

Réalisation d'une plateforme de visualisation holographique de type hélice holographique

Rapport de projet T052 – A2021

MERCY Valentin
LEVAYER Sandra

Département Informatique

Université de Technologie de Belfort-Montbéliard

12 Rue Thierry Mieg

90000 BELFORT



Enseignant responsable

GECHTER Franck

Remerciements

Nous tenons en premier lieu à remercier l'ensemble des personnes ayant permis l'aboutissement du projet, pour leur accompagnement tout au long du semestre. Nous remercions particulièrement :

- M. Gechter, responsable du sujet, qui nous a accompagné et conseillé quand il le fallait
- M. Zann, ingénieur au Crunch Lab, qui nous a fourni les composants nécessaires à la fabrication de l'afficheur
- L'équipe du Crunch Lab, dont M. Lamotte et M. Laïpe qui nous ont permis d'utiliser les différents équipements de l'OpenLab

Table des matières

Remerciements	2
Table des figures	5
Introduction	7
1 Présentation du sujet	7
1.1 Etat de l'art des affichages holographiques.....	7
1.1.1 Définition de l'affichage holographique	7
1.1.2 Le fantôme de Pepper	8
1.2 Les affichages holographiques à hélice tournante.....	9
1.3 Objectifs du projet proposé	10
2 Conception et réalisation d'un prototype	10
2.1 Problématiques et solutions proposées.....	10
2.1.1 Choix de la source lumineuse.....	10
2.1.2 Dimensionnement	11
2.1.3 Motorisation.....	12
2.1.4 Transmission du contact en rotation	14
2.1.5 Mesure de la vitesse de rotation	16
2.2 Conception électronique.....	17
2.2.1 Alimentation	17
2.2.2 Capteur de prise d'origine	17
2.2.3 Schéma final et liste des composants utilisés	19
2.3 Conception mécanique	20
2.4 Réalisation.....	21
3 Programmation du prototype.....	21
3.1 Système d'exploitation	21
3.2 Préparation de la séquence d'affichage	22
3.3 Affichage de la séquence sur l'hélice.....	23
3.3.1 Essais en Python.....	24
3.3.2 Passage au langage C	25
3.4 Architecture logicielle globale	29
3.5 Interface graphique de commande et autres fonctionnalités ajoutées	29
4 Résultat.....	32
5 Organisation et fonctionnement du binôme	33
6 Conclusion	34
7 Bibliographie	34
8 Annexes	35
8.1 Annexe 1 : code Python des fonctions permettant de créer la séquence d'affichage...	35

8.2	Annexe 2 : Code C permettant d'afficher une image avec l'hélice – fonctions et macro	38
8.3	Annexe 3 : Code de l'interface web de commande de l'afficheur.....	39

Table des figures

Figure 1 : photographie du premier hologramme authentique mis au point par LG	8
Figure 2 : Schéma du dispositif du fantôme de Pepper	8
Figure 3 : Schéma du fonctionnement de l'œil humain	9
Figure 4 : nano-ordinateur de type Raspberry Pi Zero W	10
Figure 5 : Ensemble de points à prélever sur l'image	11
Figure 6 : Image générée par le simulateur l'image d'entrée est le logo de l'UTBM)	12
Figure 7 : Photographie d'une bague collectrice pleine	14
Figure 8 : Schéma de principe d'une bague collectrice creuse	14
Figure 9 : Montage envisagé dans le cas d'une bague collectrice creuse (vue en coupe)	15
Figure 11 : Modélisation 3D d'une bague collectrice creuse - vue de face	15
Figure 11 : Modélisation 3D d'une bague collectrice creuse - vue en perspective	15
Figure 12 : Modélisation 3D du bloc moteur	16
Figure 13 : capteur à effet Hall	16
Figure 14 : acquisition du signal du capteur à effet Hall avec l'oscilloscope pour déterminer la vitesse de rotation de l'hélice	17
Figure 15 : alimentation de PC similaire à celle qui nous a été fournie	17
Figure 16 : Schéma de l'AOP UA741 utilisé en comparateur	18
Figure 17 : Capteur de fin de course à contact physique (à gauche) et à barrière lumineuse (à droite)	18
Figure 18 : signal numérique renvoyé par le capteur de fin de course à barrière lumineuse	19
Figure 19 : Schéma du montage électronique conçu pour l'afficheur	19
Figure 20 : vue en perspective	20
Figure 21 : vue arrière du boîtier de base	20
Figure 22 : vue en perspective du boîtier moteur	20
Figure 23 : Vue de côté du capteur de prise d'origine	20
Figure 24 : Photographie du bloc moteur et du capteur	21
Figure 25 : Connexions électroniques à l'intérieur boîtier de base	21
Figure 26 : bloc moteur et hélice vus de face	21
Figure 27 : Afficheur complet vu de face	21
Figure 28 : découpage du balayage de l'hélice en secteurs équiangulaires	23

Figure 29 : Mesure à l'oscilloscope de l'impulsion haute renvoyée par le capteur de fin de course lors du passage de l'obstructeur.....	25
Figure 30 : Photographie de l'hélice affichant le nombre maximum de secteurs.....	26
Figure 31 : Description des trames de données attendues par les bandeaux APA102...	27
Figure 32 : Traitements réalisés afin d'affihcer une image sur l'hélice	29
Figure 33 : Interface de commande de l'afficheur - avant prévisualisation.....	30
Figure 34 : Interface de commande de l'afficheur – après prévisualisation	31

Introduction

Le présent rapport rend compte du projet réalisé dans le cadre de l'unité de valeur TO52 « projet de développement informatique ». Celui-ci a consisté à concevoir, fabriquer et programmer un dispositif d'affichage à hélice holographique. Cette technologie, en vogue actuellement, consiste à afficher du contenu (image fixe, vidéo ou animation 3D) sans que l'observateur ne puisse voir le support d'affichage : il a alors l'illusion que ce contenu flotte dans l'air, c'est l'objectif premier d'un affichage holographique. Contrairement aux moyens d'affichage classiques (écran LED, plasma, projecteur sur écran blanc), l'affichage holographique offre une expérience utilisateur bien plus immersive. Plusieurs technologies d'affichage holographique existent : l'hologramme vitrine (« Pepper's Ghost »), le cube à LED, ou encore l'hélice holographique. Ce dernier dispositif consiste à utiliser une propriété très intéressante de la vision humaine, qui peut être vue comme un de ses principaux défauts mais que nous détournerons ici à notre avantage : il s'agit de la persistance rétinienne, ou « Persistence Of Vision » en anglais.

1 Présentation du sujet

1.1 Etat de l'art des affichages holographiques

1.1.1 Définition de l'affichage holographique

Le terme d'affichage « holographique » utilisé dans ce rapport est en fait un abus de langage pour désigner l'illusion d'optique créée par un tel affichage. Cette illusion d'optique permet de donner à l'observateur l'impression qu'il est face à un hologramme réel (affichage 3D sans support) alors qu'il n'en est rien, puisqu'un support est bel et bien présent. En effet, l'hologramme réel consistant à projeter un objet tridimensionnel pour l'afficher dans l'espace 3D sans support, tel qu'imaginé par Jules Verne en 1892, n'est pour l'instant possible qu'en laboratoire et dans des conditions très particulières : c'est l'exploit qu'a réalisé l'entreprise LG en 2015 en projetant un cube coloré grâce à un système d'interférences entre plusieurs lasers.

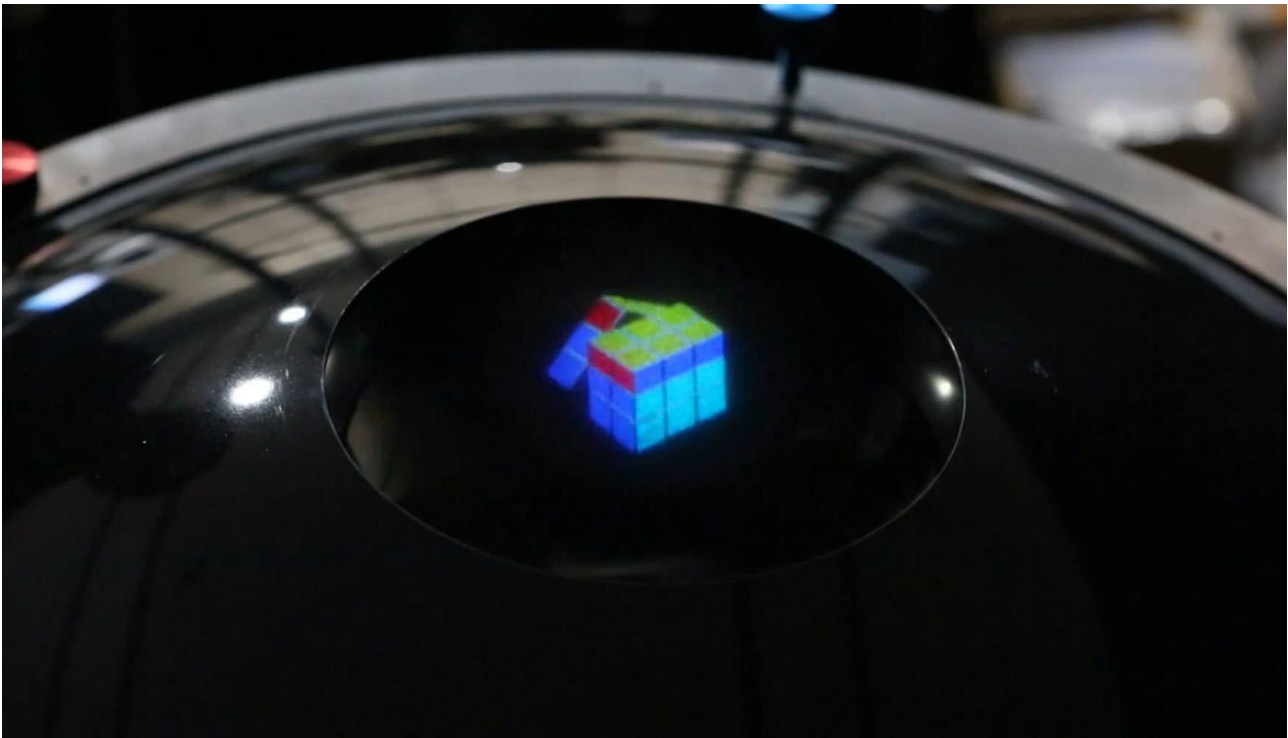


Figure 1 : photographie du premier hologramme authentique mis au point par LG

Dans la suite de ce rapport, on désignera donc par « affichage holographique » tout dispositif permettant de donner l'illusion d'un hologramme en utilisant un support d'affichage 2D rendu invisible.

1.1.2 Le fantôme de Pepper

Parmi eux, le plus connu est l'affichage holographique de type Pepper's Ghost (« Fantôme de Pepper » en français). Celui-ci est principalement utilisé dans les représentations scéniques, notamment pour faire apparaître des personnages réels à deux endroits simultanément, des personnages imaginaires ou faire réapparaître des artistes décédés sur scène.

Le principe du fantôme de Pepper est relativement simple :

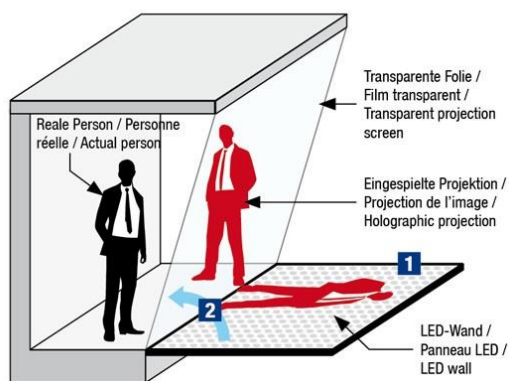


Figure 2 : Schéma du dispositif du fantôme de Pepper

- Un écran 2D classique est placé à l'horizontale (en le rendant si possible invisible par l'observateur, par exemple en le cachant dans une fosse)
- Une vitre ou un film transparent incliné est placé sur la scène, au-dessus de l'écran
- Ainsi, l'image affichée par l'écran projette une lumière qui se reflète sur la vitre transparente pour être visible du spectateur
- Le spectateur voit donc la scène en arrière-plan (avec tout ce qui peut s'y trouver, notamment des personnes réelles), auquel est incrusté l'image holographique diffusée sur l'écran

Toutefois, ce système comporte plusieurs inconvénients. Premièrement, la vitre inclinée est généralement faite de verre, qui a forcément une épaisseur non nulle. Cette épaisseur cause donc un

double reflet : un premier sur la surface côté spectateur, un second sur la surface côté scène. L'illusion est donc imparfaite. De plus, cette installation est difficilement transportable à cause de la taille de l'écran et de la vitre.

1.2 Les affichages holographiques à hélice tournante

Parmi les technologies d'affichage holographique disponibles actuellement, l'hélice tournante offre des résultats satisfaisants malgré l'apparente simplicité du dispositif. Pour cela, les hélices holographiques exploitent le principe de persistance rétinienne.

La persistance rétinienne est un phénomène bien connu de la vision humaine. Ce dernier est notamment constitué d'une rétine, qui tapisse le fond de l'œil et permet de capter les différents rayons lumineux qui ont été convergés par le cristallin pour former une image. Ces informations sont ensuite transmises au cerveau sous forme de signaux électriques par le nerf optique.

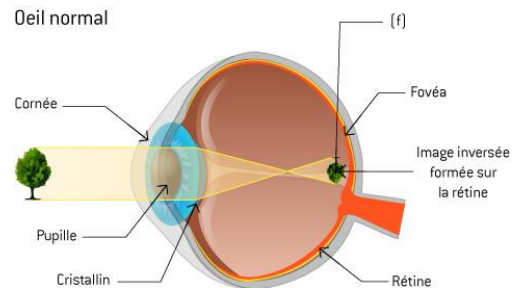
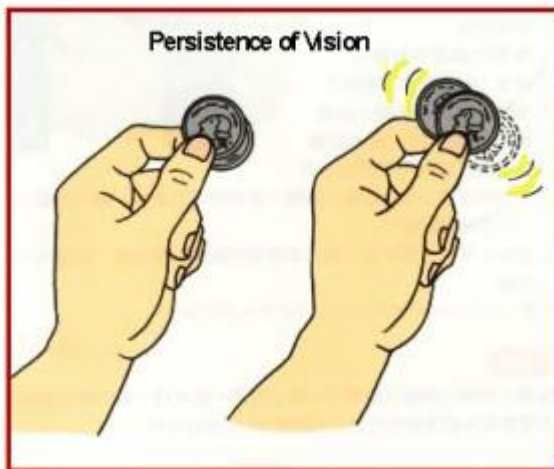


Figure 3 : Schéma du fonctionnement de l'œil humain

Toutefois, une fois cette transformation effectuée par la rétine, l'image convertie est gardée en mémoire quelques instants (environ 50 millisecondes) avant d'être renouvelée. Ceci est une conséquence du traitement biochimique des signaux optiques captés par les photorécepteurs (cônes et bâtonnets) qui

composent la rétine, traitement effectué par le cerveau et qui n'est pas instantané. Ainsi, comme ce processus se répète en permanence, le cerveau garde en mémoire la dernière image qu'il a traitée complètement pour assurer la continuité de notre vision. La durée de cette persistance n'est pas identique en tout point de l'image : plus le rayon est lumineux (parce qu'il est directement émis par une source de lumière ou bien parce qu'il est réfléchi par une surface très réfléchissante), plus sa projection sur l'image captée par la rétine sera persistante.



