TRIMBUR Marius
TOMBAZZI Loïc
IMA4 SC

Projet n°39 - IMA4 - Projecteur Laser

Introduction

Dans le cadre de notre quatrième année d'ingénieur nous avons eu l'opportunité de réaliser un projet sur une longue période. Pour notre part, nous avons choisis de fabriquer un projecteur laser. En effet, nous avons mis en place une interface web capable de projeter une image par l'intermédiaire d'un laser et de galvanomètres. Ce projet a été divisé en différentes parties avec la mise en place de l'interface web, la réalisation du système embarqué, la création d'un serveur websocket et enfin la partie projection.

I - Présentation générale du projet

Le projet consiste à réaliser avec des lasers des figures géométriques par l'intermédiaire d'un système galvanométrique.

Notre premier travail fut de mettre en place un serveur WebSocket hébergé sur la RaspberryPi pour servir d'interface entre l'application web et l'arduino qui est connecté via une liaison série sur la RaspberryPi. La programmation des formes à effectuer a été réalisée par l'intermédiaire d'une application web hébergée, elle aussi, sur la RaspberryPi qui permet de récupérer une image, d'y appliquer divers traitements et filtres afin d'en extraire les contours, et de calculer les commandes pour positionner les galvanomètres.

Ensuite nous nous sommes intéressés à l'utilisation d'un système embarqué pour contrôler les galvanomètres et donc dessiner avec le laser sur un plan 2D. En effet, le laser reste toujours fixe, seuls les galvanomètres, et donc les miroirs bougent. Les galvanomètres étant contrôlés en courant il a fallu réaliser une interface électronique pour convertir une commande en tension en commande de courant. La commande en tension a été assurée par une carte Arduino.

Enfin il a fallu concevoir la partie projection. Pour cela, nous avons dû aligner le laser et les galvanomètres. En effet, les miroirs changeront d'inclinaison suivant le courant traversant les galvanomètres. Il nous a fallu également créer un système d'alimentation qui à partir d'une même source (réseau 230 V) est capable d'alimenter les lasers et les galvanomètres ainsi qu'une Arduino et une Raspberry Pi.

II-Conception du projecteur

Bien qu'initialement nous avions commandé un boitier ABS afin d'isoler uniquement les composants d'alimentation, nous avons finalement constaté qu'avec un peu de réflexion nous pouvions placer l'intégralité des composants à l'intérieur. Le projecteur terminé est donc constitué d'un boîtier 60*240*120 mm avec une LED indicatrice de mise en tension, un connecteur C14 pour une alimentation 230V et d'une ouverture ovale muni d'un film dissipateur afin de limiter l'intensité du rayon laser.

II.1. Matériel

Une partie du matériel nous as été directement remis au commencement du projet, le reste à été commandé par nos encadrants, quelques pièces proviennent également de stocks personnels.

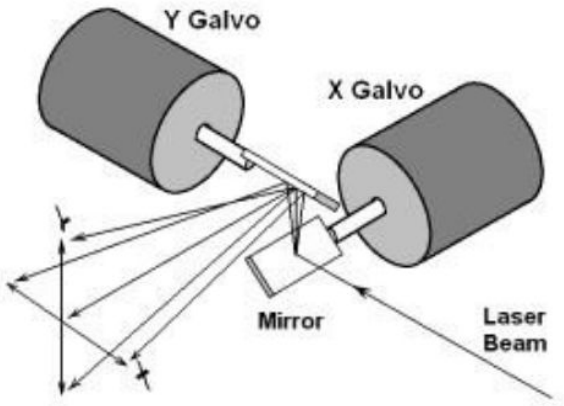
Nous ferons ici une liste exhaustive du matériel qui a été utilisé et leur utilité dans notre projet :

- Raspberry Pi : la raspberry pi, qui à été configurée en point d'accès Wifi, permet à l'utilisateur d'accéder à la page web (hébergée avec un serveur Apache2) qui permet de contrôler le projecteur. Elle héberge également un serveur WebSocket qui permet de récupérer une image de l'utilisateur et d'en détecter les contours. En outre, elle permet d'envoyer des commande à l'Arduino Pro Mini via une liaison I2C.

- Arduino Pro Mini : ATMega 328p version 16 Mhz et 5V, elle nous permet d'envoyer des commandes de position aux contrôleurs de galvanomètres grâce à ses sorties PWM. Initialement, elle n'était pas prévu dans le projet mais le fait qu'une seule sortie PWM hardware était disponible sur la Raspberry Pi nous l'as imposé.

- Pololu md08a : basé sur un pont hacheur TB6612FNG, ce composant nous permet de contrôler les deux galvanomètres. En effet, grâce à deux entrées PWM provenant de l'Arduino, elle délivre deux signaux analogiques réversibles en tension (dans notre cas +5V -5V).

- Galvanomètres à miroir (x2) : les galvanomètres sont des systèmes couramment utilisé dans certains ampèremètre analogiques. Ils sont généralement constitué d'une bobine de cuivre (ou deux) avec au centre une partie mobile alimentée. Le passage d'un courant dans la bobine induit un champ magnétique comme décrit d'après les lois fondamentales de l'électromagnétique et va ainsi permettre de mettre en rotation la partie aimantée. Dans notre cas, sur cette partie mobile, se trouvent à l'extrémité de chaque galvanomètres un miroir dont nous pouvons contrôler la position. Ceci va nous permettre de réfléchir le laser selon deux directions paramétrables X et Y. L'avantage de ce dispositif est sa rapidité, les galvanomètres que nous utilisons sont capables de commuter 20 000 fois par seconde ce qui permet de faire mouvoir le rayon laser de façon avoir l'impression d'obtenir une trace grâce à la persistance rétinienne.



- Contrôleurs de galvanomètres (x2) : les galvanomètres étant commandés en courant, nous utilisons ces contrôleurs afin de se ramener à une commande en tension. Ces contrôleurs sont alimentés en +/- 15V et les pins de contrôle supportent une tension entre -5 et 10V. Par précaution, nous utilisons du -/+ 5V ce qui limite très légèrement l'amplitude maximale de la position du laser

- Diode laser : nous avons récupéré sur un projecteur laser de show une diode émettrice laser de couleur vert. Cet émetteur appartient à la classe AB des lasers, il est donc à manipuler avec précautions.

