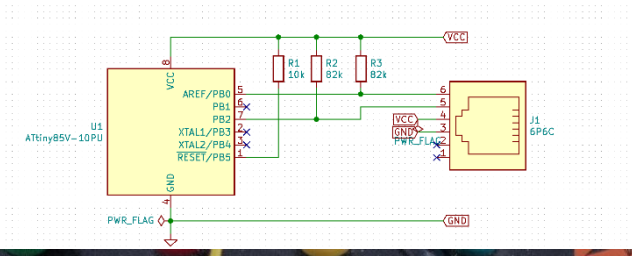
Un problème s'est posé lors de la deuxième phase de développement : Atmel Studio ne fonctionne plus sur les ordinateurs ayant des ports COM. Le seul argument dans l'utilisation de ces ordinateurs était la simplicité de programmation avec le STK-500.

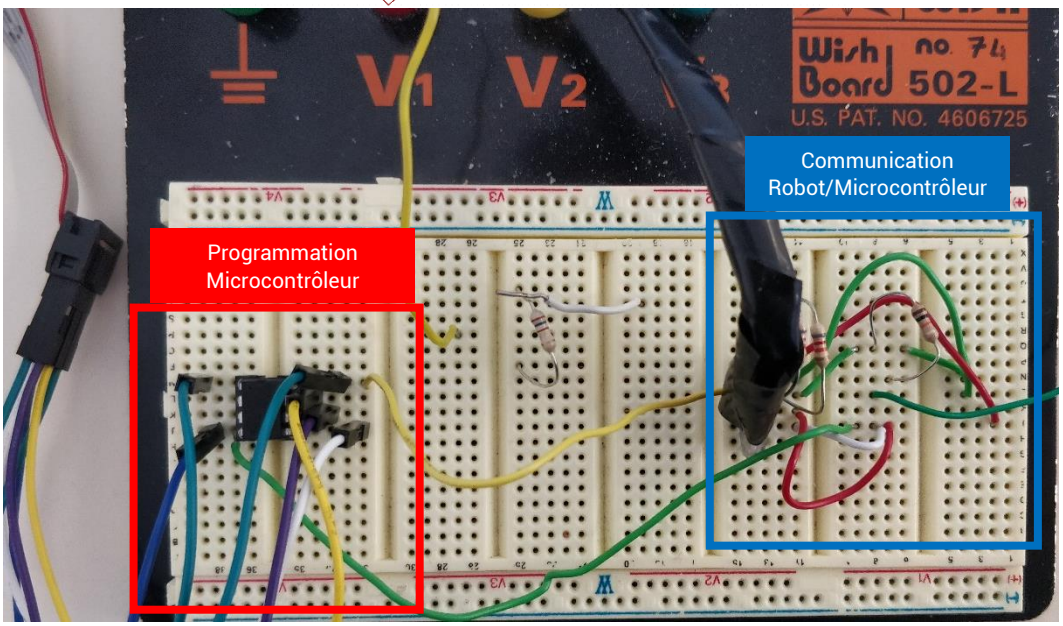
Une autre méthode possible est d'utiliser le module AVRISP MK2, mais pour cela il faut savoir quels pins relier, la datasheet ne nous expliquant pas exactement le sens du connecteur, nous nous sommes aidés d'une image :



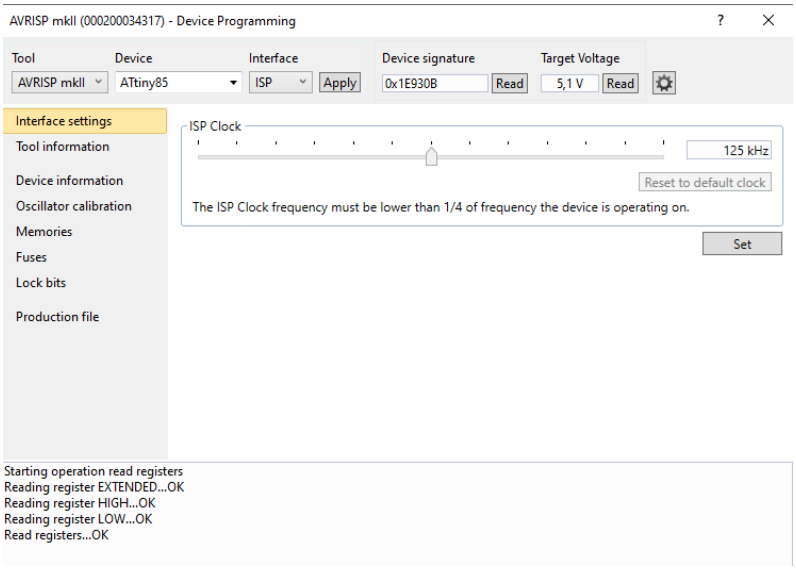


Pour rappel, nous avions déjà une partie pour la communication avec le robot, nous obtenons donc deux parties distinctes, pour rappel :





Il suffit par la suite de vérifier si la tension est correcte, ainsi que son identifiant dans la section « Device Programming », nous pouvons observer cela :



D'après le texte écrit tout en bas, tout paraît correct, il nous reste plus qu'à programmer pour observer les résultats. Avec ces deux modules sur la plaque nous pouvons simplement brancher tout ce dont nous avons besoin sur les pins du microcontrôleur ainsi que simplement le programmer.

12. Amélioration de la liaison I2C

L'un des grands problèmes observés après la mise en place de cette liaison est l'incapacité à pouvoir réaliser une CAN ou CNA. Le problème venant tout d'abord de la fréquence d'horloge du microprocesseur et ensuite de l'utilisation d'une librairie mal optimisée.