Pour observer le phénomène de persistance rétinienne, une expérience simple est de déplacer à

grande vitesse un objet sur un fond noir tout en l'observant de face : l'observateur (supposé fixe) alors l'impression que l'objet se trouve partout en même temps, le long de son trajet. Cette conséquence de la persistance rétinienne est simple à expliquer. Considérons un échantillon infinitésimal du déplacement effectué par l'objet, entre deux points A et B quasiment confondus (très proches). Lorsque l'objet se trouve en A, la rétine de l'observateur imprime l'image de l'objet sur fond noir se trouvant en A. Cette image prendra un certain temps à s'effacer à cause de la persistance rétinienne. Lorsque l'objet se trouve en B, l'image de l'objet en A ne s'est pas encore effacée mais la rétine capte déjà l'image de l'objet en B. Le même processus se répète sur tout le déplacement de l'objet : les images de l'objet aux différentes positions s'additionnent et l'observateur ne perçoit que cette image résultante. Il est alors incapable de déterminer la position de l'objet à un instant t puisqu'il ne perçoit que son déplacement.

Le principe de l'hélice holographique est d'exploiter cette conséquence de la persistance rétinienne. Pour cela, une multitude de sources de lumière (en l'occurrence, des LED) sont montées côte à

côte sur un support entraîné en rotation autour de son centre (support qu'on nommera hélice par abus de langage, bien qu'elle n'ait pas de forme hélicoïdale puisqu'elle n'est bien-sûr pas utilisée comme moyen de propulsion dans ce montage). Rappelons que puisqu'il s'agit de sources de lumières, la persistance rétinienne est plus forte et plus longue. L'observateur perçoit alors une image qui correspond à la combinaison des positions instantanées de l'hélice en rotation, c'est-à-dire un disque lorsque toutes les LED sont allumées. Il suffit alors de contrôler ces LED séparément et en synchronisation avec la position de l'hélice, pour obtenir l'illusion d'un affichage fixe. Comme la persistance des LED est dominante (puisque'il s'agit de sources lumineuses), l'hélice disparaît dans l'arrière-plan et le support d'affichage devient donc invisible, donnant l'illusion d'un hologramme.

1.3 Objectifs du projet proposé

L'objectif proposé pour ce projet est de concevoir et réaliser un outil de projection holographique de type hélice rotative. Ceci comprend la conception mécanique et électronique du prototype et la programmation de l'affichage. Quelques contraintes ont également été données. Concernant la mécanique et l'électronique, nous devons nous appuyer sur le matériel et les outils disponibles au Crunch Lab de l'UTBM. Concernant l'aspect informatique du projet, il est impératif de se limiter à la puissance de calcul d'un nano-ordinateur de type Raspberry Pi Zero W.



Figure 4 : nano-ordinateur de type Raspberry Pi Zero W

De plus, il est attendu que le prototype soit suffisamment abouti et simple d'utilisation pour pouvoir être utilisé en tant que support de communication innovant lors d'événements ouverts au public.

2 Conception et réalisation d'un prototype

2.1 Problématiques et solutions proposées

Avant de commencer la conception du prototype à proprement parler, nous avons listé les principales problématiques liées au projet et avons commencé à y chercher des solutions. Nous avons essayé de trouver un maximum de solutions par nous-mêmes avant de nous inspirer des solutions employées par des personnes ayant déjà réalisé ce genre d'appareils.

2.1.1 Choix de la source lumineuse

Premièrement, nous avons choisi la source de lumière que nous allions utiliser. Celle-ci devait répondre à plusieurs contraintes : elle devrait être composée de plusieurs éléments les plus rapprochés possibles, pouvant être allumés de différentes couleurs et contrôlables séparément. L'emploi de diodes électroluminescentes (LED) nous semblait alors le plus adapté. Ces dernières années, les bandeaux de LED se sont beaucoup développés. Généralement utilisés comme décoration d'intérieur, la plupart d'entre eux n'offrent pas la possibilité de contrôler chaque élément séparément. Toutefois, ces dernières années les bandeaux de LED dits « adressables » se sont beaucoup développés. Ceux-ci offrent justement cette possibilité. Nous nous sommes donc renseignés davantage sur leurs caractéristiques et la manière de les contrôler. La caractéristique la plus intéressante dans notre cas est la densité de LED : il s'agit du nombre de LED par unité de longueur. En effet, de cette densité allait dépendre directement la résolution radiale de notre affichage. Après quelques recherches, nous avons trouvé que la densité maximale disponible actuellement sur le marché était de 144 LED par mètre, ce qui nous paraissait suffisant. En effet, les

modèles d'hélices holographiques en vente sur internet offrent une densité de 533 LED/m et obtenir un affichage de résolution radiale égale au quart de celle d'un modèle commercial serait satisfaisant. Une autre possibilité aurait été de faire notre propre montage à base d'ampoules LED individuelle, mais ceci aurait demandé un montage électronique bien trop complexe (démultiplexeur à plusieurs centaines de canaux de sortie) et nous n'aurions pas pu le miniaturiser suffisamment pour qu'il puisse être monté sur le rotor.

Nous avons donc opté pour un bandeau de LED adressable de densité 144 LED/m que nous avons trouvé à l'UTBM Crunch Lab. Celui-ci porte la référence APA102.

2.1.2 Dimensionnement

Connaissant désormais le type de bandeau LED que nous allons utiliser, nous pouvons dimensionner l'hélice, c'est-à-dire déterminer la longueur de la section de bandeau que nous allons utiliser. Pour cela, nous avons créé un simulateur d'affichage à hélice holographique. Le principe est simple : il s'agit de créer un programme informatique qui prenne en entrée une image et fournisse en sortie une prévisualisation de l'image telle qu'elle sera affichée par l'hélice holographique.

Le traitement réalisé par le programme est relativement simple :

1. Premièrement, il calcule les coordonnées d'un ensemble de points formant une grille circulaire où chaque point appartient à un rayon. Le long d'un rayon, les points sont calculés pour être régulièrement espacés, comme c'est le cas sur un bandeau de LED. Cette opération est réalisée par une fonction qui prend en entrées le nombre de points sur chaque rayon (résolution radiale) et le nombre de rayons (résolution angulaire) et renvoie en sortie un tableau contenant l'ensemble des points sous forme de coordonnées polaires.
2. Les coordonnées polaires des points calculés à l'étape précédente sont ensuite converties en coordonnées cartésiennes
3. Puis, une nouvelle fonction prend en entrée l'ensemble des points sous forme de coordonnées cartésiennes et y ajoute, pour chaque point, la couleur prélevée sur l'image à ce point.
4. Enfin, le programme crée une image composée d'un fond noir auquel sont ajoutés les points prélevés sur l'image. Ces points sont colorés avec la couleur prélevée sur l'image à la même position.

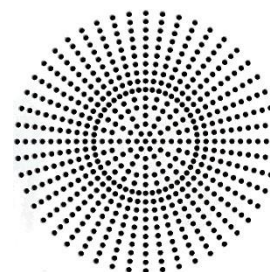


Figure 5 : Ensemble de points à prélever sur l'image

Voici un exemple d'image générée par le simulateur :

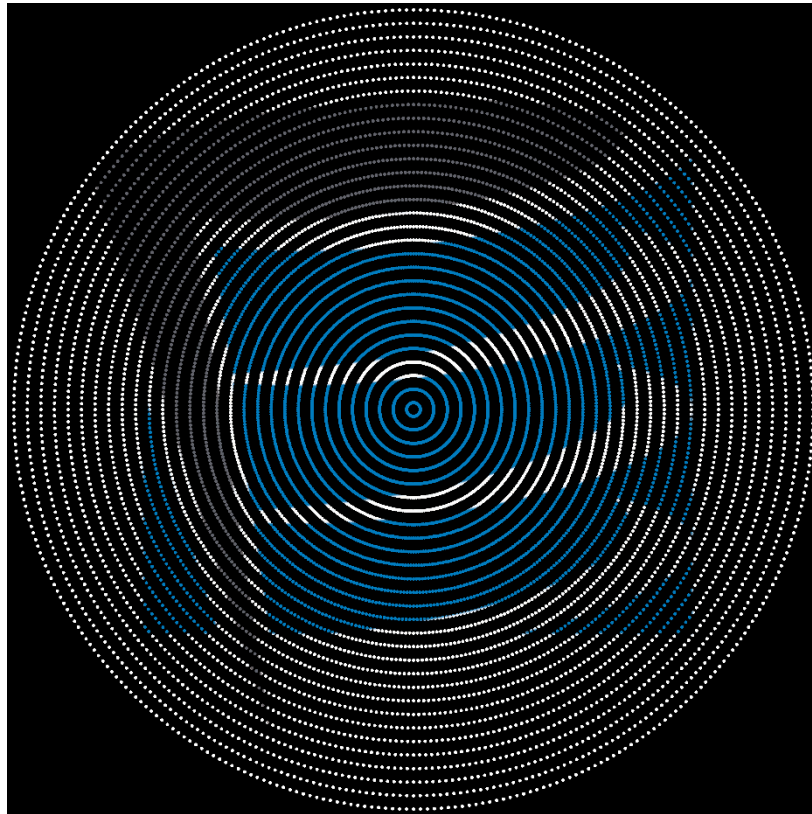


Figure 6 : Image générée par le simulateur l'image d'entrée est le logo de l'UTBM)

Nous avons ainsi pu déterminer qu'une quantité de 48 LED par rayon (soit une hélice comprenant en tout $2 \times 48 = 96$ LED) permettrait d'afficher des images avec une résolution radiale satisfaisante. La longueur de l'hélice serait donc proche de $\frac{96}{144} = 0.67m$.

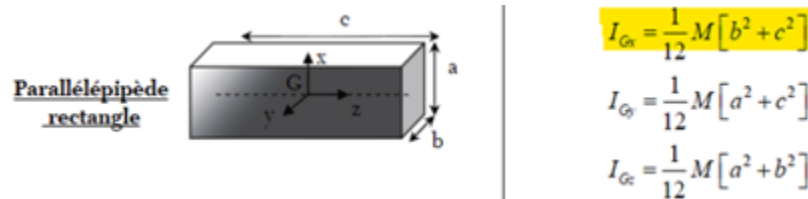
2.1.3 Motorisation

Puis, nous avons réfléchi aux différentes possibilités qui s'offraient à nous en termes de motorisation. Nous avons fait un calcul rapide permettant de déterminer la vitesse de rotation minimum acceptable pour le moteur qui entraînerait l'hélice en rotation : sachant que la persistance rétinienne moyenne de l'œil humain dure 50ms et que le fonctionnement de notre hélice implique qu'elle dure au moins un tour complet, la vitesse minimale de rotation est de : $\Omega_{min} = \frac{1}{50 \times 10^{-3}} = 20 \text{ Hz} \times 60 = 1200 \text{ RPM}$.

Toutefois, nous pourrions nous satisfaire d'une vitesse de rotation avoisinant la moitié de cette valeur. En effet, la valeur de 50ms prise ci-dessus correspond à la persistance rétinienne dite « positive », c'est-à-dire intervenant sans source lumineuse. Mais comme évoqué précédemment, la durée de la persistance rétinienne dépend directement de la luminosité de l'élément observé. Avec une source lumineuse telle que le bandeau de LED utilisé, nous pourrions obtenir un résultat satisfaisant avec une vitesse de rotation de 500 RPM (tours par minute).

L'autre caractéristique intervenant dans le choix d'un moteur est le couple nécessaire pour entraîner l'hélice en rotation. Celui-ci se calcule ainsi : $C = I * \omega'$, où I est le moment d'inertie et ω' l'accélération angulaire de l'hélice.

Le moment d'inertie I se calcule de la façon suivante pour un parallélépipède rectangle :



Dans notre cas, c est égal à la longueur du bandeau et b à la largeur du support utilisé. Nous avons décidé d'utiliser comme support le couvercle d'une moulure électrique que nous avons trouvé dans une enseigne de bricolage. Celui-ci a une largeur de 15mm et une masse de 40 grammes à laquelle s'ajoute la masse du bandeau (40g) pour une masse totale de 80g.

$$\text{On a donc : } I_{Gx} = \frac{1}{12} * 0.08 * ((15 * 10^{-3})^2 + 0.67^2) = 0.003 \text{ kg.m}^2$$

L'accélération angulaire, quant à elle, se calcule en fixant une durée maximale acceptable pour que l'hélice (initialement à l'arrêt), après mise en route, atteigne la vitesse minimale acceptable pour que l'illusion fonctionne (soit $500 \text{ RPM} = 52 \text{ rad.s}^{-1}$). Nous nous sommes fixé arbitrairement une durée de 10s, ce qui donne une accélération angulaire $\omega' = \frac{52}{10} = 5.2 \text{ rad.s}^{-2}$.

$$\text{On obtient donc un couple minimal } C = I * \omega' = 0.003 * 5.2 = 0.0156 \text{ N.m}^{-1} = 160 \text{ g.cm}^{-1}.$$

Plusieurs types de moteurs électriques pourraient répondre à nos besoins.

Nous avons premièrement pensé aux moteurs brushless (sans balais) qui offrent une vitesse de rotation généralement élevée (entre 5000 et 10000 RPM) mais facilement contrôlable grâce aux circuits électroniques de type ESC (*Electronic Speed Controller*). Quant au couple, nous n'avons trouvé aucune donnée concernant les moteurs brushless avec lesquels nous pourrions travailler (principalement le modèle A2212). Les kits composés d'un moteur brushless et d'un ESC se sont largement répandus ces dernières années avec la démocratisation des drones, qui sont pour la plupart motorisés avec ce type de composants. Nous nous sommes donc très facilement procuré ces composants et les avons manipulés à l'UTBM Crunch Lab dans le but de voir si cette solution de motorisation pourrait convenir ou non. Pour cela, nous avons simplement monté l'hélice directement sur l'arbre moteur et avons mis en marche celui-ci à vitesse maximale. L'hélice s'est alors mise en rotation avec une accélération très faible et nous avons observé des saccades importantes dans le mouvement, nous indiquant que le moteur ne fonctionnait pas dans son régime habituel. L'ESC, quant à lui, a subi une forte montée en température qui l'a certainement endommagé. Nous en avons conclu que la puissance absorbée par le moteur (dépendante du couple nécessaire pour mettre en rotation l'hélice) était trop importante et non supportée par l'ESC. Nous avons donc exclu cette solution de motorisation.

La solution que nous avons retenue pour ce projet est d'utiliser un moteur à courant continu. Ce type de moteur cause des pertes énergétiques (sous forme de chaleur) plus importantes mais est encore bien plus répandu que les moteurs brushless. En faisant ce choix, nous augmentons donc nos chances de trouver un moteur adapté à nos besoins en termes de vitesse et capable de fournir la puissance nécessaire pour entraîner l'hélice en rotation.

C'est ainsi que nous avons trouvé à l'UTBM Crunch Lab un moteur d'imprimante papier portant la référence STD MTR QK1-4637 qui semblait parfaitement adapté à notre projet. Après quelques tests réalisés avec une alimentation de laboratoire réglée à 12V, le moteur permettait bien de faire tourner l'hélice à haute vitesse (sans que nous puissions toutefois mesurer celle-ci) mais une impulsion manuelle était nécessaire pour démarrer l'hélice.

Afin que le moteur soit assez puissant pour démarrer lui-même l'hélice, il serait donc nécessaire de ne pas coupler directement l'hélice à l'arbre moteur mais d'intercaler un engrenage permettant de démultiplier le couple. La vitesse, quant à elle, serait réduite et il devenait alors important de la mesurer pour savoir si celle-ci serait toujours supérieure au seuil de 500 RPM (voir la partie [Mesure de la vitesse de rotation](#)).