- Alimentations : afin d'alimenter les différents composants nous avons recours à plusieurs alimentations dont :
    - Une alimentation 230V/15V pour les contrôleurs de galvanomètres
    - Une alimentation pour le laser récupérée dans le projecteur laser de show
    - Une alimentation 230V/5V afin d'alimenter la Raspberry pi et l'Arduino

Ces trois alimentations sont connectées entre elles sur le secteur 230V par l'intermédiaire d'un connecteur domino.

- Boitier ABS 60*240*120 mm

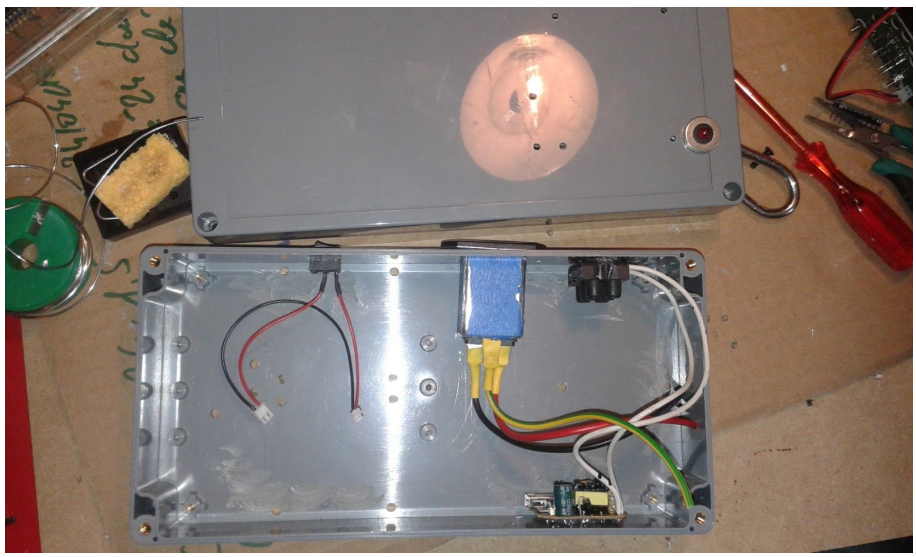
- Autre : a cette liste s'ajoutent quelques composants comme un connecteur C14 mâle afin d'alimenter le boîtier en 230V, une LED rouge indicatrice de mise en tension, un switch à bascule pour allumer ou éteindre le laser et de la quincaillerie (vis, equerre…).

II.2. Réalisation

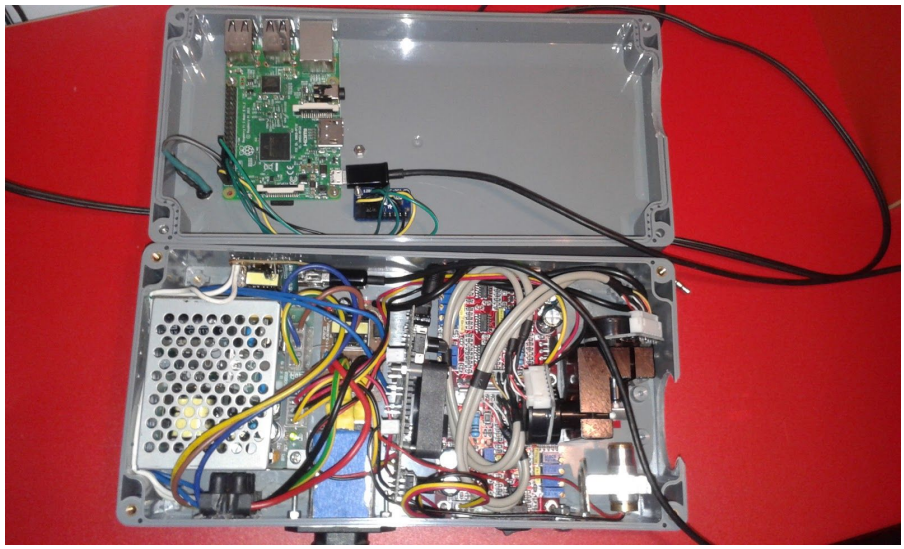
La réalisation du boitier s'est avérée assez complexe, nous avions tout juste l'espace disponible pour y placer tous les composants.

Une longue première étape a consisté à tester différents placements des composants afin de trouver la meilleure optimisation. Cette étape nous a pris un certain temps, mais elle nous a permit d'éviter des erreurs qui aurait pu être difficilement corrigés si nous avions commencé directement le montage.

Une fois les composants disposés on procède au traçage, puis au perçage des trous pour fixer les composants. Les trous les plus difficiles à réaliser étaient ceux pour le connecteur C14, le switch à bascule ainsi que pour l'ouverture du laser. Certains composants qui n'avaient de trous pour des vis ont été collés.

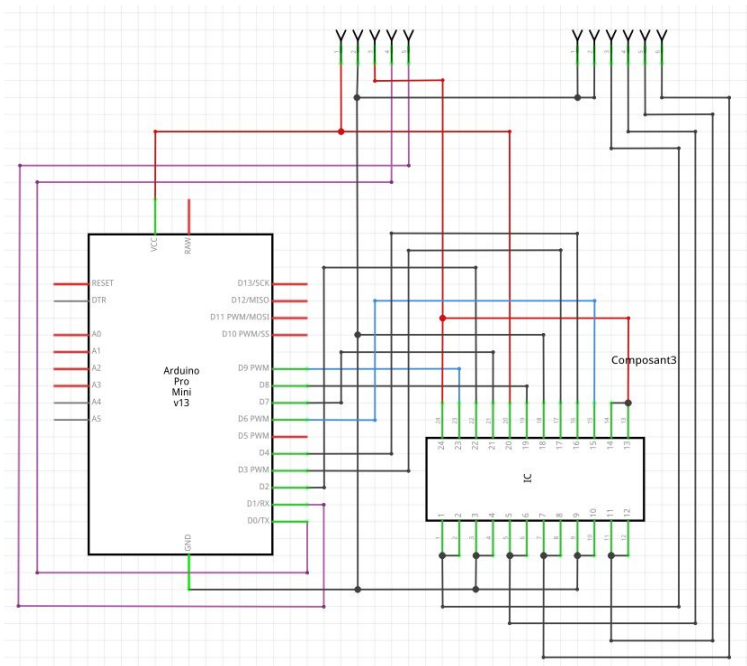


Une fois les trous percés, nous avons pu fixer toutes les pièces et faire quelques soudures.