En regardant dans la documentation de l'ATTiny85 :

Table 20-5. Fuse Low Byte

Fuse Low Byte	Bit No	Description	Default Value
CKDIV8 (1)	7	Clock divided by 8	0 (programmed)
CKOUT (2)	6	Clock output enabled	1 (unprogrammed)
SUT1 (3)	5	Start-up time setting	1 (unprogrammed)(3)
SUT0 (3)	4	Start-up time setting	0 (programmed)(3)
CKSEL3 (4)	3	Clock source setting	0 (programmed)(4)
CKSEL2 (4)	2	Clock source setting	0 (programmed)(4)
CKSEL1 (4)	1	Clock source setting	1 (unprogrammed)(4)
CKSEL0 (4)	0	Clock source setting	0 (programmed)(4)

Il y a plusieurs fusibles programmables. Le plus important à noter est CKDIV8 qui, comme expliqué précédemment, permet de diviser ou non l'horloge par 8. Il faut

mettre ce bit à 1 pour le désactiver. Il y a ensuite CKOUT qui permet de sortir l'horloge du microcontrôleur sur PB3 (CLKI) , 1 pour la désactiver, elle nous servira plus tard.

Enfin, il nous reste les bits SUT et CKSEL, soit d'après la documentation :

Table 6-4. High Frequency PLL Clock Operating Modes

CKSEL[3:0]	Nominal Frequency
0001	16 MHz

Table 6-5. Start-up Times for the High Frequency PLL Clock

SUT[1:0]	Start-up Time from Power Down	Additional Delay from Power-On Reset (V_{CC} = 5.0V)	Recommended usage
01	14CK + 16K (16384) CK + 4 ms	4 ms	Fast rising power
10	14CK + 1K (1024) CK + 64 ms	4 ms	Slowly rising power
11	14CK + 16K (16384) CK + 64 ms	4 ms	Slowly rising power

Nous obtenons donc

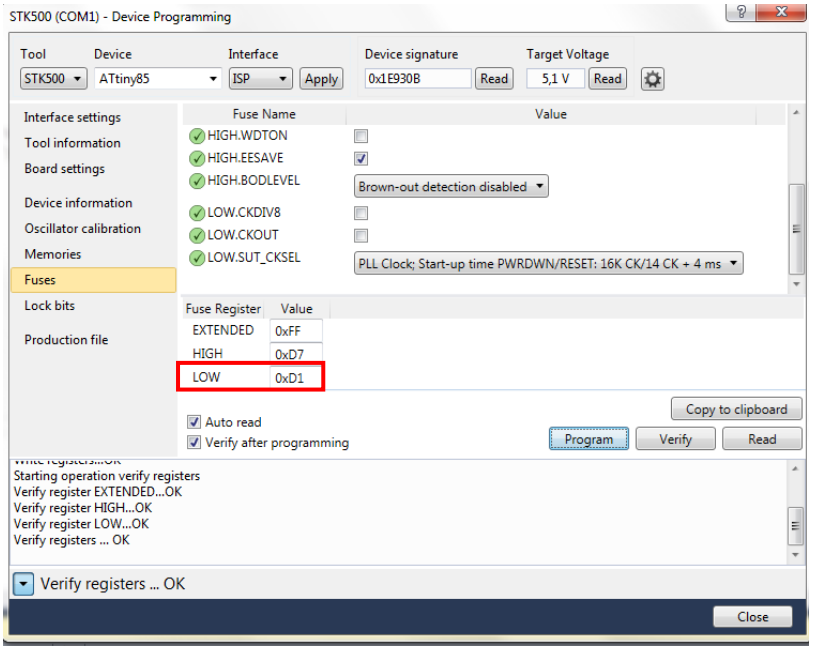
- **SUT**[1:0] = 01
- **CKSEL**[3:0] = 0001

Le choix technique de prendre le temps de démarrage le plus court est simple : La brique lego se connectera instantanément et nous voulons que le microcontrôleur réponde instinctivement.

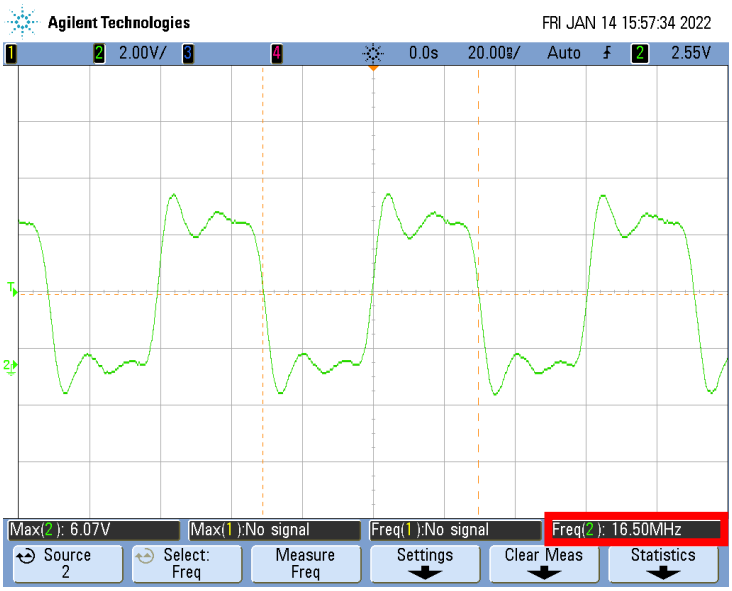
Nous nous retrouvons donc à programmer les fusibles de cette manière :

CKDIV8	CKOUT	SUT		CKSEL			
1	1	0	1	0	0	0	1
D				1			

Finalement nous programmerons 0xD1 sur les fusibles dits « Low »:



Pour tester si nous avons bien cette fréquence, il faut changer CKOUT à 0 de manière à observer sur PB3 le cycle d'horloge à 16 MHz.



La fréquence d'horloge est donc de 16.5 MHz, cela a donc bien fonctionné !