2.1.4 Transmission du contact en rotation

Puis, s'est posé le problème de l'approvisionnement en électricité de l'hélice. En effet, les LED montées sur l'hélice auront besoin d'énergie électrique pour s'allumer, ainsi que d'un signal leur permettant de s'allumer de la bonne couleur au bon moment. Or, ces connexions (6 canaux au total : +5v, GND, et 2 fils de données par bandeau¹) doivent être continues malgré la présence d'une liaison mécanique de type « pivot » entre le stator et le rotor du moteur. Nous avons alors pensé à deux solutions possibles : la première consistant à intégrer une batterie sur la partie en rotation (le rotor) ainsi qu'un système de communication sans fil pour assurer la communication du signal vers les bandeaux de LED entre le stator et le rotor (hélice). Cette première solution est complexe à mettre en œuvre car elle rajoute une masse non négligeable (celle de la batterie) et implique de recharger cette dernière régulièrement. La deuxième solution, plus élégante et plus pratique, consiste à transmettre les contacts électriques entre le



Figure 7 : Photographie d'une bague collectrice pleine

stator et le rotor à l'aide d'une pièce technique appelée « bague collectrice ». Ce système est composé de deux éléments, un stator et un rotor. Sur le rotor sont disposées des pistes cuivrées (le cuivre étant un excellent conducteur électrique) qui viennent assurer les contacts avec des balais (aussi appelés « charbons » ou « collecteurs ») fixés, eux, au stator.

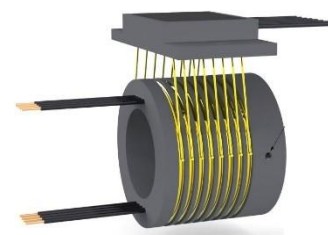


Figure 8 : Schéma de principe d'une bague collectrice creuse

Il existe deux types de bagues collectrices : les bagues collectrices creuses (dont un exemple est représenté sur la figure 8) et les bagues collectrices pleines (dont un exemple est donné en figure 7).

¹ En effet, le protocole SPI du bandeau de LED APA102 permet de chaîner les LED pour n'avoir au final que deux broches de données (MISO et MOSI : Master Input/Output, Slave Input/Output) permettant de commander l'ensemble du bandeau de LED adressable.

Pour ce projet, il nous semblait plus simple d'employer une bague collectrice creuse. En effet, nous pensions ainsi pouvoir coupler directement l'hélice à l'arbre du moteur tout en fixant la bague collectrice autour de ce couplage à l'aide du montage suivant :

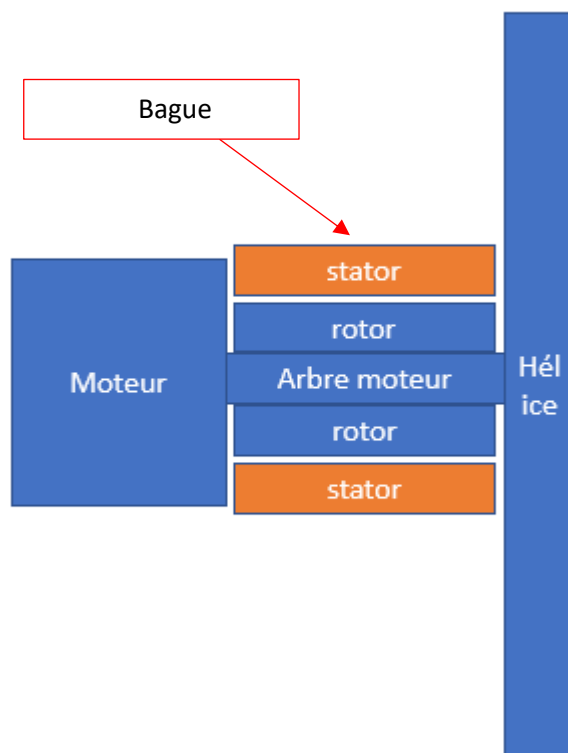


Figure 9 : Montage envisagé dans le cas d'une bague collectrice creuse (vue en coupe)

Toutefois, les bagues collectrices creuses sont minoritaires sur le marché (car moins utilisées dans l'industrie et plus chères à produire) et par conséquent il est difficile d'en trouver à la vente pour un prix raisonnable. Nous avons donc tenté d'en reproduire une nous-mêmes grâce à l'impression 3D. Voici le modèle 3D que nous avons imprimé et assemblé :

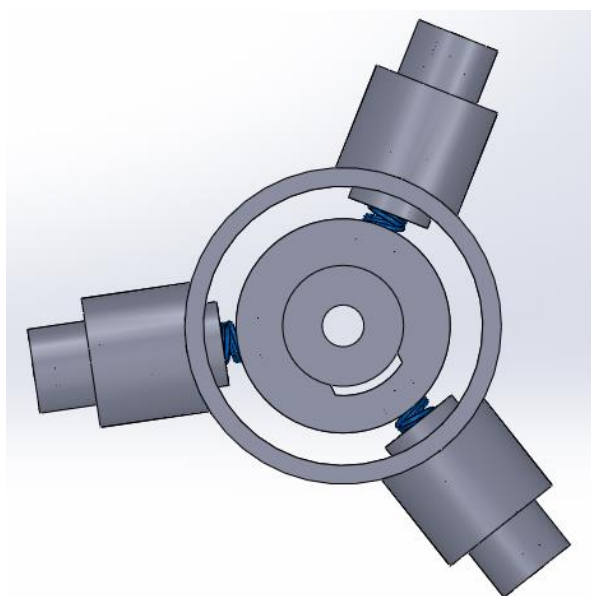


Figure 11 : Modélisation 3D d'une bague collectrice creuse - vue de face

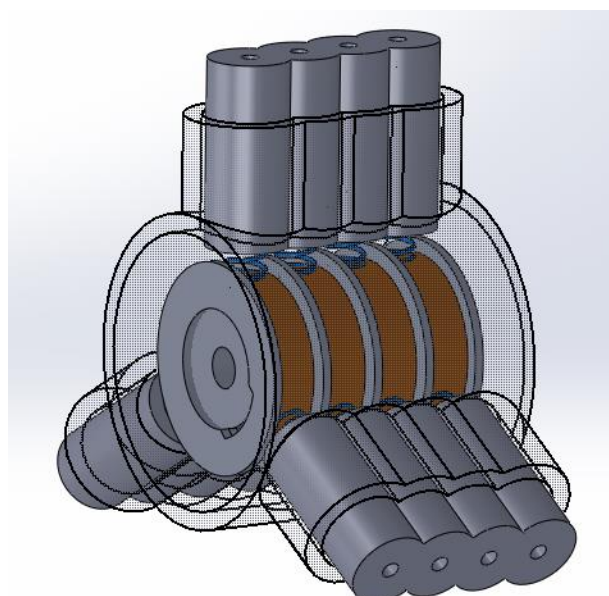


Figure 11 : Modélisation 3D d'une bague collectrice creuse - vue en perspective

La bague collectrice fonctionne par pression des ressorts (attachés au stator) sur les pistes cuivrées collées au rotor creux. Celui-ci est constitué de plastique de type PLA.

Malheureusement, nous avons rapidement dû abandonner cette solution. En effet, le frottement généré par les ressorts sur les pistes cuivrées permet certes de maintenir le contact électrique, mais dégage beaucoup de chaleur ce qui déforme le rotor imprimé en PLA (température de fusion : 175°C).

Les bagues pleines, quant à elles, sont facilement trouvables à un prix raisonnable (environ 10€). Nous avons donc opté pour cette option, malgré la nécessité de sous-jacente de revoir complètement notre montage. En effet, il serait désormais impossible de coupler directement l'hélice à l'arbre moteur puisque celle-ci devra être couplée à la partie mobile (rotor) de la bague collectrice. Nous avons donc confirmé la nécessité de réaliser un montage à engrenage : en effet, il serait ainsi possible d'entraîner le pignon fixé à l'hélice par sa circonférence grâce à un autre pignon. Nous avons donc modélisé un tel assemblage (voir la figure 12).

Après impression 3D des composants et assemblage, l'assemblage a permis de solutionner les deux problèmes précédemment explicités. En effet, l'hélice peut désormais tourner à une vitesse plutôt élevée (à confirmer par un second montage expérimental) tout en démarrant sans assistance grâce au couple supérieur généré par l'engrenage.

En effet, le couple s'est ainsi vu divisé par le rapport de réduction $\eta = \frac{Z_1}{Z_2} \cdot \frac{Z_2}{Z_3} = \frac{Z_1}{Z_3} = \frac{28}{46} = 0.609$

2.1.5 Mesure de la vitesse de rotation

Une fois l'assemblage du bloc moteur réalisé, il était nécessaire de vérifier que la vitesse de rotation serait satisfaisante. Pour cela, nous avons utilisé un capteur à effet Hall. Le capteur à effet hall est un capteur électronique permettant de mesurer une variation de champ magnétique sous la forme d'une variation de tension électrique. Ainsi, en plaçant un aimant sur l'hélice de sorte qu'il passe à proximité du capteur à effet Hall (fixe par rapport au stator), nous pourrions mesurer le temps entre deux passages, et donc deux variations de tension grâce à un oscilloscope.

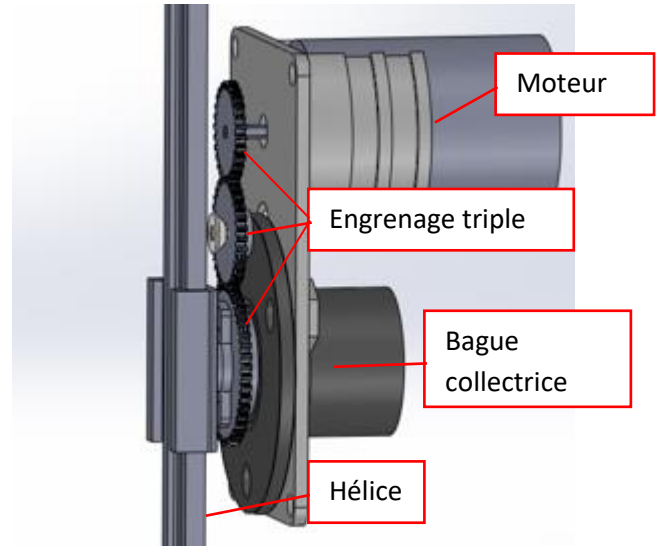


Figure 12 : Modélisation 3D du bloc moteur



Figure 13 : capteur à effet Hall

Une fois le montage réalisé et l'hélice mise en rotation avec une tension de 15V appliquée à l'hélice², voici l'acquisition que nous avons pu faire grâce à l'oscilloscope :

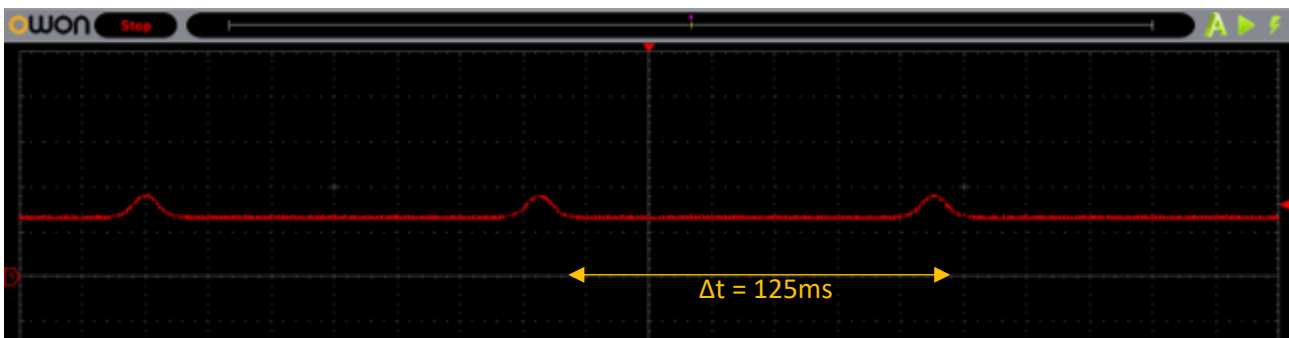


Figure 14 : acquisition du signal du capteur à effet Hall avec l'oscilloscope pour déterminer la vitesse de rotation de l'hélice

On peut alors conclure que la vitesse de rotation de l'hélice est égale à $\omega' = 60 * \frac{1}{\Delta t} = \frac{1}{0.125} = 480 \text{ RPM}$

Cette vitesse avoisine les [500 RPM évoqués précédemment](#) et nous pouvons donc nous attendre à un résultat satisfaisant grâce à ce montage.

2.2 Conception électronique

2.2.1 Alimentation

Nous avons choisi l'alimentation en fonction des besoins que nous aurions en termes de tensions électriques. En effet, nous savons que :

- Le moteur à courant continu utilisé fonctionne sous 15V
- Le Raspberry Pi fonctionne sous 3.3V
- Les bandeaux de LED doivent être alimentés en 5V

Nous avons donc choisi d'utiliser une alimentation de PC. En effet, la plupart des alimentations de PC fournissent ces tensions de manière stable (excepté la différence de potentiel de 15V pour le moteur mais celle-ci peut être obtenue par addition des tensions de -12 et +3.3). Une alimentation de PC nous a été fournie par le Crunch Lab.



Figure 15 : alimentation de PC similaire à celle qui nous a été fournie

2.2.2 Capteur de prise d'origine

La prise d'origine est un problème qui nous a beaucoup occupés dans le cadre de ce projet. La prise d'origine est essentielle pour que l'afficheur soit asservi : elle permet d'indiquer au Raspberry, à chaque tour de l'hélice, la position (et donc l'instant) où doit commencer la séquence d'affichage pour que l'image soit affichée correctement.

Nous avons envisagé dans un premier temps d'utiliser le capteur à effet Hall (déjà utilisé pour [mesurer la vitesse de rotation du moteur](#)). Toutefois, le seul capteur à notre disposition était le modèle KY-

² Nous avons choisi cette tension car elle permettait de maintenir l'hélice en rotation pendant plusieurs minutes sans que la chaleur dégagée par le moteur ne soit trop importante.

035, un modèle renvoyant un signal analogique. Or le Raspberry Pi, via ses ports GPIO³ ne peut lire que des entrées numériques binaires (0 pour l'état bas ou 3.3v pour l'état haut). Ainsi, nous aurions dû utiliser un comparateur analogique à seuil fixe permettant de renvoyer un signal numérique à l'état haut lorsque l'aimant attaché à l'hélice passe devant le capteur, et à l'état bas le reste du temps. Nous avons donc utilisé un amplificateur opérationnel (AOP) de référence UA741 en tant que comparateur pour assurer cette tâche. Le montage ainsi réalisé a fonctionné correctement sur une platine d'expérimentation. Toutefois, devant la complexité du montage (nécessitant entre autres de graver un circuit imprimé pour être intégré au prototype), nous avons pensé à une autre solution plus simple : utiliser un autre type de capteur pour assurer la prise d'origine.