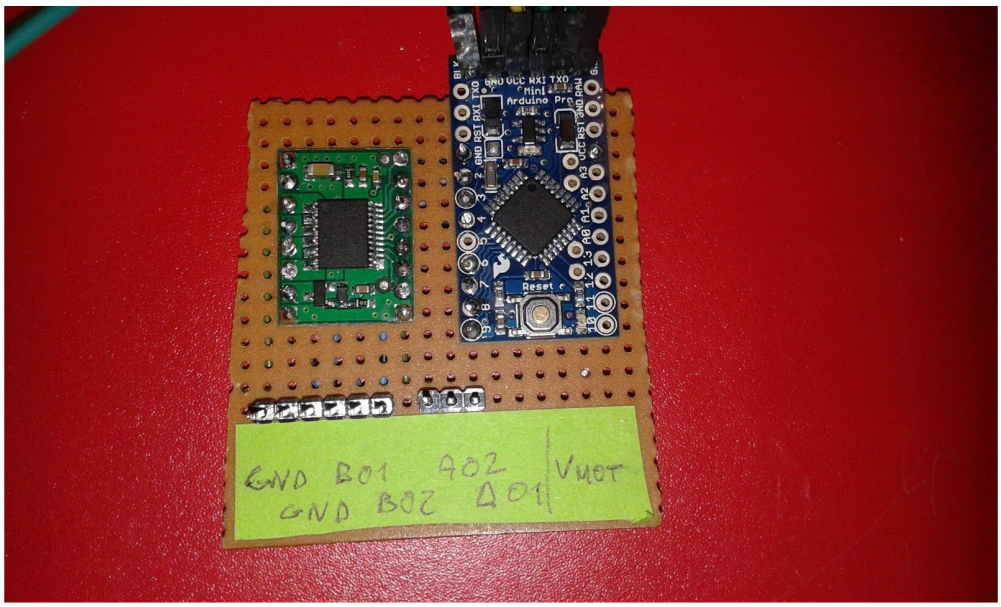


Afin de faciliter l'alignement minutieux du laser avec les galvanomètres, une vis permet de régler avec précision sa position.

Le pololu md08a nous a permit d'obtenir une tension relativement analogique (la PWM et le hachage provoquent un bruit non négligeable, spécialement avec une fréquence faible de PWM) et réversible en tension. Ce pololu comporte deux canaux pour contrôler deux moteurs. Sur chaque canal, on dispose d'une entrée PWM qui contrôle le hachage de la tension, deux entrées digitales qui permettent de contrôler la réversibilité en tension et une sortie analogique. Le pololu permet également d'imposer la tension de référence maximale qui sera hachée. Nous avons optés pour une tension de 5V comme imposé par les galvanomètres et également car il était aisé de récupérer cette tension par l'intermédiaire l'Arduino ou la Raspberry Pi. Voici le schéma de câblage réalisé avec Fritzing :



En premier lieu, nous avions choisi de dessiner le circuit PCB pour être ensuite gravé, mais des difficultés avec Fritzing et un manque de temps croissant nous ont plutôt orienté vers une réalisation avec une plaque de prototypage. Rétrospectivement, ce fut un bon choix car des modifications de dernière minute ont dû être effectuées ce qui n'aurait pas été possible avec une carte PCB. Voici la carte finalisée :



II.3. Difficultés rencontrés et précautions nécessaires

Un certain nombre de difficultés sont intervenues pendant la réalisation du projecteur. Tout d'abord une erreur d'étude nous as imposé un changement de composant. En effet, afin d'obtenir une tension analogique afin de commander les galvanomètres, nous avions besoin d'un CNA mais par erreur nous avons commandé un CAN. Nous nous sommes donc orienté vers l'utilisation du pololu md08a mais celui-ci étant commandé en PWM et le manque de sortie PWM hardware de la Raspberry Pi nous as imposé l'utilisation d'une Arduino. Heureusement, nous avions encore un peu d'espace disponible dans le boitier pour les ajouter.

Du au manque d'espace dans le boitier, il a fallu apporter un soin tout particulier à l'isolation des divers composants, le risque de contact entre deux composant étant très fort. Toutes les connexions sont donc isolées avec de la gaine thermo-rétractable et des vérification minutieuses ont été effectuées.

L'utilisation d'une source d'alimentation 230V a nécessité une prise de précautions toute particulière. Prenant en compte qu'avec cette tension, un courant de 100 mA suffit à provoquer un arrêt cardiaque chez un adulte, il a fallu limiter au maximum la manipulation du boîtier ouvert quand celui-ci était en tension.

En outre, le laser que nous utilisons appartient à la classe AB des lasers ce qui signifie qu'une exposition directe avec les yeux provoque instantanément des lésions définitives et une exposition indirecte (réflexion sur un mur) prolongée est également néfaste. Ces restrictions nous ont imposé une très grande prudence, notamment de ne pas manipuler le boîtier avec des objets potentiellement réfléchissant (comme une montre), l'utilisation de lunettes de protection adaptées sur les phases à risque comme l'alignement du laser. Néanmoins l'utilisation des lunettes diminuent drastiquement l'acuité visuelle, la perception des couleurs et occulte quasi-parfaitement le rayon laser, il n'a donc pas été aisé de procéder à cette phase d'alignement. En outre, nous avons constaté que même une exposition prolongée de la réflexion du laser contre un mur avec peinture mat devenait désagréable à la longue. Nous avons donc ajouté un film de filtrage des rayons (réalisé avec des lunettes 3D) afin de limiter l'intensité du rayon.

III-Partie client

L'interface web a été mise en place par l'intermédiaire de différents langages de programmation. Nous avons en effet, utilisé le code HTML pour la mise en page simple de l'application. Javascript pour la communication avec le serveur et pour l'utilisation dynamique de la page. Et enfin, l'utilisation de css pour un apport visuelle et esthétique de la page web.

III.1. Mise en page de l'interface web

Le code HTML (ou HyperText Markup Language) est un format de données conçu pour représenter les pages web. Celui-ci permet notamment de structurer sémantiquement et de mettre en forme le contenu des pages en incluant des ressources multimédias (images dans notre cas), des formulaires de saisie ou des programmes informatiques (Javascript).

Notre code HTML sert donc à créer les éléments d'interactions avec l'utilisateur et à les adapter sémantiquement. Nous avons notamment créé deux boutons de contrôle de la raspberry l'un pour l'éteindre « SHUTDOWN » et l'autre pour la rallumer « RESET ». Ces boutons nécessitent bien entendu une confirmation pour éviter tout problème.