La prochaine étape concernant l'amélioration de la liaison I2C est une nouvelle librairie. La dernière étant plutôt vieille (2007) elle pose certains problème quant à la rapidité d'exécution ou encore les perturbations avec les autres interruptions. Après de longues recherches sur le site GitHub j'ai pu trouver une librairie qui était à l'origine destinée à la gestion d'un LCD à l'aide d'un Arduino :

https://github.com/rumpelrausch/Arduino-LCD-Philips-ET8861S-I2C

Dans les fichiers, nous pouvions retrouver :

usiTwiSlave.c	feat: is git initialized	10 months ago
usiTwiSlave.h	feat: is git initialized	10 months ago

Soit la plus récente version de la librairie USI_TWI_SLAVE, utilisant toutes les normes actuelles, ainsi qu'avec quelques améliorations, nous pouvons d'ailleurs lire un changelog directement dans le code du .c :

```
Change Activity:

    Date       Description
   ------      -------------
  16 Mar 2007  Created.
  27 Mar 2007  Added support for ATtiny261, 461 and 861.
  26 Apr 2007  Fixed ACK of slave address on a read.
  04 Jul 2007  Fixed USISIF in ATtiny45 def
  12 Dev 2009  Added callback functions for data requests
  06 Feb 2016  Minor change to allow mutli-byte requestFrom() from master.
  10 Feb 2016  Simplied RX/TX buffer code and allowed use of full buffer.
  13 Feb 2016  Made USI_RECEIVE_CALLBACK() callback fully interrupt-driven
  12 Dec 2016  Added support for ATtiny167
  23 Dec 2017  Fixed repeated restart (which broke when making receive callback
                interrupt-driven)
```

Pour l'incorporer directement dans notre code il a fallu changer le nom de quelques fonctions, ainsi qu'inclure cette nouvelle librairie, soit :

```
usiTwiSlaveInit(slaveAdress);

sei();

while(1)
{
    if( usiTwiDataInTransmitBuffer() )
    {
        cmd = usiTwiReceiveByte();
        switch(cmd)
        {
        case NOM_CONSTRUC :
            for (i=0; i<8; i++)
            {
                usiTwiTransmitByte(nom_capteur[i]);
            }
            break;

        case NOM_CAPTEUR :
            for (i=0; i<8; i++)
            {
                usiTwiTransmitByte(sensor_type[i]);
            }
            break;

        case VALEUR_DEMAN :
            usiTwiTransmitByte(recVal/*recVal*/);
            break;

        case BASIC_MODE :
            usiTwiTransmitByte(recVal);
            break;
        }
    }
}
return 0;
}
```

- usiTwiSlaveInit(int8)
- usiTwiDataInTransmitBuffer()=bool
- usiTwiReceiveByte()=int8
- usiTransmitByte(int8)

Après avoir programmé l'ATTiny85 un dernier problème était perçu. La valeur lue par la brique est uniquement 255. Pour régler le problème l'idée était de réutiliser des fonctions de l'ancienne librairie, en effet, bien que moins optimisé, elle fonctionnait. De ce fait, nous garderons tous les avantages de la nouvelle librairie, mais avec la praticité de l'ancienne.

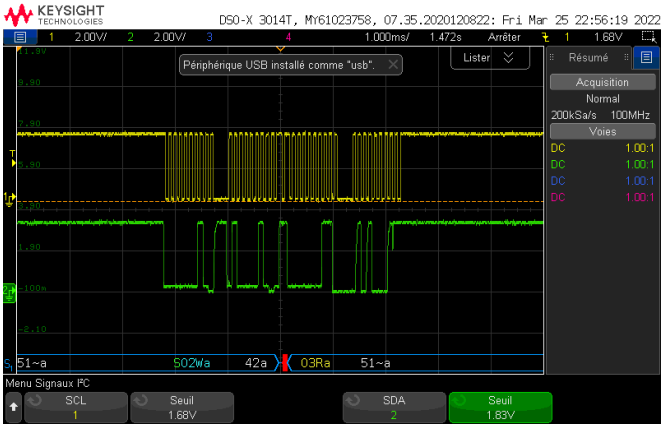
Malheureusement en modifiant usiTwiTransmitByte, cela n'a absolument rien changé. Après quelques temps en observant toutes les fonctions de la librairie, j'observe quelque chose de particulier :

```
391  bool usiTwiDataInTransmitBuffer(void)
392  {
393
394    // return 0 (false) if the receive buffer is empty
395    return txCount;
396
397  } // end usiTwiDataInTransmitBuffer
```

La fonction pour vérifier si le maitre appel l'esclave ne retourne que la variable txCount. Or cette variable est à 1 quand il y a une transmission et non une réception. Ce qui pose un problème car dans notre algorithme précédent nous demandons une donnée au maitre avant tout cela, donc une réception. En changeant la fonction par l'ancienne avec la modification des noms de variables correct nous sommes arrivés à cela :

```
unsigned char USI_TWI_Data_In_Receive_Buffer( void )
{
    unsigned char tmpRxTail;                    // Temporary variable to store volatile
    tmpRxTail = TWI_RxTail;                     // Not necessary, but prevents warnings
    return ( TWI_RxHead != tmpRxTail );         // Return 0 (FALSE) if the receive buffer is empty.
}
```

```
bool usiTwiDataInTransmitBuffer(void)       //Fonction modifié : affiche 255 constamment
{                       //Solution : creer une variable temporaire recuperant rxTail et la comparant avec rxHead

    unsigned char tmpRxTail;                // Creation d'une variable temporaire pour stocker rxTail au moment present
    tmpRxTail = rxTail;                     // on recupere rxTail dans la variable temporaire
    return ( rxHead != tmpRxTail );         // Renvoi 0 si aucune donnee n'est envoyee

  //  renvoi 0 mais perturbe la transmission
  //return txCount;

} // end usiTwiDataInTransmitBuffer
```



Après ce changement nous retrouvons les bonnes données, cela a donc fonctionné sans problème ! Bien que la liaison I2C a été grandement amélioré. La conversion analogique numérique ne fonctionne plus et les délais sur les Leds posent toujours un problème. Il faut donc réécrire tout cela.