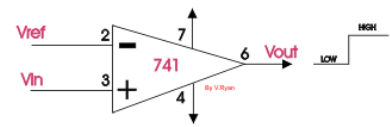


Figure 16 : Schéma de l'AOP UA741 utilisé en comparateur

La contrainte principale à respecter pour le choix d'un tel capteur est qu'il faut éviter tout contact entre l'élément tournant (l'hélice) et la partie fixe sur laquelle sera monté le capteur. Cela élimine donc les capteurs de fin de course avec contact physique, mais il existe des capteurs de fin de course de type « barrière lumineuse » qui pourraient convenir dans notre cas :

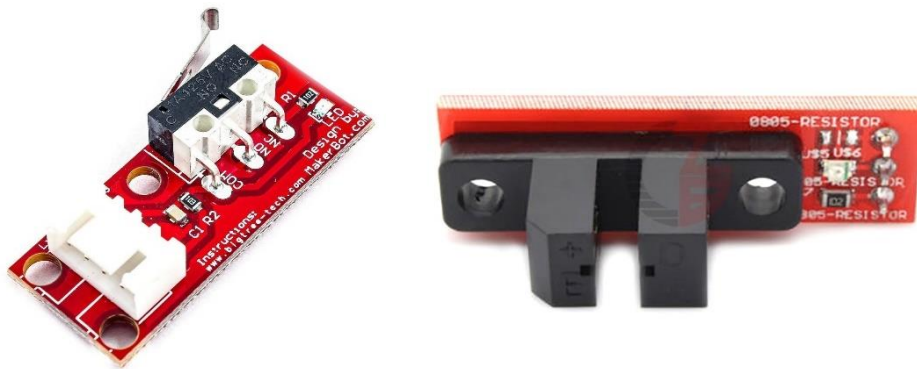


Figure 17 : Capteur de fin de course à contact physique (à gauche) et à barrière lumineuse (à droite)

Ce type de capteur est composé d'une fourche contenant un émetteur et un récepteur infrarouge à chacune de ses extrémités. Le capteur renvoie un signal à l'état bas par défaut, qui passe à l'état haut lorsque de la fourche est obstruée par un élément opacifiant. Ce type de capteur renvoie donc un signal numérique compatible avec les ports GPIO du Raspberry Pi, que nous avons mesuré à l'oscilloscope :

³ General Purpose Input Output

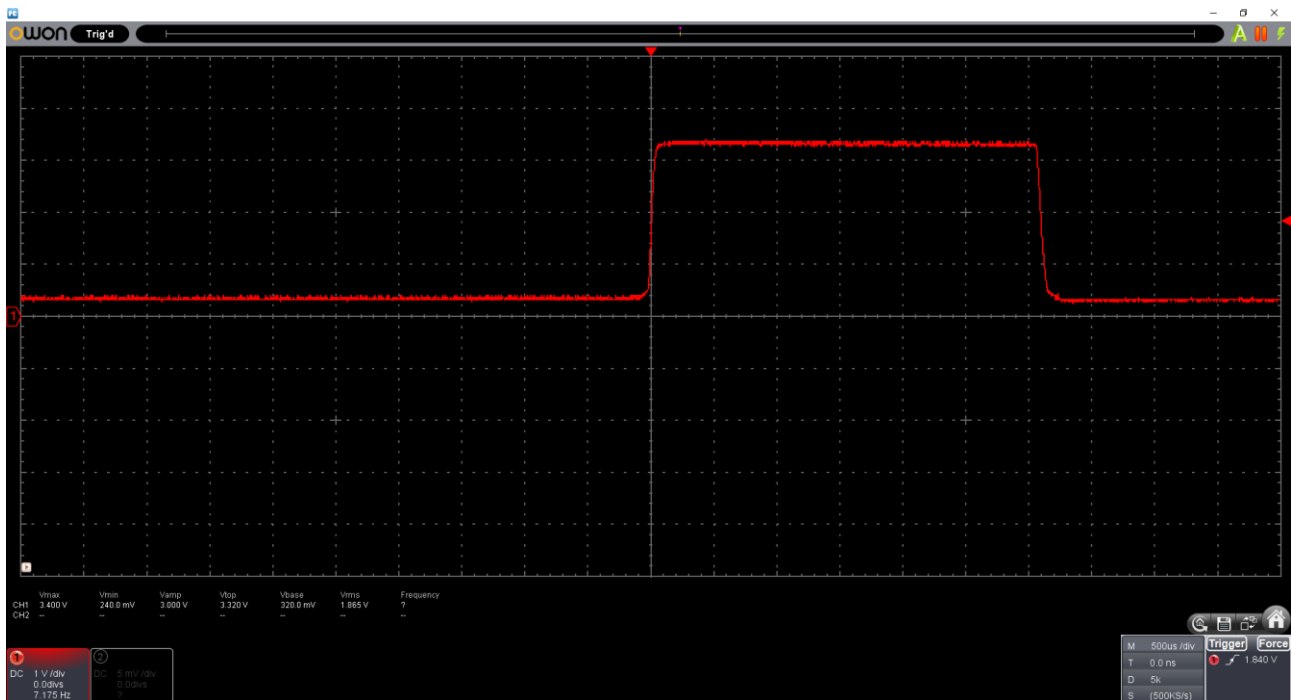


Figure 18 : signal numérique renvoyé par le capteur de fin de course à barrière lumineuse

2.2.3 Schéma final et liste des composants utilisés

Voici le schéma électronique final résultant de notre conception :

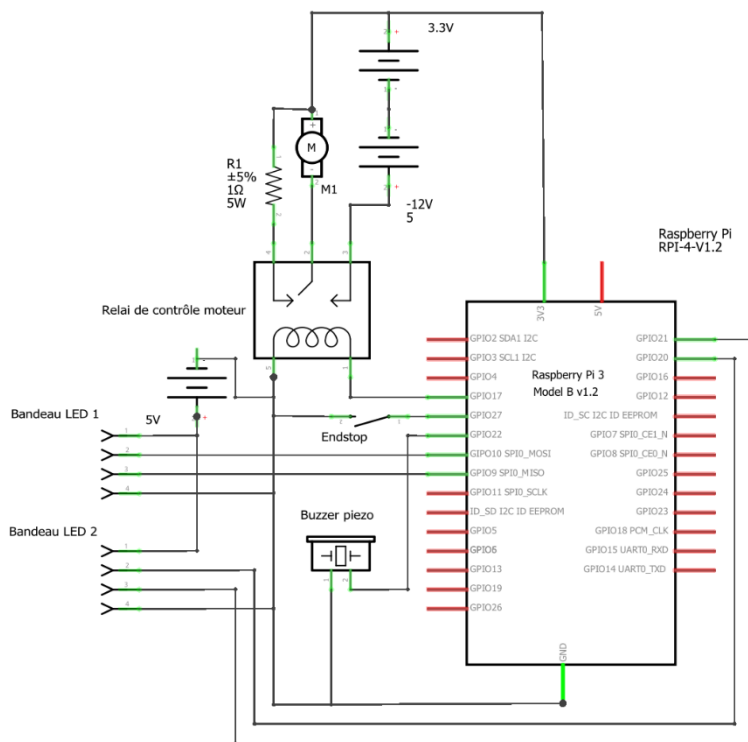


Figure 19 : Schéma du montage électronique conçu pour l'afficheur

Quelques remarques explicatives :

- Nous avons décidé d'utiliser un relai pour commander le moteur. En effet, nous n'aurions pas besoin d'en commander la vitesse de rotation puisqu'il serait soumis à la tension maximale acceptable
- Les connexions entre le moteur et le relai sont étudiées de façon que le moteur, lorsqu'il n'est pas activé, soit relié à une résistance par ses deux bornes. Cela permet au moteur, lors d'un passage de l'état haut (activé) à l'état bas (désactivé), de freiner en dissipant l'énergie cinétique de l'hélice dans la résistance sous forme d'énergie thermique
- Concernant les bandeaux de LED : les fils de connexion aux bandeaux arrivent à l'hélice en son centre, en sortie

de la bague collectrice. Or, un bandeau de LED adressable ne peut être connecté que par son extrémité. Ainsi, pour éviter d'ajouter un fil qui irait du centre de l'hélice à son extrémité (là où

commencerait l'unique bandeau de LED) ce qui risquerait de la déséquilibrer, nous avons décidé de sectionner le bandeau en deux bandeaux de 48 LED chacun. En effet, nous avons pu nous le permettre puisque la bague collectrice utilisée permet bien de transporter 6 canaux (dans notre cas : +5V, GND, et 2 fils de données SPI par bandeau).

- Un buzzer piézoélectrique a été ajouté au schéma électronique. Il permet d'émettre un signal sonore d'avertissement avant la mise en route de l'hélice, à des fins de sécurité.

2.3 Conception mécanique

Une fois que nous avons solutionné les différentes problématiques évoquées ci-dessus et achevé la conception électrique, nous avons continué la conception mécanique de l'afficheur en modélisant complètement celui-ci grâce au logiciel SolidWorks 2015 édité par Dassault Systèmes.

Ce modèle a pris la forme d'un assemblage comprenant l'ensemble des pièces qui composeraient l'hélice. Nous avons décidé de maintenir l'hélice à la verticale en la fixant à un profilé d'aluminium lui-même maintenu par un boîtier lesté par le poids de l'alimentation de PC.

Voici le résultat de cette modélisation :

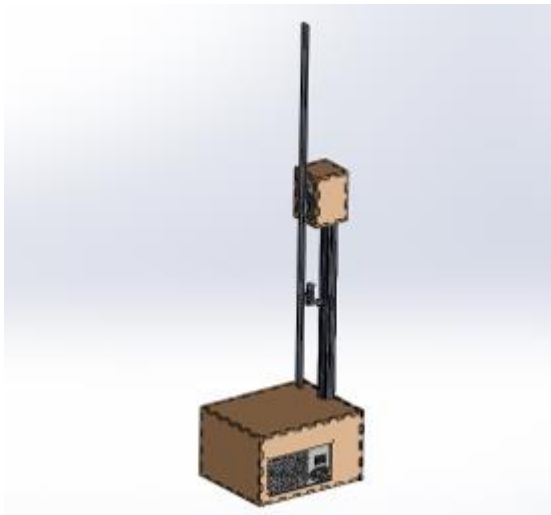


Figure 20 : vue en perspective

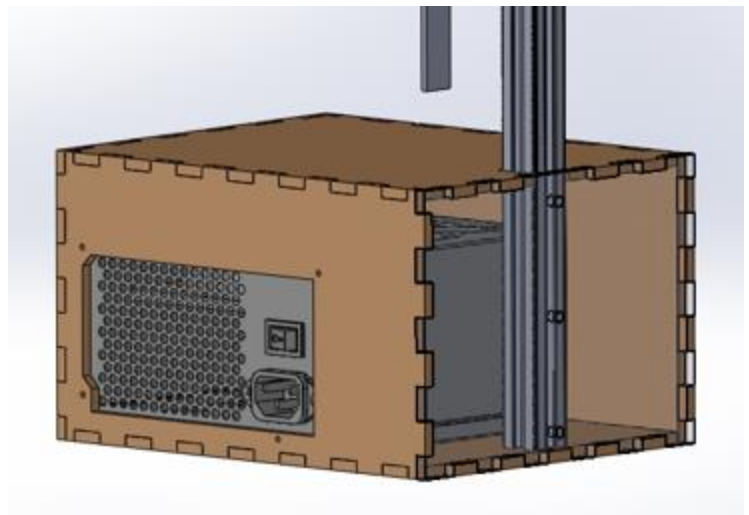


Figure 21 : vue arrière du boîtier de base

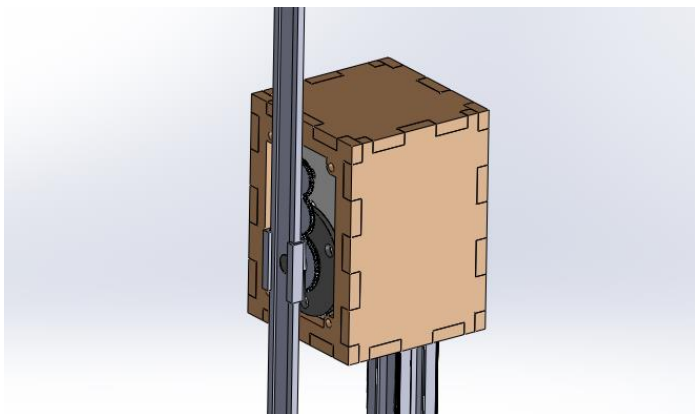


Figure 22 : vue en perspective du boîtier moteur

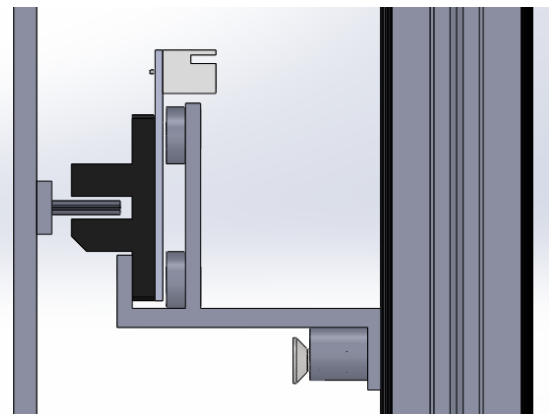


Figure 23 : Vue de côté du capteur de prise d'origine

Pour les deux boîtiers, nous avons choisi un assemblage en créneaux pour un assemblage plus aisé et plus solide.

2.4 Réalisation

Pour réaliser les boîtiers (boîtier de base et boîtier moteur) nous avons utilisé la machine de découpe laser du Crunch Lab afin de découper les faces dans du bois de 5mm d'épaisseur.

Voici le résultat de la réalisation de l'afficheur :

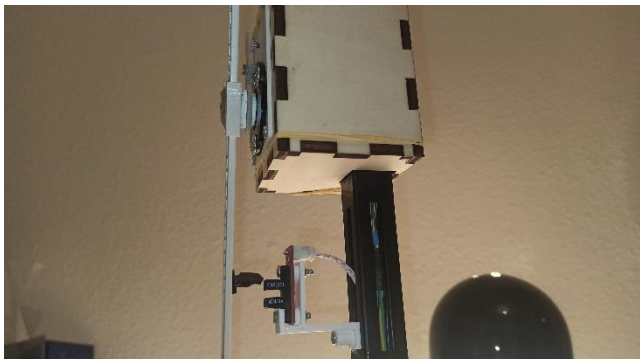


Figure 24 : Photographie du bloc moteur et du capteur

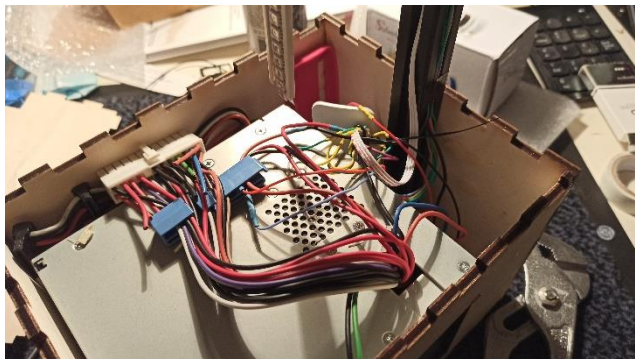


Figure 25 : Connexions électroniques à l'intérieur du boîtier de base

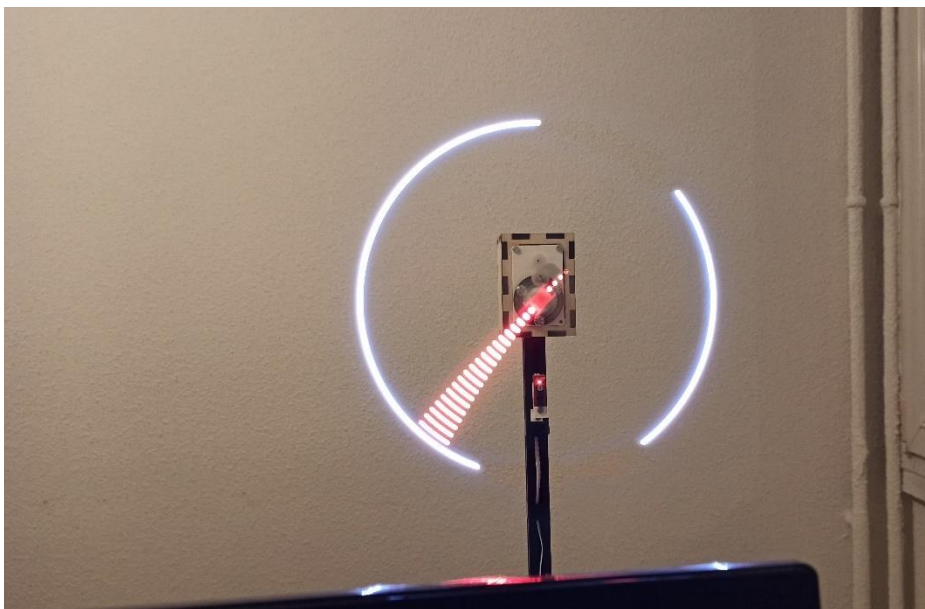


Figure 26 : bloc moteur et hélice vus de face



Figure 27 : Afficheur complet vu de face

3 Programmation du prototype

Une fois l'afficheur assemblé et testé pour être sûrs de la fiabilité mécanique et électronique du montage, nous avons pu entreprendre sa programmation. Dans cette partie du rapport nous vous présenterons l'architecture logicielle retenue et implémentée par nos soins.