Nous avons ensuite créé les points de communications avec le serveur par l'intermédiaire des boutons « Envoyer le texte » et « Envoyer l'image ». Ce premier bouton permet, comme son nom l'indique, d'envoyer le texte écrit dans le rectangle et le second envoie l'image qui a été téléversé par l'intermédiaire du bouton « Parcourir… » qui a été configuré pour n'accepter que des images (jpg, gif, png, etc.). Le fonctionnement technique de celui-ci sera expliqué dans la partie Javascript. Nous avons ajouté deux canvas, l'un pour l'affichage de l'image téléversée et l'autre pour le rendu de l'image traité par le serveur (voir ci-dessous). Enfin, nous avons rajouté une note de bas de page pour une brève présentation et un lien vers le wiki de notre projet.

Le code CSS (ou Feuilles de style en cascade) est quant à lui un langage informatique qui décrit la présentation du code HTML. Celui-ci permet d'améliorer le rendu visuel d'une page web (fond d'écran, position des éléments sur la page, police de caractères). Voici notre interface web avec et sans le code CSS :

Avec CSS

Sans CSS

III.2. Communication Client - Serveur

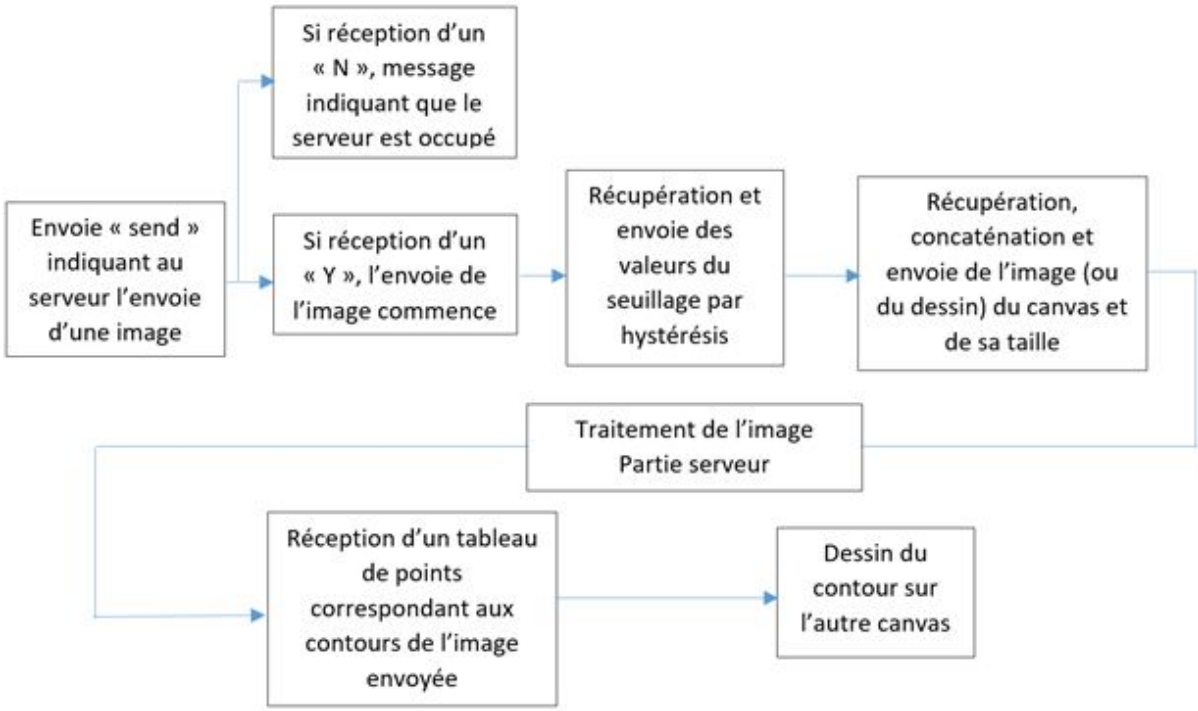
Le JavaScript est un langage de programmation orienté objet principalement employés dans les pages web interactives. Ce langage sert généralement à contrôler les données saisies dans des formulaires HTML, ou à interagir avec celui-ci, il est aussi utilisé pour réaliser des services dynamiques.

Nous avons décomposé notre code Javascript en deux parties distinctes. La première partie *« parcourir.js »* permet le téléversement d'images sur la page web, la possibilité de dessiner sur le canvas de gauche (possibilité de faire les deux en même temps) ainsi que de régler le seuillage par hystérésis de l'image (Ceci permet d'influencer le traitement de l'image en affichant plus ou moins de contours). Et la seconde partie, *« communication.js »*, s'occupe de la communication entre le client et le serveur (envoie et traitement des messages).

La partie communication de l'interface web va donc faire la plus grande partie du travail de l'interface web. En effet, celui vérifiera à intervalle régulier la connexion avec le serveur et en cas d'erreur, affichera une fenêtre d'alerte. En cas de bon fonctionnement, le client pourra donc communiquer avec le serveur. Il existe différentes communication possible entre les deux parties qui sont :

- Réception d'un message d'erreur provenant du serveur
- Envoie d'une requête d'extinction ou de redémarrage
- Envoie d'un message
- Envoie de l'image avec la communication qui en découle (voir partie *IV.2.3. Notre protocole*)

La fonction envoie de l'image suit un protocole bien défini, selon ces étapes :



IV-Partie serveur

IV.1. Configuration point d'accès wifi, Apache

Pour la réalisation du projet nous avons commencé par initialiser le port série de la raspberryPI afin de récupérer les différents logiciels nécessaires à la réalisation du projecteur laser (Apache2, dnsmasq, hostapd, bind9, etc). La mise en place du serveur apache fût réalisé en configurant les fichiers */etc/network/interfaces* et */etc/resolv.conf* qui permettent de définir une adresse statique, le netmask, le Gateway ainsi que l'adresse du serveur.

Delà nous avons pu configurer le point d'accès wifi de la raspberryPI avec la mise en place d'une adresse IP statique et la désactivation de la négociation automatique de bail. Puis, en configurant le logiciel *hostapd* qui s'occupe de gérer le point d'accès. Ce logiciel permet de définir différents points dont l'interface wifi, le nom du réseau, le canal à utiliser ou encore le mot de passe du réseau. De plus, ce logiciel permet d'activer le démarrage automatique du point d'accès wifi lors de la mise sous tension de la raspberry. Nous avons terminé cette mise en place du point d'accès par la configuration du serveur DNS via le logiciel *dnsmasq*. Finalement, la page d'accueil de l'interface web est donc disponible à l'adresse 192.168.3.1 en se connectant au réseau *Projecteur-Laser* de la raspberryPI.