13. Conversion sous Interruptions

Le seul moyen pour permettre à la conversion de se faire tout en gardant la liaison I2C est de gérer cela avec des interruptions. En effet, le problème est que le microcontrôleur ne peut faire qu'une seule tâche à la fois prédéfinie. Il faut donc choisir le bon moment pour pouvoir faire cette CAN. Il faut donc le faire après chaque communication I2C.

Table 17-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Comme nous avons maintenant 16 MHz, pour nous rapprocher d'en dessous de 200 kHz il faut un facteur de division bien supérieur, soit 128 pour obtenir 125 kHz :

```
void ADC_init(void)
{
    DDRB &=~(1<<PB3);                                   //Pin B3 en entree
    ADMUX |= (1<<MUX1)|(1<<MUX0)|(1<<ADLAR);            //MUX = 0011 donc entree PB3 et ADLAR a 1 pour avoir les bits de poids fort en premier
    ADCSRA|= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADIE);      //ADEN conversion autorisee
                                                        //On divise 16Mhz par 128 (125 KHz) < 200 KHz
                                                        //Interruptions autorisees
    ADCSRB &= ~(0b111<<ADTS0);                          //free-running mode
    sei();
};
```

Pour que cette interruption intervienne convenablement il faut la désactiver à chaque fois que la liaison USI/I2C commence. En effet les interruptions fonctionnent dans un ordre particulier sur l'ATTiny85 et si nous laissons cette interruption se faire constamment, la liaison I2C ne pourra plus être réalisée, soit :

Table 9-1. Reset and Interrupt Vectors

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	PCINT0	Pin Change Interrupt Request 0
4	0x0003	TIMER1_COMPA	Timer/Counter1 Compare Match A
5	0x0004	TIMER1_OVF	Timer/Counter1 Overflow
6	0x0005	TIMER0_OVF	Timer/Counter0 Overflow
7	0x0006	EE_RDY	EEPROM Ready
8	0x0007	ANA_COMP	Analog Comparator
9	0x0008	ADC	ADC Conversion Complete
10	0x0009	TIMER1_COMPB	Timer/Counter1 Compare Match B
11	0x000A	TIMER0_COMPA	Timer/Counter0 Compare Match A
12	0x000B	TIMER0_COMPB	Timer/Counter0 Compare Match B
13	0x000C	WDT	Watchdog Time-out
14	0x000D	USI_START	USI START
15	0x000E	USI_OVF	USI Overflow

Comme nous pouvons le voir l'USI possède une des priorités les plus basses (14), alors que ADC lui possède une priorité de 9, il faut donc à chaque interruption de l'USI désactiver l'interruption pour ADC pour pouvoir réaliser la liaison I2C correctement :

```
443  /*************************************************************************
444
445                          USI Start Condition ISR
446
447  *************************************************************************/
448
449  ISR( USI_START_VECTOR )
450  {
451    ADCSRA = ADCSRA & ~( (1<<ADIE)|(1<<ADEN) );
452    uint8_t usi_pins;
```

Pour effectuer un test nous devons connecter sur le port PB3 une tension variable et voir si les valeurs sont correctes. A l'aide de la liaison I2C que nous avons réalisé nous pourrons renvoyer la valeur de adcLed1 :

La fonction d'interruption étant déjà écrite au préalable dans la partie *Conversion Analogique Numérique* il est temps d'améliorer cette fonction pour obtenir une valeur moyenne.

14. Valeur moyenne d'une tension

Le fonctionnement du capteur est donc comme Charles l'a expliqué de renvoyer le signal de la LED s'il y a un obstacle, et sinon de renvoyer une tension continue. Ce qui va nous permettre de savoir si nous avons une tension ou une autre est la valeur moyenne :

En effet, si nous recevons un signal carré allant de 5-0V nous aurons une valeur moyenne de 2.5 V, alors que pour un signal continu, notre valeur moyenne sera de 5V, il faut donc créer un algorithme lors de la conversion récupérant la valeur moyenne de la tension :

```
ISR (ADC_vect){
    adcled1=measureLed1();                  //on récupère la première mesure
    adcled2=measureLed2();                  //on récupère la seconde mesure
    moyLed1=moyLed1 + adcled1;              //
    moyLed2=moyLed2 + adcled2;              //On additionne 2000 échantillons
    iMoy++;                                 //dans des nouvelles variables
    if(iMoy >= 2000)                        //
    {
        moyLed1=moyLed1/iMoy;               //On réalise la moyenne des 2000
        moyLed2=moyLed2/iMoy;               //échantillons
        recVal1=(int)moyLed1;               //Pour optimiser le temps de calcul
        recVal2=(int)moyLed2;               //les floats sont convertis en INT
        recVal=compareLed(recVal1,recVal2); //On introduit les deux valeurs analogiques dans l'algorithme
        moyLed1=0;                          //
        moyLed2=0;                          //Les moyennes sont raz
        iMoy=0;                             //
    }
    ADCSRA |= (1<<ADSC);                    //On relance une conversion
}
```

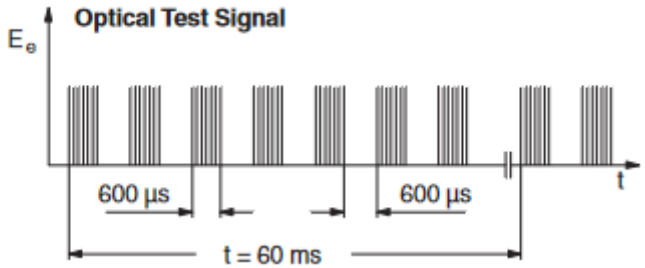
Le test est donc très convenable : Nous obtenons une tension moyenne de 2.45 V (125) quand un signal carré de 50% est envoyé. Sinon dans le cas d'un signal continu, nous recevons une tension moyenne de 4.9 V (249).