3.1 Système d'exploitation

Le Raspberry Pi utilisé pour contrôler l'afficheur fonctionne sous Raspbian, un système d'exploitation libre et gratuit basé sur Debian (distribution Linux) optimisé pour ce nano-ordinateur. Raspbian fournit toutes les bases nécessaires pour programmer l'hélice : un interpréteur Python 3, un compilateur de langage C ainsi que le gestionnaire de version Git et un le protocole SSH qui nous permettraient de mettre à jour le code via la connexion Wi-Fi prise en charge par le Raspberry.

Deux processus principaux sont impliqués par l’affichage d’une image grâce à l’hélice holographique : la préparation de l’image à afficher (traitement prenant en entrée un fichier image brut et donnant en sortie un fichier correspondant à la séquence d’affichage) et l’affichage lui-même (processus prenant en entrée le fichier descripteur de la séquence d’affichage obtenu en sortie du premier processus). Ces deux processus indépendants sont réalisés par le Raspberry Pi grâce aux codes que nous avons écrit.

3.2 Préparation de la séquence d’affichage

Ce premier processus est très proche du [simulateur](#) que nous avons réalisé pour dimensionner notre afficheur. Puisque le simulateur a été réalisé en Python et que ce processus de conversion ne doit pas nécessairement être instantané (nous pouvons donc utiliser un langage interprété), nous avons décidé de l’écrire lui aussi en Python. En effet, cela nous permettrait de gagner du temps puisqu’une bonne partie du travail avait déjà été faite lors de la réalisation du simulateur. Nous en avons donc repris la plupart des fonctions.

Ce processus prend comme entrées l’image à afficher (sous forme de fichier .png ou .jpg), le nombre de secteurs d’affichage (nous expliquerons cette notion dans la sous-partie suivante) et le nombre de LED à utiliser sur chaque bandeau. Il fournit en sortie un fichier .h contenant le tableau des points de couleur à afficher sur l’hélice, sous la forme d’une macro :

```
#define COLOR_POINTS \
{\
{ 0, -47, 0, 0, 0 },\
{ 0, -46, 0, 0, 0 },\
{ 0, -45, 0, 0, 0 },\
{ 0, -44, 0, 0, 0 },\
{ 0, -43, 0, 0, 0 },\
{ 0, -42, 253, 253, 253 },\
{ 0, -41, 248, 248, 248 },\
{ 0, -40, 97, 99, 107 },\
....
```

```
{ 9, 47, 0, 0, 0 },\
}
```

Chaque ligne de ce fichier correspond à un point de couleur prélevé sur l’image, sous la forme de 5 entiers $\{indexAngulaire, indexRadial, red, green, blue\}$ où $indexAngulaire$ et $indexRadial$ sont les coordonnées polaires du point considéré et $red, green, blue$ des entiers entre 0 et 255 correspondent à la couleur à afficher (dans le système des couleurs RGB).

Voici un résumé du traitement effectué par ce premier programme, très similaire au traitement effectué par le simulateur (pour le détail de chaque étape, se reporter à la partie [dimensionnement](#)) :

- Calcul des coordonnées polaires (θ, r) des points constituant une grille circulaire de même taille que l’image, enregistrées dans un tableau
- Pour chacun de ces points :
 - Conversion des coordonnées polaires (θ, r) en coordonnées cartésiennes (x, y)
 - Prélèvement de la couleur du pixel de l’image d’entrée ayant pour coordonnées (x, y) sous forme d’un triplet (r, g, b) qu’on enregistre dans le même tableau
- Tri de ce tableau en fonction des coordonnées polaires (θ en premier niveau, r en second niveau)
- Enregistrement du tableau final dans un fichier .h présenté ci-dessus

Le code Python correspondant à ce traitement est le suivant :

```

# Configuration variables
nbLEDused = 2*35 # number of LED used to display one line
imageFileName = 'logo_utbm.png' # input file name
outputFileName = 'main/libapa102/examples/logo_utbm.h' # path to save the .h
file
previewFileName = 'result_picking.png' # path to save the preview image
nbSectors = 16 # number of displaying sectors
zoomFactor = 90 # image zoom factor in percent

if __name__ == "__main__":
    angleStep = int(180/nbSectors)

    sourceImg = Image.open(imageFileName)
    sourceImg.convert('RGBA')
    imgMaxSize = max(sourceImg.size)

    # we zoom/dezoom the image so that it fits well with display grid
    baseImgSize = imgMaxSize*100//zoomFactor
    centeredImage = Image.new('RGB', (baseImgSize,)*2, 'black')
    centeredImage.paste(sourceImg, ((
        baseImgSize-sourceImg.size[0])//2, (baseImgSize-
sourceImg.size[1])//2))

    pickingPoints = getPickingPoints(baseImgSize, nbLEDused, angleStep) # we
build a grid with same size as the image
    pickedColors = pickColors(pickingPoints, centeredImage) # we pick colors
on each point of this grid

    savePickedColors(outputFileName, pickedColors, nbSectors) # we save picke
d color points in a .h file
    renderPickedPointsPreview(previewFileName, pickedColors, 10) # we render
a preview of the picked color points

```

Le code ci-dessus utilise la librairie PIL en ce qui concerne le traitement d'images ainsi que des fonctions que nous avons-nous-mêmes implémentées et qui sont disponibles en [annexe 1](#).

3.3 Affichage de la séquence sur l'hélice

Ce second processus est au cœur du fonctionnement de l'hélice. Il consiste à lire le contenu du fichier .h écrit par le processus précédent, afin de commander en temps réel les couleurs affichées par les LED de chaque bandeau.

C'est là qu'intervient la notion de secteur d'affichage, qui est le point de départ de notre raisonnement sur l'algorithme à fournir pour assurer l'affichage d'une image par l'hélice. Nous avons en effet divisé le cercle qui correspond au balayage de l'hélice en rotation, en plusieurs secteurs équiangulaires. Leur nombre est variable et dépend directement de la fréquence maximale à laquelle il nous sera possible de rafraîchir les couleurs des bandeaux de LED. Cette fréquence dépend elle-même de la

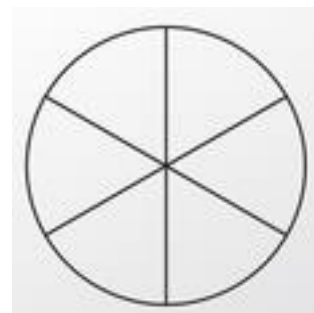


Figure 28 : découpage du balayage de l'hélice en secteurs équiangulaires

vitesse de calcul du Raspberry d'une part, mais est également limitée par la vitesse du protocole de communication (SPI, en l'occurrence) qui permet au Raspberry Pi de commander les bandeaux de LED. Cette vitesse de communication entre le Raspberry Pi et les bandeaux est incompressible et inconnue (à mesurer), tandis que la vitesse de calcul du Raspberry Pi, elle, dépend essentiellement du langage de programmation utilisé pour coder ce processus d'affichage.

Le traitement correspondant à ce processus est relativement simple :

- On lit le fichier .h contenant la séquence d'affichage
- On met l'hélice en rotation en basculant le relai de commande moteur à l'état haut
- A chaque passage de l'élément obstruteur dans le capteur de fin de course
 - Pour chaque secteur appartenant à la séquence d'affichage
 - Pour chaque point de couleur appartenant à ce secteur
 - Soit n l'index radial du point de couleur
 - Soit (r, g, b) le triplet correspondant à la couleur prélevée sur l'image
 - Si $n > 0$:
 - On allume la n -ième LED du premier bandeau, de couleur (r, g, b)
 - Sinon :
 - On allume la $(-n)$ -ième LED du second bandeau, de couleur (r, g, b)
- Si un signal d'interruption SIGINT⁴ survient :
 - On arrête la rotation de l'hélice en basculant le relai de commande moteur à l'état bas
 - On éteint l'ensemble des LED des deux bandeaux

3.3.1 Essais en Python

Nous avons essayé dans un premier temps de programmer cette tâche au moyen d'un script Python. Mais nous nous sommes heurtés à la vitesse maximale de lecture des ports GPIO permise par la librairie Python RPi.GPIO, qui était bien plus basse que nous le pensions : ainsi, l'obstruction du capteur de fin de course n'était pas détectée à chaque tour ce qui empêcherait l'afficheur d'afficher l'image en continu. Pour comprendre ce phénomène, nous avons mesuré l'impulsion haute renvoyée par le capteur de fin de course à l'aide d'un oscilloscope. Nous avons obtenu le chronogramme suivant :

⁴ Les signaux des systèmes d'exploitation peuvent-être vus comme des événements déclenchés volontairement ou involontairement par l'utilisateur. En l'occurrence, nous nous intéressons ici au signal SIGINT qui correspond à l'interruption d'un processus, que ce soit par l'utilisateur (déclenché alors par le raccourci clavier Ctrl+C) ou par le système d'exploitation lui-même (en cas d'erreur ou d'extinction du système). Nous cherchons donc à capter ce signal pour nous assurer d'arrêter l'hélice lorsque l'un des cas précités se produit.

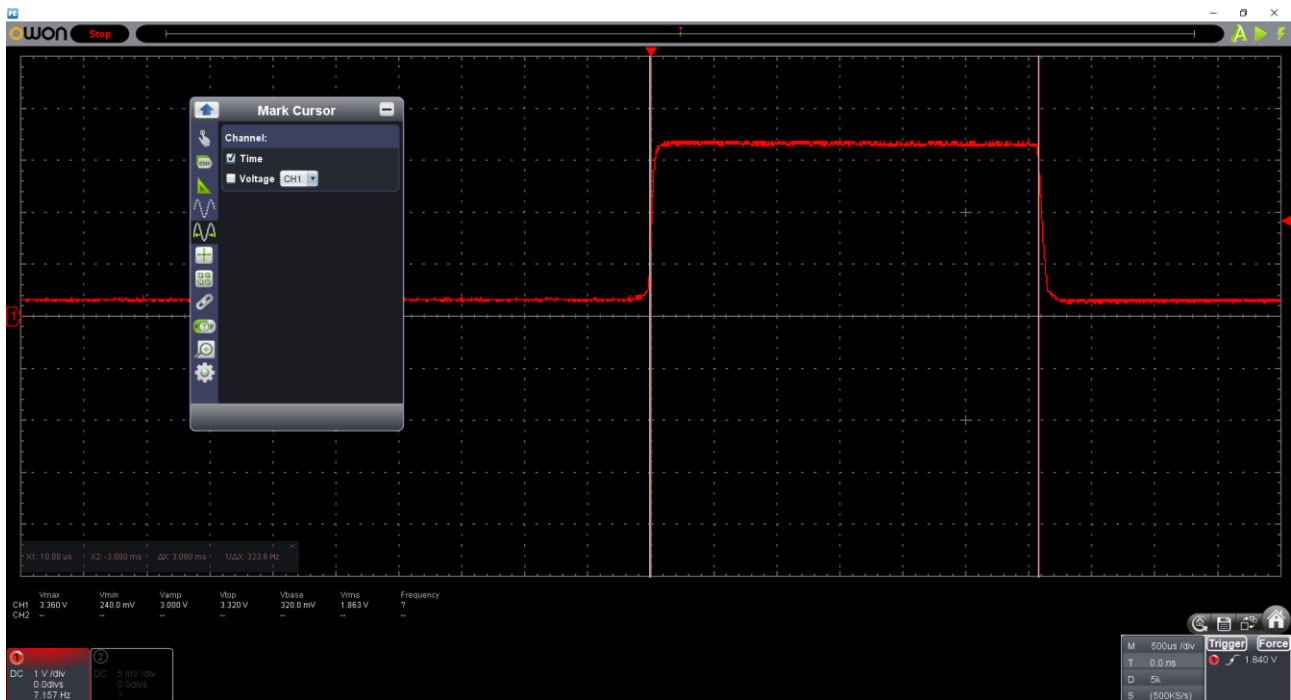


Figure 29 : Mesure à l'oscilloscope de l'impulsion haute renvoyée par le capteur de fin de course lors du passage de l'obstruteur

La mesure de la durée de l'impulsion donne un résultat de 3.090ms ce qui correspond à une fréquence de 323.6Hz. Selon le théorème de Shannon indiquant que la fréquence d'échantillonnage doit être strictement supérieure au double de la fréquence maximale du signal, nous devons donc échantillonner ce signal à une fréquence minimale égale à $2 * 323.6 \cong 650 \text{ Hz}$.

3.3.2 Passage au langage C

Nous avons donc décidé de coder le processus d'affichage en C, un langage compilé et non interprété donc plus bas niveau et plus rapide que Python. En effet, nous avons pu lire que la fréquence maximum de lecture des ports GPIO est de l'ordre de 14 MHz en C (ce qui est bien supérieur à notre besoin de 650 Hz), alors que nous n'avons pas trouvé de donnée concernant la lecture des ports GPIO avec notre précédente configuration (en Python avec la librairie RPi.GPIO sur un Raspberry Pi Zero W).

3.3.2.1 Bibliothèques utilisées

Alors que nous avons déjà trouvé l'ensemble des bibliothèques Python permettant d'interfacer avec les ports GPIO de la Raspberry et les bandeaux de LED APA102, nous avons du trouver leurs équivalents en C.

Concernant les ports GPIO, la librairie WiringPi permettant de les commander à la fois en lecture et en écriture fait déjà partie de Raspbian, nous avons donc juste à l'inclure.

Concernant les bandeaux de LED, nous avons trouvé la librairie [libapa102](#) sur GitHub. Cette librairie donne les fonctions de base permettant de commander un bandeau APA102 en C, toutefois :

- Elle ne permet d'allumer le bandeau que d'une couleur unie (toutes les LED de la même couleur), or nous devons bien-sûr allumer les bandeaux de plusieurs couleurs
- Elle ne permet de commander qu'un seul bandeau relié au port SPI0 de la Raspberry Pi, or nous utilisons aussi le port SPI1 pour commander le deuxième bandeau.