IV.2. Serveur WebSocket

Afin de mettre en place une communication entre l'utilisateur et le projecteur, nous avons opté pour le protocole WebSocket. Ce protocole va nous permettre de transmettre des commande simples comme "Projeter" ou "Stop", mais va également nous permettre de transmettre un grand nombre de données en transmettant des images.

IV.2.1. Présentation du protocole WebSocket

Le terme WebSocket désigne à la fois une API et un protocole. Le protocole WebSocket est une surcouche au protocole HTTP lui-même basé sur le protocole TCP.
Il permet d'établir une connexion bidirectionnelle full duplex (envoi et émission simultanés) entre un client et un serveur. Son avantage par rapport au protocole HTTP est qu'une fois la connexion établie, le client et serveur peuvent dialoguer sans devoir effectuer toute la procédure lourde du protocole HTTP d'envoi de requête puis d'attente de réponse ce qui permet d'obtenir une communication quasi temps réel.

La mise en place du protocole webSocket s'effectue en plusieurs étapes :

- Connexion d'un client

- Handshake : permet de faire le lien entre le protocole HTTP et WS. Le client envoie une requête HTTP demandant un upgrade de la connexion en protocole WS et le serveur répond en configurant la connexion.
  Exemple de requête du client :

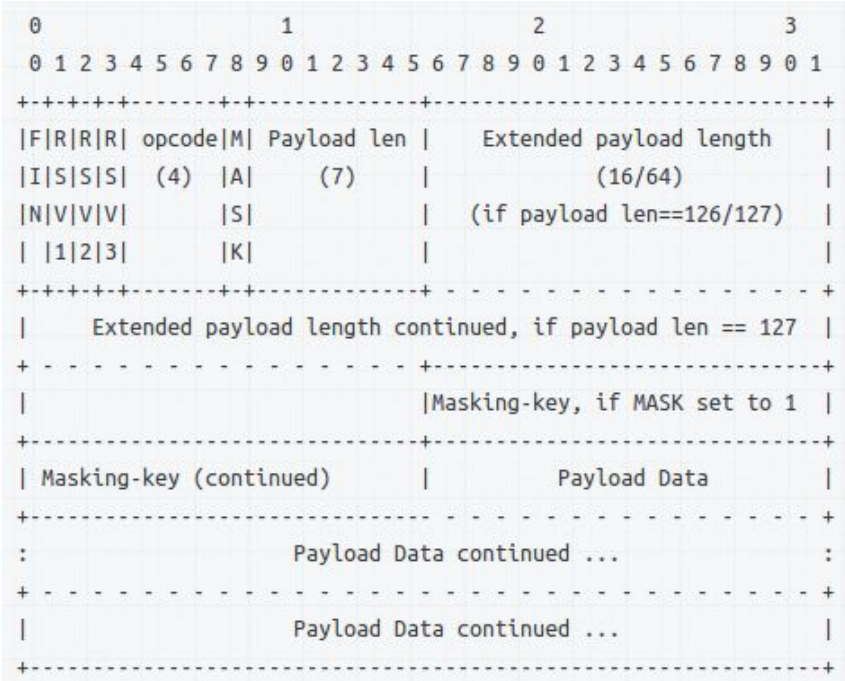
```
GET /chat HTTP/1.1
Host: example.com:8000
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

  Exemple de réponse du serveur :

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

- Réception et décodage des messages : le serveur reçoit des trames brutes, on extrait donc la taille du message ainsi que le message. De plus, les données sont encodées (cryptage XOR) par un masque qui doit également être extrait.
- Fermeture de la communication.

Voici également le format des trames WebSocket :



L’opcode permet de déterminer la nature de la trame (données textes, données binaires, trame de contrôle…), nous avons ensuite un bit MASK qui permet de déterminer si les données sont encodées, puis la taille des données de la trame (extensible sur plusieurs octets), la clé de masquage, et enfin des données (extensibles, jusqu'à 2^63 octets) .

Le masquage des données correspond à un simple cryptage symétrique en ou exclusif qui n’a en aucun cas la vocation de protéger la trame contre des attaques (le cryptage XOR est une très faible sécurité qui est de toute façon totalement anéantie, la clé étant passée en clair dans la trame) mais plutôt d’éviter que des intermédiaires comme des proxys n'interprètent des données textes comme des commandes car l’encodage XOR rend le texte inintelligible. La spécification WebSocket affirme que les données client vers serveurs doivent être obligatoirement masquées, sinon le serveur doit déconnecter immédiatement le client.

Il existe plusieurs trames dites de contrôle identifiées grâce à l’opcode, notamment les trames de PING ou de PONG (le serveurs et clients s’échangent régulièrement ces trames) les trames CLOSE qui sont reçue lorsque le client se déconnecte.

A noter que le premier octet FIN + OPCODE permet de détecter les fragmentations de trames. En effet, il est possible qu’une implémentation des WebSockets choisisse de découper des trames, nous pouvons donc les détecter et les réassembler grâce à cet octet.

IV.2.2. Implémentation Java

Le choix du Java provient du fait que c’est le nouveau langage que nous avons appris ce semestre que nous souhaitions nous tourner vers des styles de programmations quelques peu plus différents que d’habitude.

Les APIs WebSocket en Java sont rares et mal documentées et nécessite quelques fois l’ajout de programmes tiers, nous perdons donc en accessibilité et compréhension du code. Nous avons donc choisi de créer notre propre implémentation de serveur WebSocket en utilisant des librairies de sockets TCP. Il s’est avéré que ce choix était un peu ambitieux et nous pris une bonne partie du temps de réalisation du projet mais nous avons finalement un serveur WebSocket fonctionnel qui respecte les spécifications du protocole WebSocket.

Notre serveur WebSocket est basé sur une classe serveur TCP pré-existante en Java qui implémente des méthodes pour connecter un client et dialoguer en TCP avec celui-ci.

A chaque fois qu’un client se connecte, le serveur décode la requête HTTP et renvoie la confirmation d’upgrade au protocole WebSocket (handshake), un thread est ensuite créé pour dialoguer avec le client sans bloquer le serveur. Le serveur peut donc gérer parallèlement plusieurs clients (le nombre maximal est fixé à 100).

Notre serveur est ensuite capable d’envoyer et de recevoir des trames WebSocket (au nombre maximal de données soit 2^63 octets).

Notre serveur gère également les trames de contrôles : il renvoie un PONG lorsqu'il reçoit un PING, il déconnecte un client qui lui a envoyé un CLOSE…