C'est donc très convenable et nous pourrons l'utiliser avec l'algorithme

15. LED sous Interruptions et Algorithme

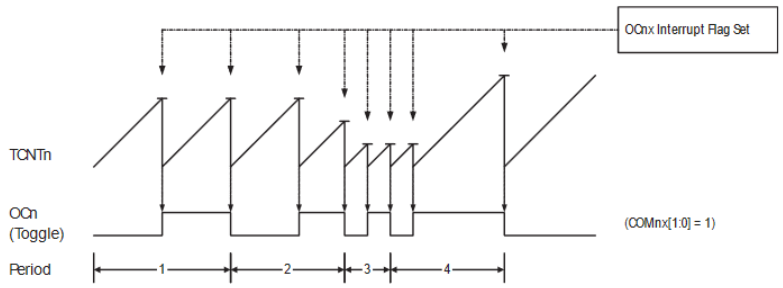
L'utilisation précédente de la fonction delay pose un problème bien connu : le processeur est mis en pose quoi qu'il arrive, temps que la fonction n'a pas fini. L'idée est donc d'utiliser un timer sous interruption, de cette manière la liaison I2C se fera à tout moment, tout comme la conversion analogique numérique.

Bien que dans la dernière partie sur les Leds nous avions pris 500 ms, ce n'est pas réaliste, d'après la documentation du TSOP4830, il est demandé d'avoir un temps entre chaque impulsion des Leds de 600 us, soit :



Pour ce faire, nous allons cette fois-ci utiliser sur le timer0 le mode normal, soit : quand 0xFF est atteint par le compteur (256 en décimal), il retourne à 0 directement :

Figure 11-7. CTC Mode, Timing Diagram



A chaque fois qu'il atteint 256, une impulsion sous forme d'interruption ISR est donné dans le microcontrôleur, il a donc fallu changer légèrement la formule précédente, de tel sorte à ne pas compter en double, car ici il n'est plus question de fréquence sur un état haut et bas, mais simplement d'un temps entre chaque changement d'état, soit :

$$\frac{prediv \cdot (256 + 1)}{f_{CPU}} = T_{CTC}$$

Ici nous cherchons donc à être simplement inférieur à 600 us, mais avec le prediv le plus bas possible, nous nous retrouvons avec un prediv de 8 que nous multiplierons par un facteur (ce facteur sera expliqué plus tard dans le programme ISR), soit :

$$\frac{8 \cdot (256 + 1)}{16 \times 10^6} \times 5 = 643us$$

Il nous faut maintenant réaliser la fonction permettant l'initialisation de ce timer, donc rappelons certains points :

- Nous ne voulons pas renvoyer le signal sur une pin
- Nous voulons une interruption sur chaque front
- Nous voulons une division par 8
- Nous voulons le mode de comparaison en normal
- 5 étant le nombre d'itérations pour atteindre le temps voulu

Soit :

Table 11-2. Compare Output Mode, non-PWM Mode

COM0A1 COM0B1	COM0A0 COM0B0	Description
0	0	Normal port operation, OC0A/OC0B disconnected.
0	1	Toggle OC0A/OC0B on Compare Match
1	0	Clear OC0A/OC0B on Compare Match
1	1	Set OC0A/OC0B on Compare Match

Table 11-5. Waveform Generation Mode Bit Description

Mode	WGM 02	WGM 01	WGM 00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX(1)
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM(2)
2	0	1	0	CTC	OCRA	Immediate	MAX(1)
3	0	1	1	Fast PWM	0xFF	BOTTOM(2)	MAX(1)
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM(2)
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM(2)	TOP

Table 11-6. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

11.9.7 TIMSK – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x39	–	OCIE1A	OCIE1B	OCIE0A	OCIE0B	TOIE1	TOIE0	–	TIMSK
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

Les seuls bits à mettre à 1 sont donc : CS01 pour avoir une division par 8. Et TOIE0 pour activer l'interruption pour le timer0.

```
45    void timer0_init(void)
46    {
47        TCCR0A=0x00;            //Mode normal
48        TCCR0B=0x00;
49        TCCR0B |= (1<<CS01);    //divise par 8
50        TCNT0=0;
51        TIMSK|=(1<<TOIE0);      //Interruptions autorisees
52    }
```

Nous développons par la suite la fonction ISR liée au timer0 :

```
54  ISR (TIMER0_OVF_vect)
55  {
56      if (intr_count==5)                  //on attends 5 fois pour avoir
57      {
58          timer_flag=1-timer_flag;
59          intr_count=0;                   //raz pour recommencer
60      }
61      else  intr_count++;
62  }
```

Il faut enfin réécrire la manière dont les mesures sont prises par rapport au LED (comment les allumer puis les éteindre) :

```
51  char measureLed1(void)
52  {
53      if(timer_flag == 1)         //Quand l'interruption occure
54      {
55          if(led1_state==0)       //Quand la LED est éteinte
56          {
57              led1_init();        //on l'allume
58              ledresult1=ADCH;    //on récupère la tension actuelle
59          }
60          else                    //Quand la LED est allumée
61          {
62              led1_stop();        //on l'éteint
63          }
64      }
65      return ledresult1;          //on renvoi la valeur de la tension
66  };
```