Nous avons donc dû modifier cette librairie pour ouvrir ces nouvelles fonctionnalités : rajouter un paramètre entier sur toutes les fonctions permettant d'initialiser et de commander un bandeau, correspondant au bandeau cible (ce paramètre modifie le channel SPI appelé par la librairie WiringPiSPI

utilisée par libapa102), et implémenter une nouvelle fonction permettant d'allumer un bandeau selon une trame de couleurs différentes :

```
/**
 * @brief Drives an APA102 LED Strip by filling it according to a color frame
 *
 * @param strip The strip to drive
 * @param colors 2-
dimensionnal array containing colors to display in the form [LEDIndex][Color] where
Color belongs to {0,1,2} = {R,G,B}
 */
void APA102_FillWithDifferentColors(struct APA102* strip, uint8_t colors[][3]
) {
    uint8_t led_frame[4];
    int i;

    APA102_Begin(strip->interface);
    for(i = 0; i < strip->n_leds; i++) {
        led_frame[0] = 0b11100000 | (0b00011111 & 31); //start frame
        led_frame[1] = colors[i][2];
        led_frame[2] = colors[i][1];
        led_frame[3] = colors[i][0];

        wiringPiSPIDataRW(strip->interface, led_frame, 4); // strip->interface is
now passed to wiringPiSPIDataRW to specify the strip to fill
    }
    APA102_End(strip->interface);
}
```

3.3.2.2 Détermination du nombre maximum de secteurs

Nous n'avons pas tout de suite codé le traitement permettant d'afficher une image avec l'hélice, mais avons dans un premier temps cherché à déterminer le nombre maximum de secteurs d'affichage à l'aide d'un programme plus simple (les secteurs affichés étant de couleur unie, alternée un secteur sur deux). Le résultat obtenu est visible sur la figure 30. Il est important de préciser que, comme toutes les photographies de l'hélice en fonctionnement, le fait qu'on ne puisse observer qu'une partie de l'image affichée par l'hélice est tout à fait normal puisque que le capteur de l'appareil photo utilisé (qui équivaut à la rétine de l'œil humain) n'a pas la même persistance. Ainsi, un observateur humain verrait bien la totalité de l'image.

En extrapolant la figure 30, on peut dénombrer 16 secteurs sur la moitié du parcours de l'hélice (l'hélice étant double, on ne compte qu'un balayage de 180°). Nous avons donc notre résolution angulaire maximale : $\frac{180}{16} = 11.25^\circ$.

En faisant d'autres tests en utilisant seulement une partie des bandeaux de LED, nous nous sommes aperçus que davantage de secteurs apparaissaient. Nous avons alors compris que le nombre de secteurs

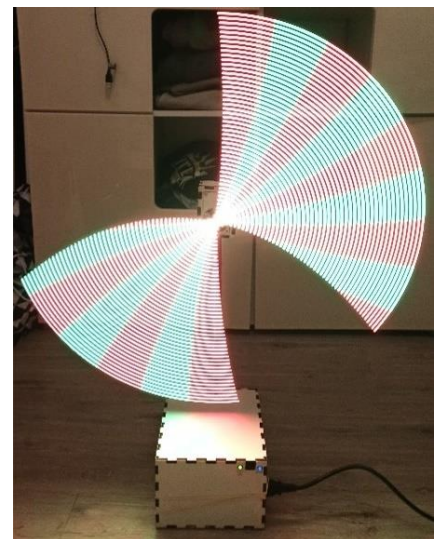


Figure 30 : Photographie de l'hélice affichant le nombre maximum de secteurs

utilisables (et donc la résolution angulaire) seraient inversement proportionnels au nombre de LED utilisées sur chaque bandeau. Ce phénomène s'explique aisément en étudiant de plus près le protocole de communication SPI permettant au Raspberry Pi de commander les bandeaux de LED :

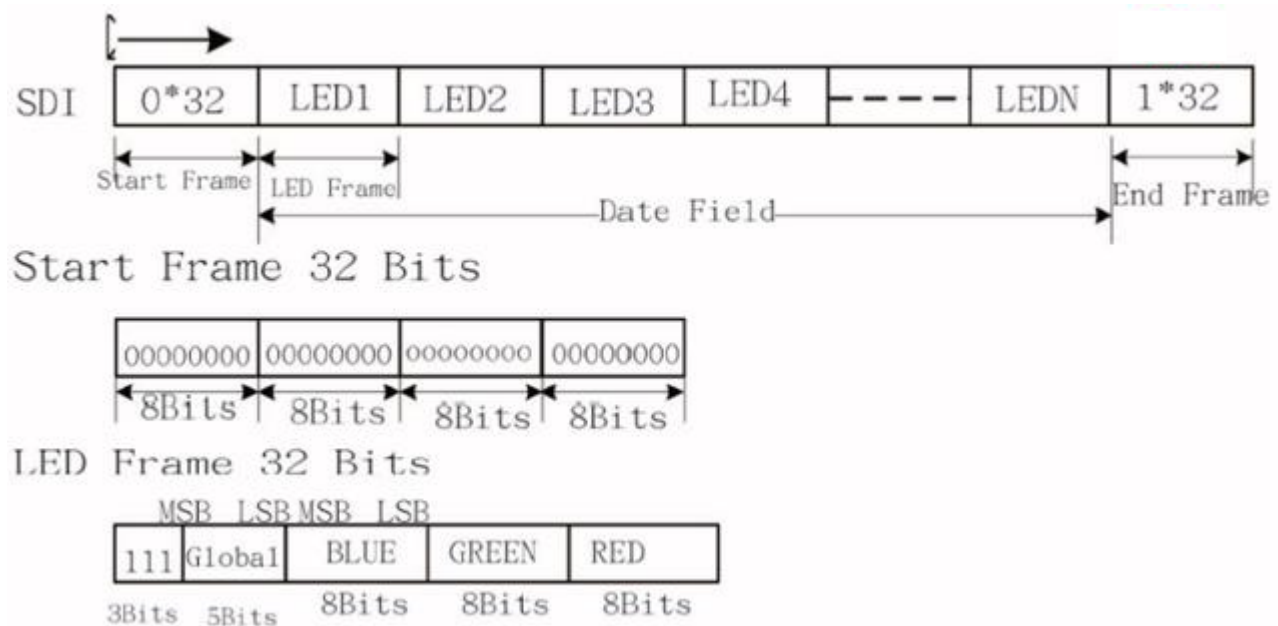
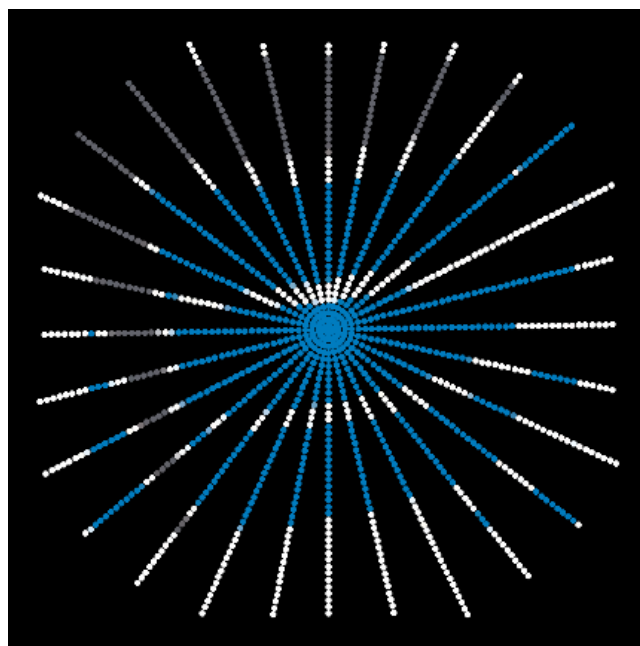


Figure 31 : Description des trames de données attendues par les bandeaux APA102

On observe en effet que la longueur d'une trame de données dépend directement du nombre de LED à commander. Or, la transmission d'une trame prend un certain temps, temps qui sera donc aussi proportionnel au nombre de LED à commander. Or comme nous l'avons vu, c'est en partie ce temps qui définit l'angle d'un secteur et donc le nombre de secteurs équiangulaires qui nous donne la résolution angulaire de l'afficheur.

Cela crée donc un dilemme entre la résolution radiale (nombre de LED utilisées) et la résolution angulaire, qu'il faut résoudre en trouvant un équilibre entre les deux résolutions. Après de nombreux tests, nous avons déterminé que cet équilibre est optimal lorsqu'on utilise 35 LED par bandeau, nous obtenons alors 16 secteurs d'affichage par 180° soit 32 en tout. Ce qui donne le résultat suivant sur le simulateur :



Nous garderons donc ces valeurs pour nos tests suivants.

3.3.2.3 Code final permettant d'afficher une image sur l'hélice

Voici le code final, commenté, permettant d'afficher une image grâce à l'hélice :

```
int main()
{
    signal(SIGINT, handle_sigint); //handling SIGINT signal

    wiringPiSetup(); //initialization of wiringPi
    pinMode(SENSOR_PIN, INPUT); //set endstop sensor as input
    pinMode(MOTOR_PIN, OUTPUT); //set motor relay as output
    startMotor(); //start motor

    //initialize both LED strips
    struct APA102 *strip = APA102_Init(NB_LEDS_PER_STRIP, 0);
    struct APA102 *strip2 = APA102_Init(NB_LEDS_PER_STRIP, 1);

    //initialize empty frame to shutdown LED strips
    struct APA102_Frame *offFrame = APA102_CreateFrame(0x00, 0x00, 0x00, 0x00);
    writeFrame(strip, strip2, offFrame);

    strip = APA102_Init(NB_LEDS_PER_STRIP, 0);
    strip2 = APA102_Init(NB_LEDS_PER_STRIP, 1);

    //initialize 2 3-
dimensionnal arrays containing all color points to display in the form [sector][rad
ial][color] for each strip
    uint8_t colorsForStrip[NB_SECTORS][NB_LEDS_PER_STRIP][3];
    uint8_t colorsForStrip2[NB_SECTORS][NB_LEDS_PER_STRIP][3];

    int colorPoints[][5] = COLOR_POINTS; //initialize colorPoints array with va
lues from .h header file generated by Python script
    for (int i = 0; i < NB_COLOR_POINTS; i++)
    {
        uint8_t sectorIndex = colorPoints[i][0];
        int rIndex = colorPoints[i][1];
        if (rIndex >= 0)
        {
            colorsForStrip[sectorIndex][rIndex][0] = colorPoints[i][2];
            colorsForStrip[sectorIndex][rIndex][1] = colorPoints[i][3];
            colorsForStrip[sectorIndex][rIndex][2] = colorPoints[i][4];
        }
        else
        {
            rIndex = -rIndex;
            colorsForStrip2[sectorIndex][rIndex][0] = colorPoints[i][2];
            colorsForStrip2[sectorIndex][rIndex][1] = colorPoints[i][3];
            colorsForStrip2[sectorIndex][rIndex][2] = colorPoints[i][4];
        }
    }
}
```

```

while (1) //repeated until SIGINT is triggered
{
    //wait until endstop sensor detects origin
    if (digitalRead(SENSOR_PIN))
    {
        for (uint8_t sector = 0; sector < NB_SECTORS; sector++)
        {
            //for each sector, send frames to both LED strips
            APA102_FillWithDifferentColors(strip, colorsForStrip[sector]);
            APA102_FillWithDifferentColors(strip2, colorsForStrip2[sector]);
        }
        writeFrame(strip, strip2, offFrame); //if rotation is incomplete, shutdown both strips
    }
}
}

```

Les fonctions et macro associées à ce code sont disponibles en [annexe 2](#).

3.4 Architecture logicielle globale

Finalement, voici un schéma reprenant l'ensemble des traitements effectués de l'image d'entrée à son affichage grâce à l'hélice :

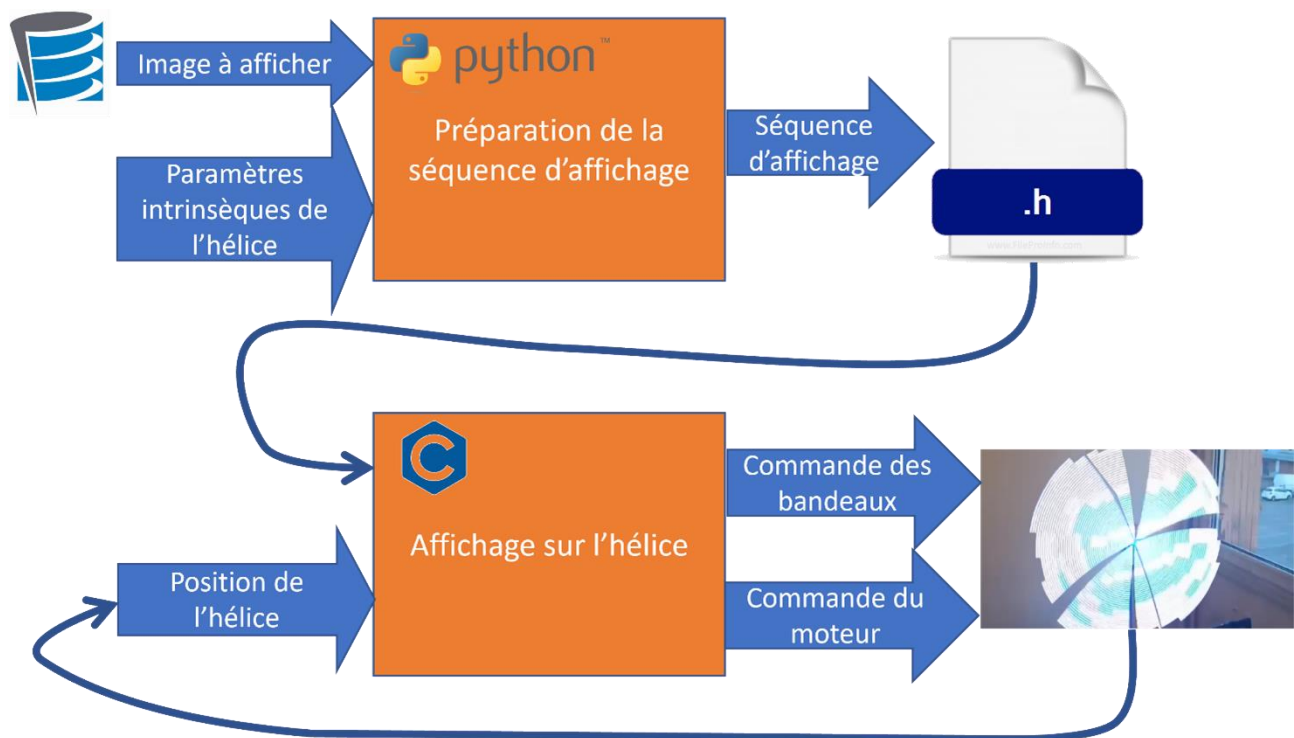


Figure 32 : Traitements réalisés afin d'afficher une image sur l'hélice

3.5 Interface graphique de commande et autres fonctionnalités ajoutées

L'architecture logicielle présentée ci-dessus que nous avons imaginée pour l'hélice la rendait capable, dans un premier temps, d'afficher une image (voir [la partie suivante](#)) et donc de répondre à la problématique initiale. Toutefois, il fallait pour cela envoyer plusieurs commandes à l'afficheur, ce qui était plutôt fastidieux voire impossible pour un utilisateur novice en informatique. Nous avons donc fait en sorte

que ces processus puissent être déclenchés par une interface de commande et communiquer entre eux malgré qu'ils utilisent des langages et des technologies différentes (langage compilé vs langage interprété).