En outre, notre serveur est capable de gérer la fragmentation TCP et WebSocket. En effet, il n'est pas garanti qu'une trame WebSocket arrive dans une seule trame TCP et dans ce cas on se sert de la taille de la trame WebSocket afin de vérifier que toutes les données ont bien été transmises. La fragmentation WebSocket est ensuite gérée grâce à l'octet OPCODE+FIN (des codes précis existent pour faciliter le réassemblage des trames) mais nous avons remarqué que cette fragmentation est très rare sur un réseau local (contrairement à la fragmentation TCP).

IV.2.3. Notre protocole

Nous utilisons le protocole WebSocket afin de transmettre des commandes simples mais également des images complètes et des ensembles de points. Il a donc fallu définir un protocole personnalisé simple qui s'organise comme suit :

- Le client envoie des commandes textes comme :
    - "project" => lance une projection si une image est chargée
    - "stop" => arrête la projection
    - "shutdown" => éteint la Raspberry
    - "reset" => redémarre la Raspberry
    - "send" => annonce la transmission d'une image

- Dans le cas d'une commande "send" :
    - Le serveur envoie 'N' si un autre utilisateur est déjà en train de transmettre (uniquement dans le cas d'une image, les transmissions sont bien parallèles mais on admet qu'une image à projeter à la fois)
    - Le serveur envoie 'Y' sinon
    - Le client envoie ensuite deux valeurs pour le seuillage par hystérésis (cf. détection de contours
    - L'utilisateur envoie ensuite l'image sous la forme d'une suite de composantes RGB précédée de la taille de l'image codée sur quatre octets
    - A tout moment le serveur peut renvoyer 'E' si une erreur apparaît notamment si la taille des données reçues ne correspond pas à la taille de l'image.
    - Si aucune erreur n'intervient le serveur renvoie au client une suite de points (en 2D, après la détection de contours) afin que l'interface web puisse tracer la prévisualisation de l'image.

IV.2.4. Support des logs

Nous nous sommes vite rendu compte de l'importance d'avoir une trace exhaustive de l'activité du serveur. Nous avons donc créé une classe Java paramétrable qui gère les sorties formatées d'erreurs, de warnings, d'information. Ces informations sont redirigées vers la sortie standard mais il également possible de créer un fichier de log (dans notre cas : log.txt) avec ces mêmes informations. Il est possible d'activer un mode DEBUG sur le serveur qui permet d'obtenir toutes les informations concernant les trames WebSocket (type, taille, contenu, fragmentation…). Toutes ces informations sont précédées de la date et l'heure.

IV.3. Algorithme de détections de contours

L'objectif de notre projet est de pouvoir projeter n'importe quelle image envoyé grâce à l'application web. Pour cela nous allons tracer à une vitesse élevée les contours de l'image sur par exemple un mur grâce au laser. Il est donc nécessaire de détecter les contours de l'image grâce à plusieurs traitements. Le serveur implémente donc plusieurs algorithme permettant une détection de contours.

IV.3.1. Conversion en niveau de gris

Une des étapes préliminaires pour effectuer la détection de contour est de convertir l'image couleur en niveau de gris. Pour cela l'algorithme est simple : il suffit pour chaque pixel de l'image d'effectuer la moyenne des composantes rouge, verte, bleu de chaque pixel. En effet, si l'on considère une image matricielle (l'image est considérée comme un tableau à deux dimensions avec chaque case qui contient la couleur du pixel), chaque pixel contient les composantes RGB codées sur un octet. La couleur gris correspond aux composantes RGB égales. Le niveau de gris est donc codé de 0 à 255, avec 0 donnant du noir et 255 du blanc. Voici le résultat obtenu :

IV.3.2. Filtre de Prewitt et Sobel

Un contour est défini par un changement brutal d'intensité lumineuse, il s'agit donc de détecter selon plusieurs directions (au moins 2) ces changements. Il existe de nombreuses techniques afin de détecter les contours d'une image (gradient, Laplacien). L'une des techniques avec un rapport efficacité/temps d'exécution correct repose sur les algorithmes de Prewitt ou Sobel. L'algorithme repose sur la convolution de matrices. Cela consiste à appliquer sur chaque pixel de l'image la matrice de Prewitt définie comme

$$L_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$$

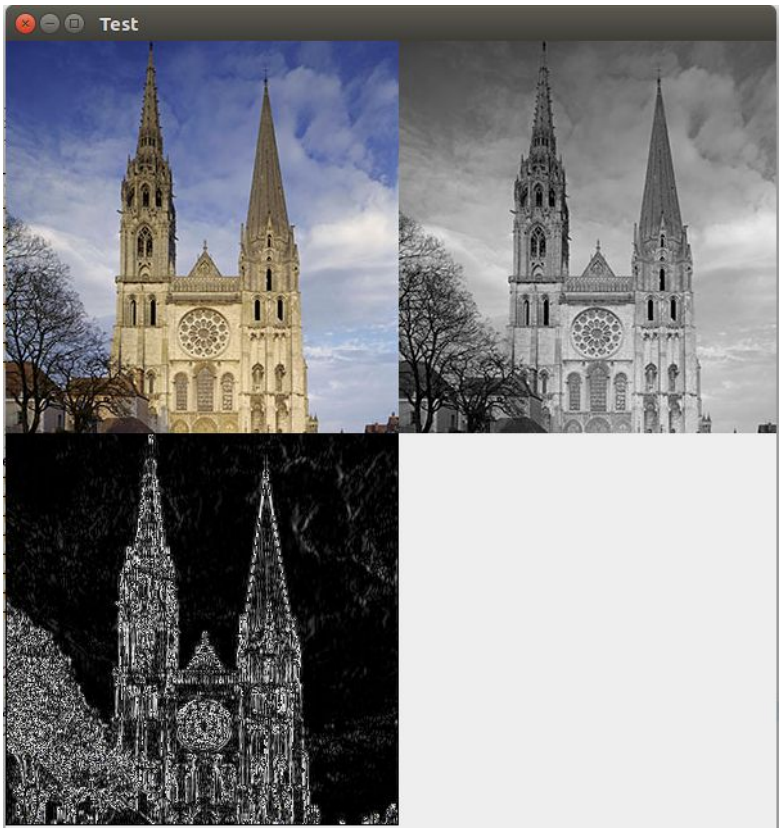
ainsi que sa transposée :

$$L_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$$

Convoluer ces deux filtres sur l'image permet de détecter respectivement les contours verticaux et horizontaux.
D'après wikipédia, on obtient de meilleurs résultats en appliquant un filtre triangulaire, on introduit donc la matrice de Sobel :

$$L = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

On obtient le résultat suivant :



IV.3.3. Pré-traitement : Flou Gaussien