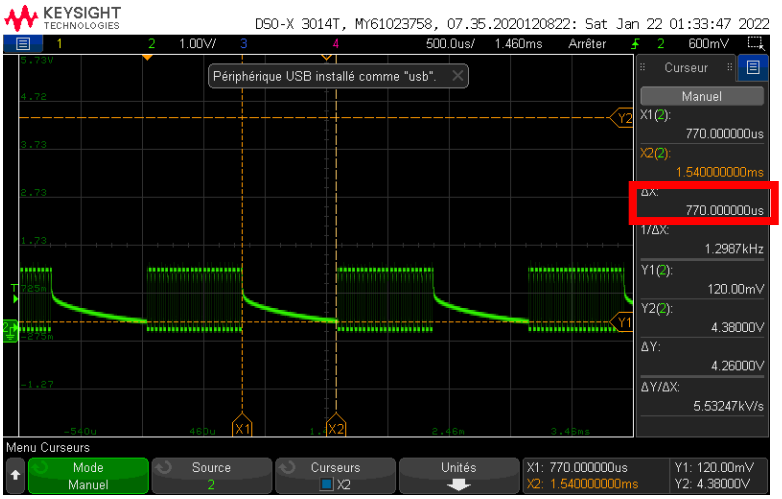
NB : timer_flag et led1_state sont des volatiles car ce sont des valeurs qui doivent exister en dehors de leurs fonctions respectives

On définit donc la variable led1_state à l'allumage -> 1, et à l'éteignage-> 0

```c
void led1_init(void)
{
    TCCR1|=(1<<CTC1)|(1<<COM1A0);       //on active le mode CTC en comparant OCR1C, COM1A0 : mode toggle
    OCR1C=35;                           //d'apres retroingenieurie OCR1C=35
    TCCR1|=(1<<CS10)|(1<<CS11);         //on active la clock, prediv de 4
    DDRB |= (1<<PB1);
    led1_state=1;
}
```

```c
void led1_stop(void)
{
    DDRB &= ~(1<<PB1);
    TCCR1 &= ~(1<<COM1A0);
    led1_state=0;
}
```

Comme vu dans l'ISR de la CAN, ces fonctions sont appelées à chaque mesure, les LEDS vont donc osciller entre l'état haut et l'état bas tous les 643 us :



Au moment de prendre la capture la valeur 5 pour le timer_flag n'était pas encore déterminée, ce screen a donc été fait avec 6.

L'algorithme a donc été aussi changé en fonction des analyses réalisées sur le capteur infrarouge, le voici :

```c
unsigned char compareLed(int led1,int led2)
{
    int adcCompare = led1 - led2;
    if(adcCompare < 1)
    {
        if(led1 > 204)
        {
            return NOTHING;
        }
        else
        {
            return FRONT;
        }
    }
    if (led1 > led2)    // adcLed2 > adcLed1
    {
        return LEFT;
    }
    else
    {
        return RIGHT;
    }
};
```

Un test assez simple avec un oscilloscope permet de tester ses fonctionnalités :

Dans le cas d'un signal carré sur l'entrée PB3 la brique Lego renverra FRONT (81)

Dans le cas d'un signal continu à 5V sur l'entrée PB3, la brique renverra NOTHING (84)

Il a été aussi décidé de changer la fréquence de ces Leds, en effet, nous ne possédions plus de TSOP4830, nous sommes à présent passé sur un TSOP34856, et qui, d'après la datasheet ne résonne pas à la même fréquence :

Les pins, elles ne changent pas, ce qui ne pose aucun problème pour la carte.

$$\frac{f_{CPU}}{2 \cdot prediv \cdot (OCR1C + 1)} = f_{CTC}$$

$$\frac{16 \times 10^6}{2 \cdot 4 \cdot (35 + 1)} = 55.5\ kHz$$

Cette valeur sera légèrement plus grande en pratique, ce qui est parfait.

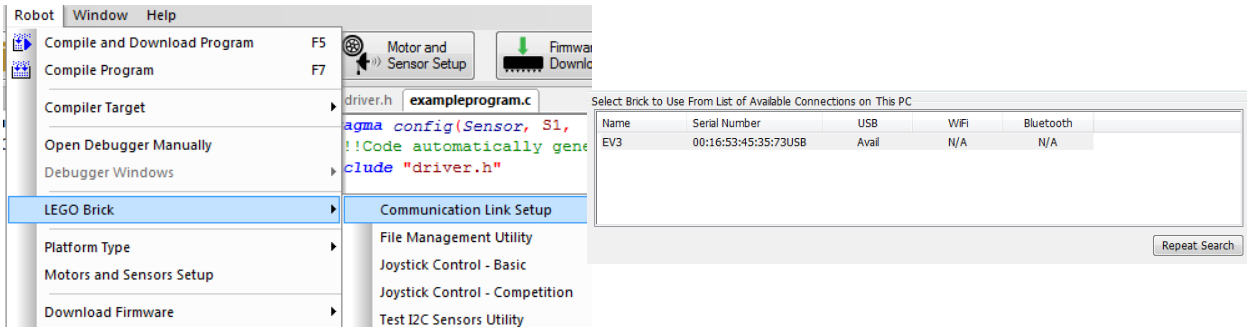
16. Programme Final

```
unsigned char adcled1;
unsigned char adcled2;
double moyLed1=0;
double moyLed2=0;
int iMoy=0;
volatile unsigned char recVal1=0;
volatile unsigned char recVal2=0;
volatile unsigned char recVal=0;

ISR (ADC_vect){
    adcled1=measureLed1();                  //on récupère la première mesure
    adcled2=measureLed2();                  //on récupère la seconde mesure
    moyLed1=moyLed1 + adcled1;              //
    moyLed2=moyLed2 + adcled2;              //On additionne 2000 échantillons
    iMoy++;                                 //dans des nouvelles variables
    if(iMoy >= 2000)                        //
    {
        moyLed1=moyLed1/iMoy;               //On réalise la moyenne des 2000
        moyLed2=moyLed2/iMoy;               //échantillons
        recVal1=(int)moyLed1;               //Pour optimiser le temps de calcul
        recVal2=(int)moyLed2;               //les floats sont convertis en INT
        recVal=compareLed(recVal1,recVal2); //On introduit les deux valeurs analogiques dans l'algorithme
        moyLed1=0;                          //
        moyLed2=0;                          //Les moyennes sont raz
        iMoy=0;                             //
    }
    ADCSRA |= (1<<ADSC);                    //On relance une conversion
}

ISR (TIMER0_OVF_vect)
{
    if (intr_count==5)                      //on attends 5 fois pour avoir
    {
        timer_flag=1-timer_flag;
        intr_count=0;                       //raz pour recommencer
    }
    else  intr_count++;
}
```