Afin d'automatiser les différentes étapes du processus d'affichage d'une image présentées ci-dessus, nous avons réalisé une interface de commande sous la forme de d'une page Web. En effet, le prototype de notre affichage étant destiné à être utilisé pour des démonstrations, il était nécessaire d'offrir une interface homme-machine la plus simple et la plus intuitive possible. Pour cela, nous avons utilisé l'un des principaux Framework web développés en Python : Flask.

Flask permet de créer un serveur Web hébergeant une application web, généralement sous la forme d'une seule page web. Ayant déjà développé la plupart du code de l'hélice en Python (notamment le programme haut-niveau de [séquençage de l'image](#)), il nous semblait particulièrement adapté d'utiliser Flask pour créer cette interface de commande.



L'interface de commande finalement réalisée est la suivante :

Sur l'unique page (accessible depuis un navigateur sur un appareil connecté au même réseau local que l'hélice), l'utilisateur est invité à importer l'image à afficher puis à spécifier les paramètres d'affichage (résolution radiale et angulaire) :

The screenshot shows a web browser window displaying the 'Hélice holographique' interface. At the top, the title 'Hélice holographique' is followed by a logo of a propeller with two circular arrows. Below the title is a subtitle 'Bienvenue sur l'interface de commande de l'hélice holographique'. A section titled 'Pour afficher une image sur l'hélice :' contains four numbered steps: 1. 'Importer l'image à afficher : Parcourir...' with a file selection button and the text 'Aucun fichier sélectionné.'; 2. 'Choisir le nombre de LED par pale (résolution radiale) : 35' with a dropdown menu; 3. 'Choisir le nombre de secteurs (résolution angulaire) : 16' with a dropdown menu; 4. 'Prévisualiser le rendu' with a button. At the bottom, a footer line reads 'Interface réalisée par Valentin Mercy dans le cadre de l'UV T052 de l'Université de Technologie de Belfort-Montbéliard.'

Figure 33 : Interface de commande de l'afficheur - avant prévisualisation

Lorsque l'utilisateur a cliqué sur « Prévisualiser le rendu », l'image est séquencée, le programme d'affichage est recompilé et la prévisualisation de l'image séquencée apparaît sur l'interface :

Hélice holographique

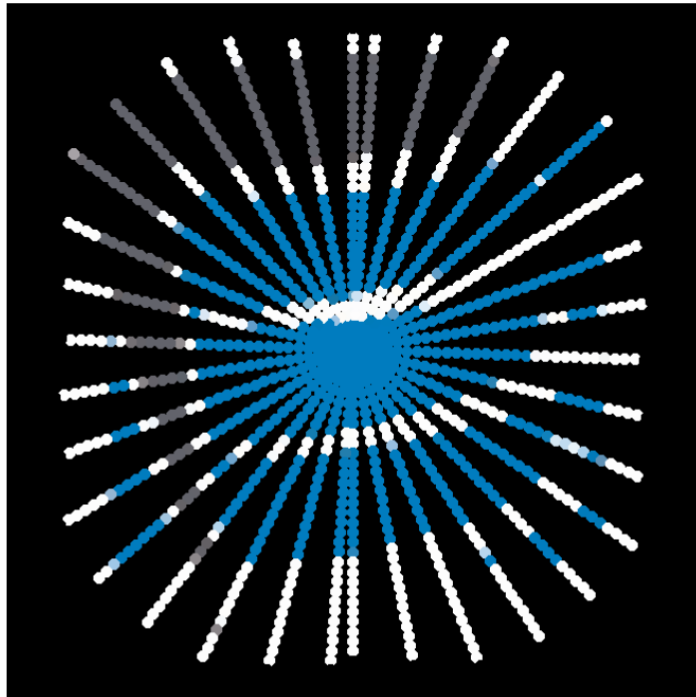


Bienvenue sur l'interface de commande de l'hélice holographique

Image importée et traitée avec succès, prévisualisation en cours

Pour afficher une image sur l'hélice :

1. Importer l'image à afficher : Aucun fichier sélectionné.
2. Choisir le nombre de LED par pale (résolution radiale) :
3. Choisir le nombre de secteurs (résolution angulaire) :
4.
5. Vérifier le rendu :



6. Démarrer/arrêter l'hélice :

Interface réalisée par [Valentin Mercy](#) dans le cadre de l'UV T052 de l'[Université de Technologie de Belfort-Montbéliard](#).

Figure 34 : Interface de commande de l'afficheur – après prévisualisation

L'utilisateur accède également à deux boutons permettant respectivement de démarrer et d'arrêter l'hélice. Le code de l'interface de commande est disponible en [annexe 3](#).

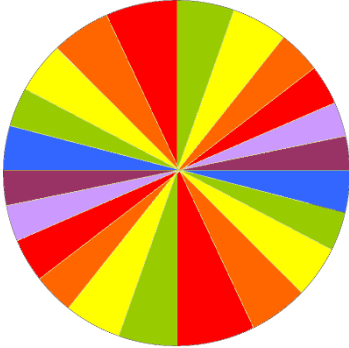
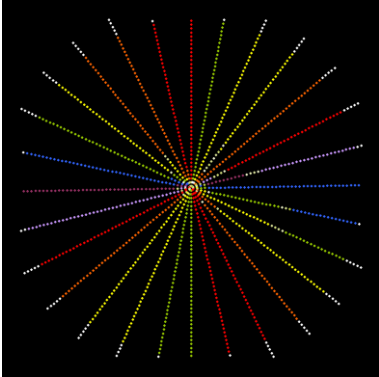
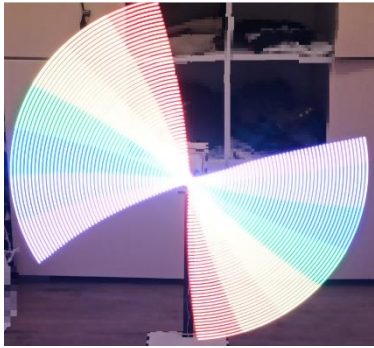

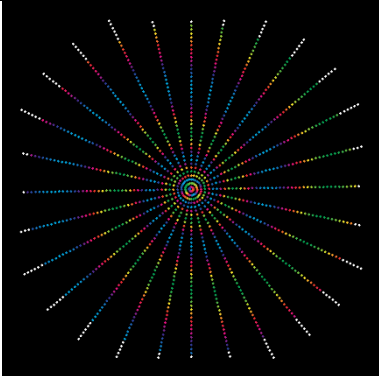
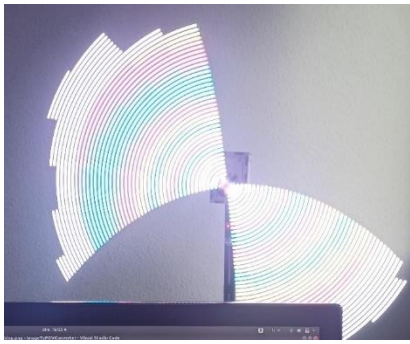
En dehors de l'interface de commande, nous avons ajouté du code à notre projet dans plusieurs autres objectifs :

- Un programme en C nommé *flash_strip.c* permettant de faire clignoter l'hélice entière de plusieurs couleurs afin d'alerter l'utilisateur en cas d'erreur/succès/avertissement ou avant que l'hélice ne démarre (afin de le préparer au danger que représente l'hélice)
- Un code en Python permettant de commander le [buzzer piézoélectrique ajouté au montage électronique](#), dans le même objectif que *flash_strip.c*

- Un Makefile permettant respectivement de compiler et recompiler⁵ automatiquement les programmes *flash_strip.c* et *displayer.c* (ce dernier étant la version finale du processus en charge d’afficher une image séquencée sur l’hélice) pour obtenir deux fichiers binaires exécutables
- Des sous-processus ajoutés au programme principal final *main.py* (incluant le serveur Web) permettant notamment d’exécuter les binaires issus de la compilation grâce au Makefile

4 Résultat

Finalement, nous avons obtenu un résultat satisfaisant de notre point de vue. L’afficheur ainsi conçu est capable d’afficher des images simples, permettant de ressentir l’effet d’un hologramme. Les différentes photographies ci-dessous montrent l’afficheur en fonctionnement. Toutefois, nous rappelons que l’appareil photo utilisé pour les capturer n’étant pas sujet au phénomène de persistance rétinienne, le rendu ne reflète pas ce que nous avons vu en tant qu’observateurs humains lors de leur capture : nous voyions bien à ce moment l’image dans son intégralité, et pas seulement des secteurs comme ci-dessous.

<u>Image d’entrée</u>	<u>Prévisualisation fournie par notre simulateur POV Simulator</u>	<u>Image affichée par l’afficheur holographique</u>
		
		

⁵ En effet, le programme *displayer.c* nécessite d’être recompilé à chaque fois que l’image à afficher change. C’est une conséquence du [choix](#) que nous avons fait d’utiliser un langage compilé (plus bas niveau) pour le processus d’affichage d’une image séquencée.

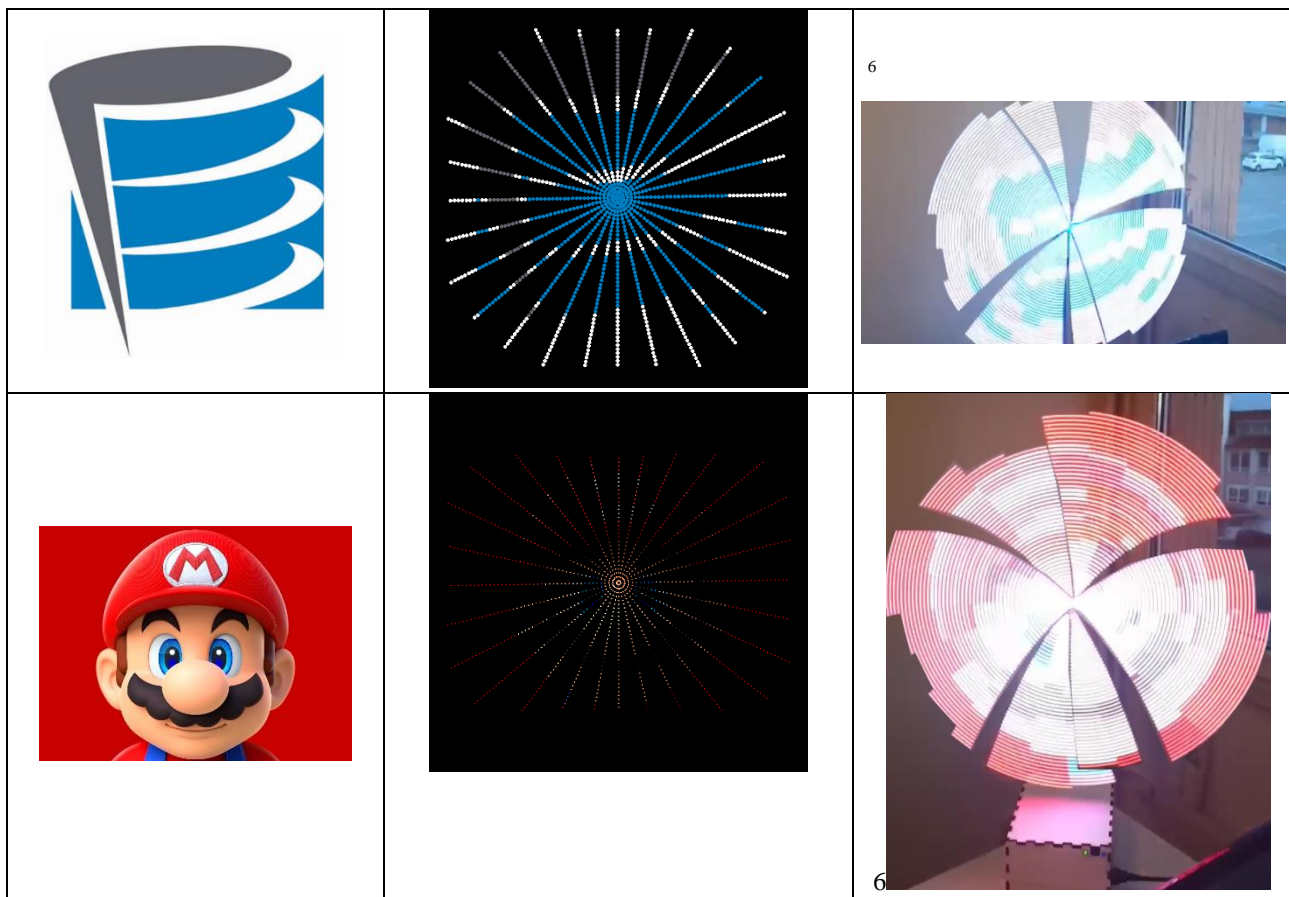


Tableau 1 : Exemples d'images affichées par notre prototype d'afficheur à hélice holographique

Une vidéo de l'hélice en fonctionnement est disponible à l'URL suivant : <https://youtu.be/Cvduv8uLtQo>

5 Organisation et fonctionnement du binôme

Pour mener ce projet à son aboutissement, nous avons employé plusieurs méthodes et outils de collaboration. Tout d'abord, nous avons commencé par nous réunir afin de dégrossir le sujet du projet et de réfléchir aux problèmes les plus évidents en essayant d'y trouver plusieurs solutions possibles. Une fois ces solutions potentielles identifiées, nous avons expérimenté plusieurs d'entre-elles afin d'en retenir la plus pertinente (voir les parties [Motorisation](#) et [Transmission du contact en rotation](#)).

Au niveau du code informatique, nous avons mis en place un dépôt Git hébergé sur Github à l'URL suivant : https://github.com/vmercy/T052_helice_holographique. Pour faciliter la communication au sein du binôme, nous avons créé une conversation sous Microsoft Teams.

Enfin, nous avons observé la répartition des tâches suivante :

Sandra :

- Participation à la phase d'idéation et d'identification des problèmes et solutions
- Construction physique du prototype et câblage électronique
- Validation du fonctionnement du prototype
- Rédaction des livrables

⁶ Ces deux dernières images ont été issues d'une fusion de plusieurs images extraites de vidéos, afin de montrer un résultat plus proche de ce que nous avons perçu en tant qu'observateurs humains

Valentin :

- Participation à la phase d'idéation et d'identification des problèmes et solutions
- Programmation du simulateur POVSimulator en Python
- Modélisation 3D du prototype sous SolidWorks
- Conception du circuit électronique du prototype et câblage électronique
- Programmation de l'afficheur (codes C et Python, interface de commande)
- Rédaction des livrables

6 Conclusion

Pour conclure, dans ce projet, après avoir utilisé un nano-ordinateur de type Raspberry Pi Zero W, un bandeau de LED adressable de densité 144 LED/m, une bague collectrice pleine, un moteur d'imprimante papier, une alimentation de PC, un capteur de fin course de type « barrière lumineuse », et malgré les difficultés rencontrées telles que la transmission du contact électrique sur élément tournant ou le capteur de prise d'origine, nous avons obtenu une hélice holographique fonctionnelle.

Elle pourrait encore être améliorée. On peut facilement imaginer pouvoir afficher des vidéos ou des images animées. D'autres améliorations pourraient aussi venir de la réduction du bruit ou de la protection de l'hélice des mains des usagers. On pourrait aussi choisir un autre bandeau de LED qui utiliserait un autre protocole que SPI et permettrait alors d'obtenir une meilleure résolution angulaire. Avec cette nouvelle solution et un autre moteur et donc une vitesse de rotation plus importante on pourrait obtenir une image plus fluide qui annulerait l'effet visuel de clignotement actuellement observable sur l'afficheur.