Afin de réduire significativement le nombre de contours, il est nécessaire d'effectuer un pré-traitement avant la détection de contours. On utilise généralement un flou gaussien afin de réduire les transitions lumineuses et ainsi réduire le nombre de contours. L'algorithme de flou gaussien repose également sur la convolution de matrices sauf qu'ici la matrice filtre est générée grâce à une fonction gaussienne paramétrée par **Sigma** qui va directement impacter le niveau de flou. Plus la matrice de filtrage est grande plus la qualité du résultat est meilleure, mais le temps d'exécution devient également plus grand.

Voici la fonction de génération du filtre :

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

où x et y correspondent aux indices de la cellule de la matrice.

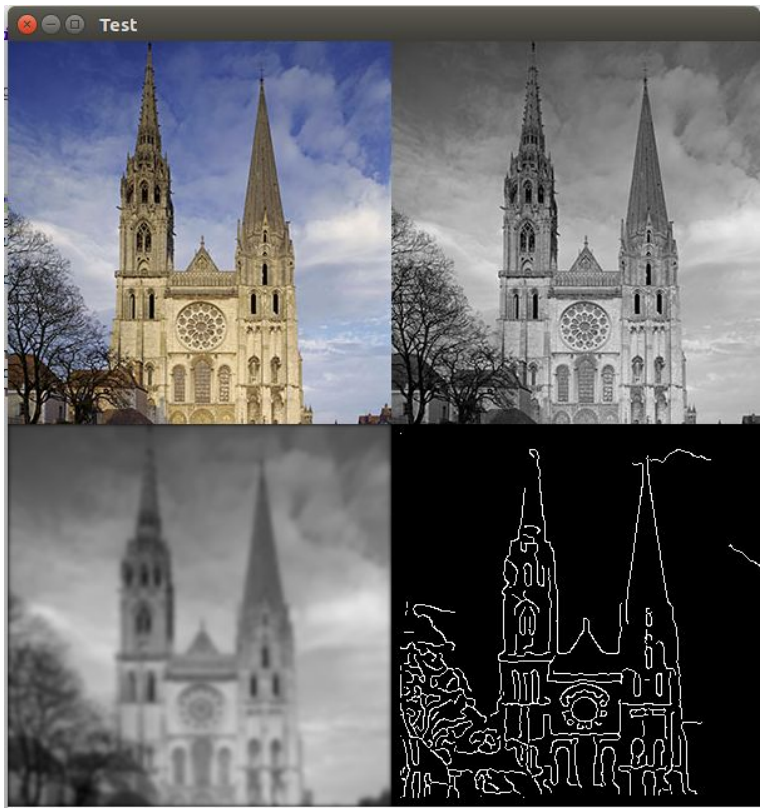
On obtient ce résultat :

On note un bien meilleur résultat. Cependant on remarque que les composantes horizontales ne sont pas correctement détectées il subsiste un problème avec le filtrage vertical que je n'ai pas réussi à identifier. De plus, il s'agit par sa suite d'éliminer les contours parasites ainsi que d'affiner le trait pour n'avoir plus qu'un pixel d'épaisseur.

IV.3.4. Post-traitement : Algorithme de Canny

Afin d'obtenir un résultat exploitable il est nécessaire d'appliquer encore quelques traitements. Pour cela nous utilisons l'algorithme de Canny qui repose que celui de Sobel mais qui ajoute un algorithme de seuillage par hystérésis qui permet de retirer les contours secondaires et de ne garder que les contours connexes dans un seuil haut et un seuil bas définis. En outre, cette algorithme affine également le trait obtenu. Pour cet algorithme nous avons directement trouvé des sources libres de droits (cf. Wiki) qui donnent des résultats très corrects.

Par exemple :



IV.4. Interfaçage avec l'Arduino pro mini

Le serveur dialogue également avec l'Arduino Pro mini afin de lui transmettre les points à projeter via une liaison I2C. A l'origine, il était prévu de transférer l'intégralité des points à projeter dans la mémoire de l'Arduino, puis de les projeter. Néanmoins, nous n'avions pas tenu compte de la faible quantité de mémoire RAM disponible, soit uniquement 2k octets ce qui est bien inférieur au nombre de points à transmettre (déjà codés sur deux octets). En conséquence, nous avons choisi d'envoyer les données et de les traiter immédiatement, soit un traitement en flux. Et c'est ici qu'intervient la principale limite de notre projet : la vitesse de projection est limité par la vitesse de transmission de la liaison série I2C qui est déjà configuré à presque son maximum (au maximum les pertes de données étaient trop importantes) soit 2,9 Mbits/s. Nous avons donc tenté d'autres liaisons séries, mais la liaison UART est difficile à utiliser sur l'Arduino Pro Mini et ne donnait aucun résultat et la liaison SPI a ici un une fréquence trop élevée même à sa valeur minimale pour être correctement traitée par l'arduino.

V-Contrôle des galvanomètres

La commande des galvanomètres s'est finalement avéré être une des parties les plus facile du projet. En effet, en se ramenant à une commande en tension il devenait plutôt aisé d'obtenir les positions désirées du faisceau laser. Il suffisait de réduire l'amplitude désiré d'un point d'une image, c'est à dire la distance entre ce point et le centre de l'image, à un écart codé sur un octet qui paramètre ainsi la PWM.

Nous avons donc configuré l'Arduino pour avoir une interruption sur la réception de la liaison I2C et à chaque interruption, on lit la donnée qui est alternativement la valeur codant la position en X, puis la valeur codant la position en Y et ainsi on détermine la valeur de la PWM. Ceci est également à l'origine d'un autre défaut : quand une valeur est perdue sur le bus I2C (arrive généralement à des débits proches du maximum supporté par la Raspberry), l'arduino inverse les positions X et Y, donc l'image apparaît retournée. Nous avons donc, après de nombreux tests, trouvés une valeur du débit I2C qui limite ce phénomène tout en restant à une valeur suffisamment élevée pour limiter le clignotement de l'image.

VI-Utilisation

VI.1. Accès à l'interface web

L'accès à l'interface web se fait très simplement, il faut simplement s'assurer que vous utilisez un navigateur qui supporte le protocole WebSocket (malheureusement assez rare sur smartphone). La Raspberry Pi étant configurée en point d'accès Wifi, il faut se connecter au réseau "Projecteur-laser" avec mot de passe "projecteur", ouvrir un navigateur, taper l'adresse "192.168.3.1" (il était prévu d'utiliser un serveur bind9 mais le temps nous as manqué) et vous pouvez maintenant utiliser le projecteur.

VI.2. Accès au serveur

Le serveur WebSocket est lancé automatiquement au démarrage la Raspberry Pi mais dans l'éventualité d'avoir recours au mode de débogage ou de contrôle manuel des galvanomètres voici la procédure :

- Le serveur étant lancé par défaut en tâche de fond il faut récupérer la main sur celui-ci ou bien tuer le processus associé au nom "Projecteur.jar" et le relancer avec

la commande “java -jar Projecteur.java” dans le répertoire /home/Pi. Plusieurs modes sont ensuite possibles.