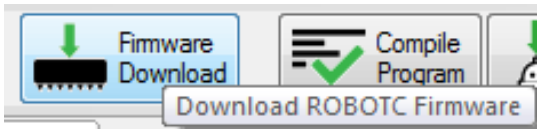
```c
int main( void )
{
    unsigned char cmd,i=0;
    unsigned char nom_capteur[8]= {'C','a','t','c','h','E','y','e'};
    unsigned char sensor_type[8]= {'I','R','O','b','s','t','a','c'};
    uint8_t slaveAdress;//data = 0x45;
    slaveAdress = 0x01;
    MCUCR |= (1<<PUD);

    usiTwiSlaveInit(slaveAdress);

    sei();
    ADC_init();
    ADC_start_conversion();
    timer0_init();

    while(1)
    {
        if( usiTwiDataInTransmitBuffer() )
        {
            cmd = usiTwiReceiveByte();
            switch(cmd)
            {
            case NOM_CONSTRUC :
                for (i=0; i<8; i++)
                {
                    usiTwiTransmitByte(nom_capteur[i]);
                }
                break;

            case NOM_CAPTEUR :
                for (i=0; i<8; i++)
                {
                    usiTwiTransmitByte(sensor_type[i]);
                }
                break;

            case VALEUR_DEMAN :
                usiTwiTransmitByte(recVal/*recVal*/);
                break;

            case BASIC_MODE :
                usiTwiTransmitByte(recVal);
                break;
            }
        }
    }
    return 0;
```

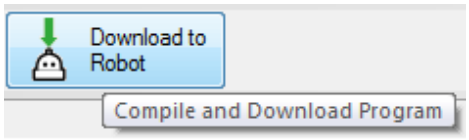
17. Librairie Robot C

La première chose à réaliser est de programmer la brique lego pour avoir le firmware personnalisé de RobotC, pour se faire il faut tout d'abord vérifier la bonne connexion entre la brique et l'ordinateur :



Nous retrouvons bien notre brique Lego, la dernière étape arrive et consiste à simplement télécharger le firmware RobotC dans la brique (attention, il faut que la version de l'OS de la brique lego soit 107) :





Pour créer une librairie robot C il suffit de créer différentes fonctions permettant à l'utilisateur de recevoir l'information voulue. Dans notre cas il nous suffit de renvoyer l'orientation selon laquelle l'obstacle est orienté.

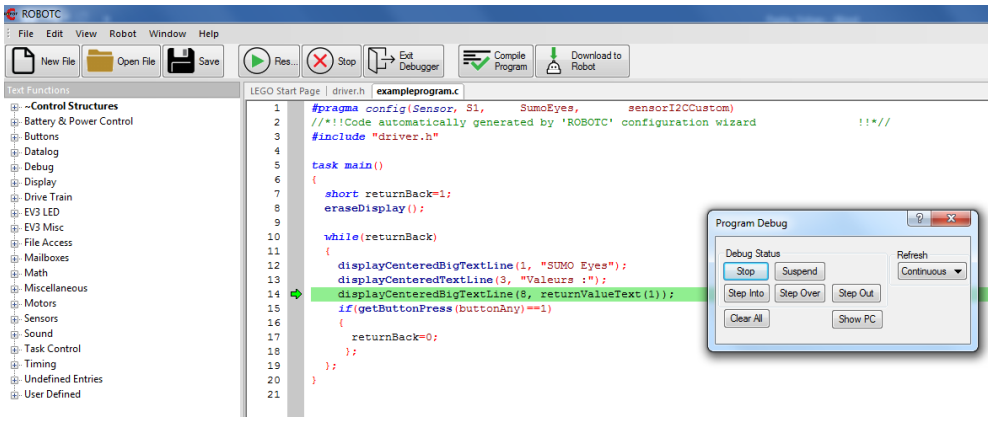
Comme expliqué dans le programme, la valeur retournée par la brique avec le capteur va de 81 à 84, avec 82 étant à gauche et 83 à droite. Voici la librairie avec le programme d'exemple :

```
#pragma config(Sensor, S1,     SumoEyes,       sensorI2CCustom)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
#include "driver.h"

task main()
{
    short returnBack=1;
    eraseDisplay();

    while(returnBack)
    {
        displayCenteredBigTextLine(1, "SUMO Eyes");
        displayCenteredTextLine(3, "Valeurs :");
        displayCenteredBigTextLine(8, returnValueText(1));
        if(getButtonPress(buttonAny)==1)
        {
            returnBack=0;
        };
    };
}
```

```c
#pragma systemFile

#ifndef DRIVER_H_INCLUDED
#define DRIVER_H_INCLUDED

#endif // DRIVER_H_INCLUDED

char returnValueSumo(int sortie)
{
    short zone;
    switch(sortie)
    {
        case 1 :
            zone = SensorValue[S1];break;
        case 2 :
            zone = SensorValue[S2];break;
        case 3 :
            zone = SensorValue[S3];break;
        case 4 :
            zone = SensorValue[S4];break;
        default:
            return -1; break;
    }
    switch (zone)
    {
        case 81: return zone;break;
        case 82: return zone;break;
        case 83: return zone;break;
        case 84: return zone;break;
    }
    return 0;
}

char* returnValueText(int sortie)
{
    char zone = returnValueSumo(sortie);
    switch(zone)
    {
        case 81: return "FRONT";break;
        case 82: return "LEFT";break;
        case 83: return "RIGHT";break;
        case 84: return "NONE";break;
    }
    return "ERROR";
}
```

