Valentin Lupa

Louis Malbranque

Guillaume Vacheron

Alexia Verpoort

***Projet M1 S2 :***

***Amélioration d’une Table*** ***Rotative***



# Remerciements

Nous souhaitons également remercier Nicolas Lissarague, commanditaire du projet, pour ses réponses à nos questions et son aiguillage au cours de ce projet. Et égalementt pour son encadrement tout au long des six semaines.

Enfin, nous remercions Christophe Denoyelle pour la gestion des commandes, et la gestion de la création des PCB.

# Sommaire

# Introduction

Ce projet est une continuation du projet du premier semestre. Monsieur Lissarague avait besoin de finaliser la conception hardware du projet mais il souhaitait également une application mobile pouvant contrôler son projet. Le groupe a été modifié en fonction des volontés de projet du groupe initial. Deux nouveaux membres ont été intégrés au groupe, Louis Malbranque et Guillaume Vacheron, tous deux intéressés par l’aspect très pratique et très utile du projet, le but étant d’avoir un produit fini pouvant être fonctionnel et reproductible.

Le but du projet initial proposé notamment au premier semestre était d’améliorer un système permettant la rotation d’une table sur laquelle repose un objet destiné à la photogrammétrie. Cet objet doit être pris en photo selon plusieurs angles en fonction des arrêts de la table afin de pouvoir par la suite produire une modélisation en 3D de l’objet.

L’objectif principal du projet était donc d’obtenir une table rotative entièrement fonctionnelle dont l’action sur le moteur était commandée directement par une application Android. Il fallait également corriger les problèmes techniques pouvant survenir lors du déroulement et donner des versions finalisées d’un PCB. Ce projet avait donc une priorité sur le résultat.

# Système de base

Le système sur lequel nous avons travaillé est la continuité d’un projet mené au premier semestre : avant d’aborder les différentes améliorations au système, nous allons commencer par expliquer notre base de départ.

SCHEMA

Nous sommes donc partis d’un système qui était découpé en 2 parties, une partie boitier de commandes, transmettant les informations aux différents périphériques que ce soit la table elle-même ou les appareils photos, et une partie moteur permettant la réception des données et le contrôle d’un driver moteur sur un moteur pas à pas. Les 2 parties distinctes imposant notamment la présence d’un câble entre le moteur et le boitier de commande.

Au niveau du hardware, chacune des deux parties étaient contrôlées par un Arduino (uno ou nano) et les deux parties pouvaient communiquer entre elles par le biais du module NRF24 par ondes radio. La partie boitier de commande recevait des informations d’une application bluetooth serial communiquant sur le port série de l’Arduino. L’utilisateur transmettait les informations suivantes : l‘accélération voulue pour la rotation du moteur, la vitesse, le nombre de photos à prendre, le nombre de pas moteur, le degré désiré et la durée d’une pause de moteur. Ces données étaient envoyées par bloc et séparées par une virgule. Elles étaient ensuite affichées sur le LCD puis transférée au bloc moteur. Le bloc moteur réalisait alors ses rotations et à chaque pause, renvoyait un signal au bloc commande pour que celui-ci active ses 3 relais en même temps qui permettent de déclencher les appareils photos.

Au niveau du fonctionnement, un seul sur 3 relais s’activait, un PCB sur deux était fonctionnel, et le boitier de commande manquait parfois un message du moteur, ce qui empêchait le moteur de s’arrêter définitivement après un tour complet.

# I/ Gestion du projet

Au départ du projet, un échange de mails avait permis d’identifier les premières tâches attendues par Monsieur Lissarague et avait permis de commencer à réfléchir de notre côté sur les potentielles options pouvant être ajoutées. Pour des raisons de méthodologie et d’efficience, le groupe de 4 s’est scindé en deux. Une réunion avait été programmée le lundi 18 mars au matin avec Monsieur Lissarague afin de connaitre ses objectifs ainsi que pour discuter les différents points d’amélioration que nous envisagions.

A la fin de la réunion, en fonction des premiers objectifs donnés, les priorités suivantes ont été déterminées :

F0 (objectifs incontournables) :

* Création d’une application fonctionnelle gérant la connexion par Bluetooth, l’envoi des données par paramétrages et l’envoi des données en temps réel. La différence entre les deux modes étant que le paramétrage ressemble à ce qui existait précédemment avec une rotation dont les pauses dépendent du nombre de photos, et le temps réel étant un mode continu, sans pause, dont la durée dépend de ce que souhaite l’utilisateur.
* Adaptation du code précédent pour pouvoir gérer le temps réel et le paramétrage. (Avant il y avait uniquement le paramétrage).
* Réglage de tous les précédents problèmes de synchronisation pour avoir une table entièrement fonctionnelle.
* Faire des PCB si possible en Chine.

F1 (objectifs nécessaires) :

* Gérer un NEMA23 pour remplacer le précédent moteur qui était un NEMA17 et ainsi permettre l’utilisation de la table pour des objets plus lourds.
* Revoir les composants : remplacer l’arduino par un ESP32 et les relais par des transistors.
* Augmenter le nombre d’appareils photos possibles et ajouter un délai entre le déclenchement de chaque appareil.
* Rajouter des paramètres variables en fonction de la taille du moteur et de la taille de la table afin de connaitre le nombre de pas nécessaires au moteur pour que la table réalise un tour entier.