- L’appui de la touche ‘d’ permet d’activer ou de désactiver le mode debug de trames du serveur
- L’appui de la touche ‘r’ permet un reboot complet du serveur
- L’appui de la touche ‘t’ permet de contrôler les galvanomètres manuellement en leur spécifiant une position
- L’appui de la touche ‘m’ permet également de contrôler manuellement les galvanomètres mais ici en utilisant les touches ‘z’,’q’,’s’,’d’ pour descendre, monter, aller à gauche ou à droite
- L’appui de toute autre touche provoque un arrêt propre du serveur

VII- Résultats, limites, pistes de solutions

VII.1. Résultat et limites

Voici les résultats obtenus pour les trois grands axes de notre projet.

VII.1.1. Serveur WebSocket

Le serveur WebSocket est parfaitement fonctionnel, les transferts d’images se déroulent sans encombres, le serveur gère très bien plusieurs utilisateurs, et nous permet de transmettre des images dans délai très raisonnables (moins d’une seconde, c’est la détection de contours qui prend le plus de temps à réaliser). De plus, en cas d’erreur, le serveur est capable de la gérer et d’en avertir l’utilisateur.

Néanmoins, nous avons expérimenté plusieurs fois des navigateurs qui ne réussissait pas à se connecter au serveur, nous recommandons donc l’utilisation de Firefox qui fonctionne également sur mobile.

VII.1.2. Détection de contours

Les algorithmes de détection de contours donnent des résultats corrects dans des délais acceptable. De plus le nombre et la densité des contours peuvent être paramétrés par l’utilisateur ce qui est utile dans le cas d’image complexe avec beaucoup de détails.

Néanmoins, l’algorithme de tri de la plus courte distance n’est pas aussi performant. C’est un algorithme assez naïf de notre propre conception qui permet de trier un ensemble de points afin de les parcourir de proches en proches ceci afin que le laser n’aille pas tracer des points à l’opposé l’un de l’autre et d’optimiser la vitesse. C’est un algorithme assez chronophage qui donne des résultats qui ne sont pas optimaux. Il est très certainement perfectible.

VII.1.3. Contrôle galvanomètres

C’est certainement cette partie qui reste la plus perfectible. En effet, nous arrivons à projeter correctement une image qui n’est pas trop complexe. Néanmoins, cette image est quelque peu déformée et imprécise. De plus sur des images trop complexes, la vitesse du laser (limité par l’I2C) n’est plus suffisante pour assurer la persistance rétinienne et l’image clignote. Nous sommes donc bridés par la vitesse de l’I2C et cela empêche par exemple de coder une position d’un axe sur deux octets, car le temps serait doublé et l’effet de clignotement augmenterai encore. De plus, le signal de sortie du pololu n’est pas parfait, c’est certainement dû au lissage RC du signal de sortie PWM et il provoque une ondulation (ce pololu est surtout utilisé dans la commande de moteurs DC qui ne souffrent pas du bruit qu'il engendre, mais ce n’est pas le cas de nos galvanomètres très sensibles).

VII.2. Pistes de résolutions

Bien que le projet soit arrivé à son terme, nous ne sommes pas à court de propositions pour résoudre les problèmes rencontrés. En voici une liste :

- Amélioration de l’algorithme de tri de la plus courte distance, notamment pour un gain en temps d'exécution mais surtout pour réduire drastiquement le nombre de points afin d’atteindre le seuil des 2K octets afin de pouvoir transmettre l’intégralité des points à projeter et ne plus être limité par la liaison I2C. On pense notamment à une détection d’angles dans le dessin à projeter, ce qui permettrait d’éliminer de nombreux points intermédiaires.

- Ajout d’une mémoire externe à l’Arduino si cela est possible et qu’il n’est pas trop ardu d’interfacer les deux.

- Passer par un convertisseur numérique-analogique (pas contrôlé en I2C) de résolution 8 voir 16 bits soit directement connecté à la Raspberry Pi et ainsi s'affranchir de l’Arduino et de la limitation de 2k octets, mais je pense que l’arduino pourra travailler à une fréquence plus élevée que la Raspberry donc il faudrait coupler cette solution à l’une des deux premières.

Conclusion

En conclusion, nous sommes arrivés à un résultat correct et fonctionnel du projet, respectant une grande partie du cahier des charges. En effet, l'utilisateur peut importer ou dessiner une image, l'envoyer au serveur pour que celui-ci la traite, la renvoyer à l'interface web qui l'affichera et enfin décider de projeter ou non l'image obtenue. Nous avons connu quelques difficultés quant à la réalisation du projet avec des composants parfois mal adaptés qui ont nécessité plus de réflexion. Le projet reste ainsi perfectible avec notamment la connexion I2C qui ne propose pas une fréquence assez rapide et peut donc provoquer des scintillements dans l'image projetée. Ce projet a été pour nous un enrichissement personnel que ce soit dans le travail en équipe, dans l'organisation ou encore dans la réflexion à la réalisation de ce projet.