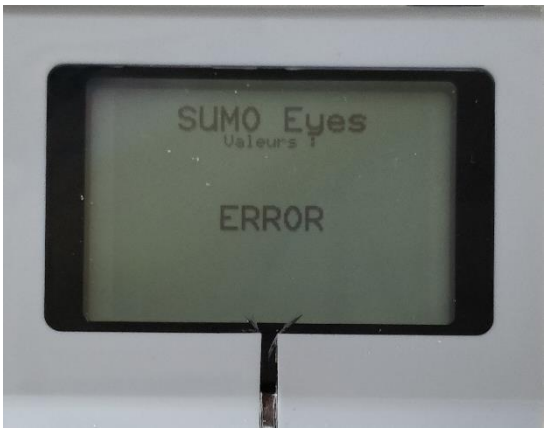
Les deux fonctions demandent l'entrée sur laquelle le capteur est placé, et renvoie soit un caractère, ou une chaîne de caractère sous forme de pointeur.

Le test s'effectue donc comme ceci :

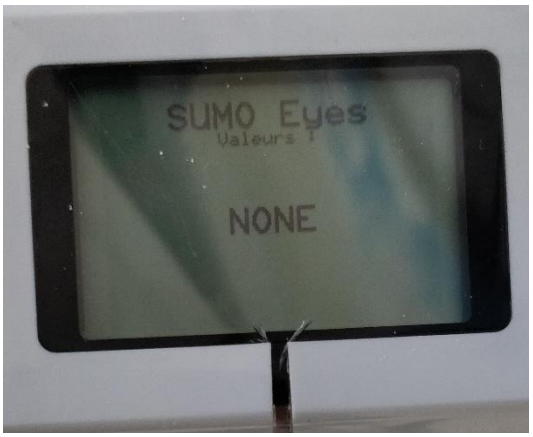


Et nous avons testé trois états :

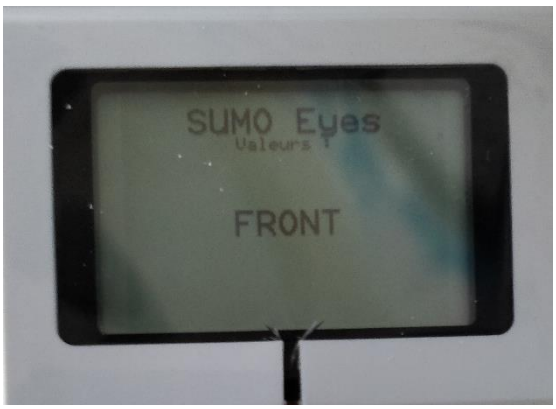
- Non connecté



- Aucun obstacle



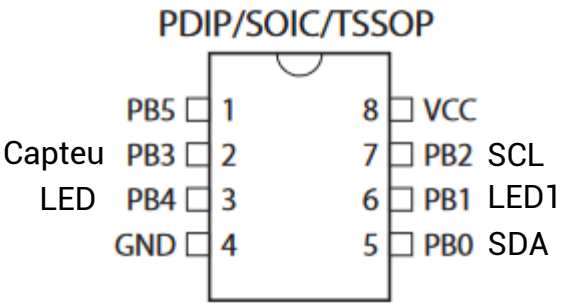
- Obstacle devant



Les résultats sont donc très satisfaisants et affiche les bonnes valeurs pour chaque réaction. Dans le premier cas la brique n'était connectée à aucun capteur, dans le second nous avions une tension continue de 5V et enfin un signal carré de 5 à 0V. Pour tester la droite et la gauche il faudrait le faire sur le vrai capteur.

Conclusion

Pour conclure sur la partie programmation et communication avec la brique lego les tests effectués sont très satisfaisants. Finalement nous avons plusieurs éléments fonctionnels sur notre microcontrôleur, définis par ce schéma :



Les fonctions sont donc les suivantes :

- Communication I2C à l'aide des ports PB2 et PB0 et de résistances de pull-up de 82 kΩ chacune
- Temporisation de deux LED à 600 µs et une fréquence de 55 kHz sur PB4 et PB1
- Conversion analogique sur PB3 avec moyennage sur 2000 échantillons

Tout cela est adapté pour être compris par la brique Lego. Au final je me retrouve avec une plaque Labdec ainsi qu'un programme fonctionnel sur le microcontrôleur. Le grand problème sur ce projet était la communication, en effet, étant trop concentré sur ma partie je n'ai pas parlé de mes problématiques à mes camarades. Notamment les résistances de pull-up qui ont posé un problème lors de l'impression de la carte. Durant les dernières séances nous nous sommes focalisés sur comment faire fonctionner la carte mais sans réel succès. Dans un premier cas le courant était trop élevé et le capteur envoyait donc une mauvaise valeur au microcontrôleur, et dans un second cas (avec une résistance à la source de tension) le courant était trop faible et ne faisait pas fonctionner correctement le microcontrôleur.

Le projet est donc complet à quelques détails. Le programme est fonctionnel sur une plaque de test. Le schéma électrique est théoriquement correct. Et enfin le cartel de la carte lui est correctement dimensionné. Le grand soucis est l'assemblage des différentes parties qui est sûrement dû à un manque de communication de l'équipe.