Enfin, nous rappelons que notre code source est intégralement disponible sur notre dépôt Git à l'URL suivant : https://github.com/vmercy/T052_helice_holographique

7 Bibliographie

Burdekin, R. (2015, Janvier). Pepper's Ghost At The Opera. *Theatre Notebook*, 69(3), pp. 152-164.

Récupéré sur

[https://www.researchgate.net/publication/295113791 PEPPER'S GHOST AT THE OPERA/link/5969e59e0f7e9b8091944444/download](https://www.researchgate.net/publication/295113791_PEPPER'S_GHOST_AT_THE_OPERA/link/5969e59e0f7e9b8091944444/download)

Mayer, N. (2022, janvier 2). *Hologramme : qu'est-ce que c'est ?* Récupéré sur futura-sciences.com:

<https://www.futura-sciences.com/sciences/definitions/physique-hologramme-16083/>

8 Annexes

8.1 Annexe 1 : code Python des fonctions permettant de créer la séquence d’affichage

```
from PIL import Image
from PIL import ImageDraw
import json
from math import ceil, pi, sin, cos

def renderPickedPointsPreview(filename, pickedColorPoints, ellipsesDiameter):
    """Renders an image containing ellipses at picked points positions with the picked color

    Args:
        filename (string): The rendered image output filename
        pickedColorPoints (array): The list of picked point colors
        ellipsesDiameter ([type]): The diameter of each ellipse representing a picked color point
    """
    maxSizingColorPoint = max(pickedColorPoints, key=(lambda x: max(x[0][1])))

    size = max(maxSizingColorPoint[0][1])
    size += ellipsesDiameter
    size = ceil(size)
    outputImg = Image.new('RGB', (size, )*2, (0, 0, 0))
    draw = ImageDraw.Draw(outputImg)
    draw.rectangle([(0, 0), (size, )*2], 'black')
    for pickedColorPoint in pickedColorPoints:
        upperLeftCornerCoordinates = (
            pickedColorPoint[0][1][0]-
            ellipsesDiameter/2, pickedColorPoint[0][1][1]-ellipsesDiameter/2)
        bottomRightCornerCoordinates = (
            upperLeftCornerCoordinates[0]+ellipsesDiameter, upperLeftCornerCoordinates[1]+ellipsesDiameter)
        draw.ellipse([upperLeftCornerCoordinates,
            bottomRightCornerCoordinates], fill=pickedColorPoint[1])

    outputImg.save(filename)

def pickColors(pickingPoints, subjectImage):
    """Pick pixel colors on image

    Args:
        pickingPoints (array): Array of tuples containing picking points coordinates
        subjectImage (PIL.Image): The image where colors must be picked

    Returns:
```

```

        Array: List of picked colors in the form (((theta,rIndex),(x,y)),(r,g
,b)))
    """
    pickedColorPoints = []
    for pickingPoint in pickingPoints:
        pixel = subjectImage.getpixel(pickingPoint[1])
        pickedColorPoints.append((pickingPoint, pixel))
    return pickedColorPoints

def computePickingPointsPositionsOnDiametralLine(pickingAreaDiameter, nbPoints, angle):
    """Computes the coordinates of aligned points where color must be picked
    on image

    Args:
        pickingAreaDiameter (int): The diameter of the area where points must
        be picked
        nbPoints (int): The expected number of points
        angle (float): The angle of the diameter line where holes must be all
        igned

    Returns:
        array: Array of (x,y) tuples containing holes center positions
    """
    spaceBetweenPoints = pickingAreaDiameter / nbPoints
    radialCoordinates = []
    if(nbPoints % 2): # number of points is odd, so we place the first one o
n the center
        radialOrigin = 0
        nbPoints -= 1
    else:
        radialOrigin = spaceBetweenPoints/2
        nbPoints -= 2
        radialCoordinates.append(radialOrigin)
    for i in range(0, nbPoints//2):
        radialCoordinates.append(radialOrigin+(i+1)*spaceBetweenPoints)
    pointsPositions = []
    for radialIndex, radial in enumerate(radialCoordinates):
        pointsPositions.append(((angle, radialIndex), (pickingAreaDiameter /
        2+radial*sin(angle), pickingAreaDiameter/2+rad
        ial*cos(angle))))
        pointsPositions.append(((angle, -radialIndex), (pickingAreaDiameter /
        2-radial*sin(angle), pickingAreaDiameter/2-
        radial*cos(angle))))
    return pointsPositions

# def addTrailArcs

```

```

def getPickingPoints(pickingAreaDiameter, nbPointsPerLine, angleStep, angleMin=0, angleMax=180):
    """Builds

    Args:
        pickingAreaDiameter (int): The diameter of the area where points must
        be picked
        nbHoles (int): The expected number of holes in each diametral line
        angleStep (int): The angle between two consecutive diametral lines
        angleMin (int, optional): The angle of the first line of holes. Defaults to 0.
        angleMax (int, optional): The angle of the last line of holes. Defaults to 180.

    Returns:
        array: Array of picking points
    """
    points = []
    for i in range(angleMin, angleMax, angleStep):
        angle_rad = pi/180.0 * i
        points += computePickingPointsPositionsOnDiametralLine(
            pickingAreaDiameter, nbPointsPerLine, angle_rad)
    if(nbPointsPerLine % 2): # if the number of points per line is odd then
        we add the center point once
        points.append(((0, 0), (pickingAreaDiameter//2,)*2))
    return points

def savePickedColors(filename, pickedColors, nbSectors):
    """Saves picked colors to json file

    Args:
        filename (string): name of JSON file to write
        pickedColors (array): array containing picked colors
        nbSectors (int): number of expected sectors

    Returns:
        array: List of cleaned picked colors in the form [(thetaIndex, radialIndex),(r,g,b)]
    """
    cleanedPickedColors = []
    lastTheta = 0
    thetaIndex = 0
    for pickedColor in pickedColors:
        if pickedColor[0][0][0] != lastTheta:
            thetaIndex += 1
            lastTheta = pickedColor[0][0][0]
            cleanedPickedColors.append(
                (thetaIndex, pickedColor[0][0][1], pickedColor[1][0], pickedColor[1][1], pickedColor[1][2]))
    cleanedPickedColors.sort(key=lambda x: (x[0], x[1]))

```

```

fileContent = '#define COLOR_POINTS \\'+'\\n{\\n'
for cleanedPickedColor in cleanedPickedColors:
    fileContent += '{ %i, %i, %i, %i, %i },\\n' % cleanedPickedColor
fileContent += '}'
with open(filename, 'w') as outfile:
    outfile.write(fileContent)
return cleanedPickedColors

```

8.2 Annexe 2 : Code C permettant d'afficher une image avec l'hélice – fonctions et macro

```

#define SENSOR_PIN 2
#define MOTOR_PIN 22
#define MAX_LEDS_PER_STRIP 48
#define BUZZER_PIN 3
#define JSON_FILENAME "logo_utbm.json"

#include "logo_utbm.h"

#define NB_SECTORS 16
#define NB_LEDS_PER_STRIP 35

#define NB_COLOR_POINTS (NB_SECTORS * NB_LEDS_PER_STRIP * 2)

void writeFrameAllStrips(struct APA102 *strip, struct APA102 *strip2, uint8_t
colorsStrip[][3], uint8_t colorsStrip2[][3])
{
    for (uint8_t i = 0; i < NB_LEDS_PER_STRIP; i++)
    {
        APA102_FillWithDifferentColors(strip, colorsStrip);
        APA102_FillWithDifferentColors(strip2, colorsStrip2);
    }
}

void writeFrame(struct APA102 *strip, struct APA102 *strip2, struct APA102_Fr
ame *frame)
{
    APA102_Fill(strip, frame);
    APA102_Fill(strip2, frame);
}

void handle_sigint()
{
    struct APA102 *strip = APA102_Init(48, 0);
    struct APA102 *strip2 = APA102_Init(48, 1);
    struct APA102_Frame *offFrame = APA102_CreateFrame(0x00, 0x00, 0x00, 0x00);
    writeFrame(strip, strip2, offFrame);
    digitalWrite(MOTOR_PIN, LOW);
}

void startMotor()

```

```
{
    digitalWrite(MOTOR_PIN, HIGH);
}
```

8.3 Annexe 3 : Code de l'interface web de commande de l'afficheur

```
startSequence()
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.secret_key = os.environ.get("flask_secret")

compileFlashStripIfNotAlreadyDone()

@app.route('/uploads/<path:filename>')
def download_file(filename):
    return send_from_directory('preview', filename, as_attachment=True)

angular_res_input = None
radial_res_input = None

@app.route("/", methods=['GET', 'POST'])
def home():
    global RUN_ALLOWED, angular_res_input, radial_res_input
    if request.method == 'POST':
        if RUN_ALLOWED:
            if request.form.get('command') == 'run_prop':
                startDisplayer(angular_res_input, radial_res_input)
                flash('Hélice démarrée avec succès', 'success')
                return render_template('index.html', preview_url="preview/preview.png",
RUN_ALLOWED = RUN_ALLOWED, startBtnDisabled = True, stopBtnDisabled = False)
            elif request.form.get('command') == 'stop_prop':
                stopDisplayer(subProcessToKill=displayerSubprocess)
                flash('Hélice arrêtée avec succès', 'info')
                return render_template('index.html', preview_url="preview/preview.png",
RUN_ALLOWED = RUN_ALLOWED, startBtnDisabled = False, stopBtnDisabled = True)
            RUN_ALLOWED = False
            angular_res_input = int(request.form.get('angular_resolution'))
            radial_res_input = int(request.form.get('radial_resolution'))
            if not isinstance(angular_res_input, int) :
                flash('La résolution angulaire saisie en 3 doit être un entier', "warning")
                return redirect(request.url)
            if angular_res_input > ANGULAR_RESOLUTION_BOUNDS['max'] or angular_res_input <
ANGULAR_RESOLUTION_BOUNDS['min']:
                flash('La résolution angulaire doit être comprise entre %i
et %i'%(ANGULAR_RESOLUTION_BOUNDS['min'],
ANGULAR_RESOLUTION_BOUNDS['max']), "warning")
                return redirect(request.url)
            if not isinstance(angular_res_input, int) :
                flash('La résolution radiale saisie en 2 doit être un entier', "warning")
                return redirect(request.url)
```

```

    if radial_res_input > RADIAL_RESOLUTION_BOUNDS['max'] or radial_res_input <
RADIAL_RESOLUTION_BOUNDS['min']:
        flash('La résolution radiale doit être comprise entre %i
et %i'%(RADIAL_RESOLUTION_BOUNDS['min'],
RADIAL_RESOLUTION_BOUNDS['max']), "warning")
        return redirect(request.url)
    #check image input file
    if 'image_to_display' not in request.files:
        flash('Aucun fichier envoyé', "warning")
        return redirect(request.url)
    file = request.files['image_to_display']
    if file.filename == '':
        flash('Aucun fichier sélectionné', warning)
        return redirect(request.url)
    if not isFilenameAllowed(file.filename):
        flash('Fichier image invalide, seuls les fichiers au format .png, .jpg
ou .jpeg sont acceptés', 'warning')
        return redirect(request.url)
    if file and isFilenameAllowed(file.filename):
        filename = secure_filename(file.filename)
        fullpath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(fullpath)

    # Configuration variables
    nbEllipsesPerDiametralline = 2*radial_res_input
    imageFileName = fullpath
    outputFileName = 'static/preview.png'
    nbSectors = angular_res_input #number of displaying sectors
    angleStep = int(180/nbSectors)
    zoomFactor = 90 #zoom factor in percent

    sourceImg = Image.open(imageFileName)
    sourceImg.convert('RGBA')
    imgMaxSize = max(sourceImg.size)
    baseImgSize = imgMaxSize*100//zoomFactor
    centeredImage = Image.new('RGB', (baseImgSize,)*2, 'black')

    centeredImage.paste(sourceImg, ((baseImgSize-
sourceImg.size[0])//2, (baseImgSize-sourceImg.size[1])//2))

    pickingPoints = getPickingPoints(baseImgSize,
nbEllipsesPerDiametralline, angleStep)
    pickedColors = pickColors(pickingPoints, centeredImage)

    savePickedColors('./displayer/sequenced_image.h', pickedColors,
nbSectors)
    renderPickedPointsPreview(outputFileName, pickedColors, 10)

    flash('Image importée et traitée avec succès, prévisualisation en
cours', 'warning')

```



```
        compileDisplayer()

        RUN_ALLOWED = True

        return render_template('index.html',preview_url="preview/preview.png",
RUN_ALLOWED = RUN_ALLOWED)
    return render_template('index.html')

if __name__=="__main__":
    app.debug = True
    app.run(host='0.0.0.0')
```

Mots clefs

Hologramme, Affichage holographique, Hélice tournante, hélice holographique, persistance rétinienne, Persistance Of Vision

Résumé

Le présent rapport présente le projet réalisé dans le cadre de l'unité de valeur TO52 « projet de développement informatique ». L'objectif de ce projet a été de concevoir et de réaliser un outil de projection holographique de type hélice rotative. Ceci comprend la conception mécanique et électronique du prototype et la programmation de l'affichage. Ce dispositif utilise une caractéristique intéressante pour nous de la vision humaine : la persistance rétinienne. Il était attendu que le prototype soit suffisamment abouti et simple d'utilisation pour pouvoir être utilisé en tant que support de communication innovant lors d'événements ouverts au public.

Plusieurs solutions étaient envisageables pour l'élaboration de ce projet. Nous nous sommes d'abord penché sur l'aspect informatique du projet, pour cela nous nous sommes limités à la puissance de calcul d'un nano-ordinateur de type Raspberry Pi Zero W. Puis nous avons choisi un bandeau de LED comme source lumineuse.

Concernant le dimensionnement, soit la taille du bandeau de LED composant l'hélice et moteur utilisé pour entraîner cette dernière en rotation à haute vitesse, nous nous sommes appuyés sur un simulateur développé par nos soins permettant de prévisualiser l'image qu'affichera notre hélice une fois construite.

Concernant la motorisation, plusieurs choix s'offraient à nous mais nous avons utilisé un moteur d'imprimante papier qui était la solution la plus convaincante. Afin que le moteur soit assez puissant pour démarrer lui-même l'hélice, il était alors nécessaire de ne pas coupler directement l'hélice à l'arbre moteur mais d'intercaler un engrenage permettant de démultiplier le couple.

L'un des principaux défis que nous avons dû relever dans ce projet a concerné l'approvisionnement en électricité de l'hélice en rotation : comment transmettre un courant continu et des signaux de commande à un élément tournant ? Ce rapport présente d'autres questions auxquelles nous avons répondu par des solutions techniques notamment au niveau du montage électronique conçu pour ce prototype, comme le choix de l'alimentation et du capteur de prise d'origine.

Une fois l'afficheur assemblé et testé pour être sûrs de la fiabilité mécanique et électronique du montage, nous avons pu entreprendre sa programmation.

Nous avons alors utilisé le langage Python pour générer la séquence d'affichage puis le langage C, plus rapide que Python car compilé, pour l'affichage sur l'hélice.

Enfin, grâce à une interface web, un utilisateur novice en informatique peut tout à fait mettre en fonctionnement l'hélice pour afficher l'image de son choix.