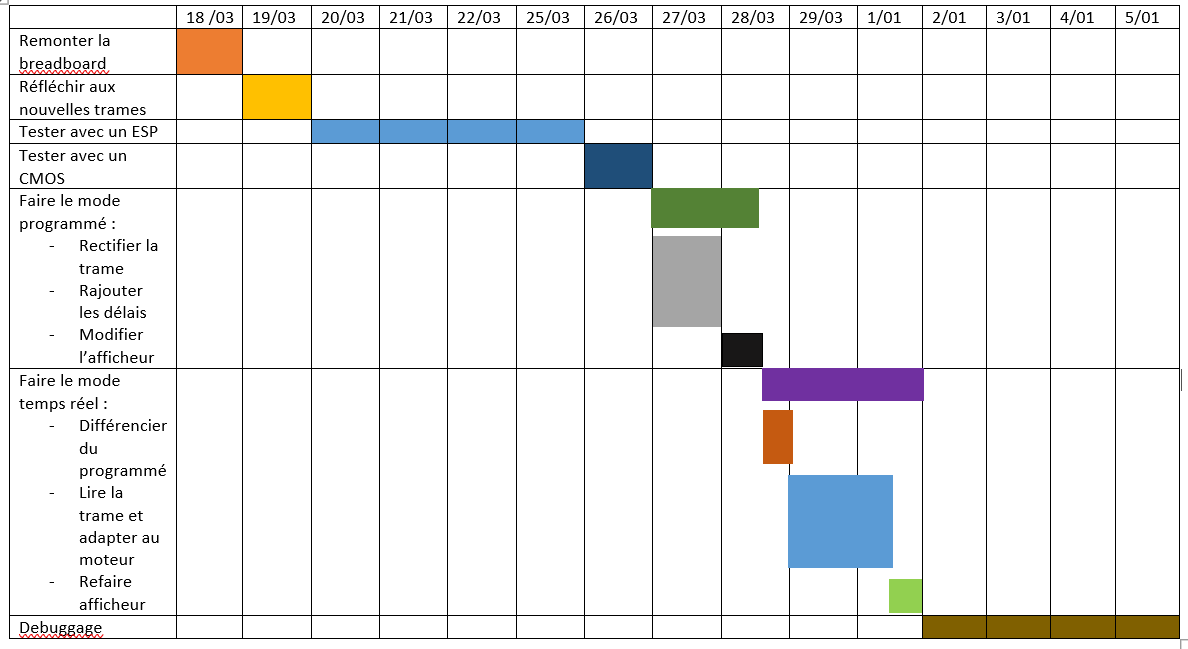
F2 (objectifs souhaitables) :

* Fonctionnalité magnétoscope c’est-à-dire un déplacement entièrement manuel se présentant sous la forme de plusieurs boutons : avance, recul, avance rapide, recul rapide.
* Réfléchir à un déclenchement infrarouge des appareils photo permettant de ne pas avoir de câbles jack qui relient les appareils photos au boitier de commande.

La réflexion autour de ces différents objectifs nous a alors permis d’élaborer un planning de travail pour les 3 premières semaines, afin de s’assurer de respecter les objectifs initiaux avant de commencer la recherche d’améliorations annexes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 18 /03 | 19/03 | 20/03 | 21/03 | 22/03 | 25/03 | 26/03 | 27/03 | 28/03 | 29/03 | 1/04 | 2/04 | 3/04 | 4/04 | 5/04 |
| Modifier le code en orienté objet boitier |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Modifier le code en orienté objet du plateau |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Application android |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Se connecter en Bluetooth avec Android Studio |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Mode programmé |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Mode temps réel |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

***Planning Binôme Louis/Valentin***



***Planning Binôme Guillaume/Alexia***

Cette première esquisse de planning, nous a permis de gérer notre temps au début du projet, nous avons choisi de ne pas se positionner sur les 3 dernières semaines puisque les délais estimés pouvaient être arbitraires, et que les ambitions de fin de projet pouvaient évoluer.

Le planning a été utile pour l’organisation, mais les délais affichés étaient en règle générale assez long, ce qui nous a permis de réfléchir à de potentielles améliorations plus poussées que ce qui était prévu et d’ajouter de nouvelles fonctionnalités suite aux nouvelles discussions avec Monsieur Lissarague.

Concernant l’organisation des réunions et la communication, nous avons choisi de faire un point assez régulier sur les avancées de chacun tous les deux jours, sans oublier l’utilisation des réseaux pour se maintenir au courant en temps réel. La charge de travail pouvait ainsi être répartie plus facilement, afin d’éviter qu’un binôme ou une personne n’ait trop d’objectifs à réaliser en parallèle alors que l’autre binôme avait terminé les siens.

Nous avons également organisé différentes réunions skype avec Monsieur Lissarague, ainsi qu’organisé une réunion physique de mi-projet pour lui faire part de nos avancées et lui faire une démonstration de ce qui était fonctionnel. Cette réunion nous a notamment permis de trouver de nouvelles idées telles que le focus stacking qui sera décrit plus bas. Enfin, les échanges de mail nous ont permis de nous organiser, mais également de clarifier certains points.

# II/ Amélioration de la partie hardware

(SCHEMA BLOC ET SCHEMA ALIMENTATION RAPPORT PREMIER SEMESTRE A REFAIRE)

***1) Réflexion autour de l’utilisation d’ESP32 et des transistors***

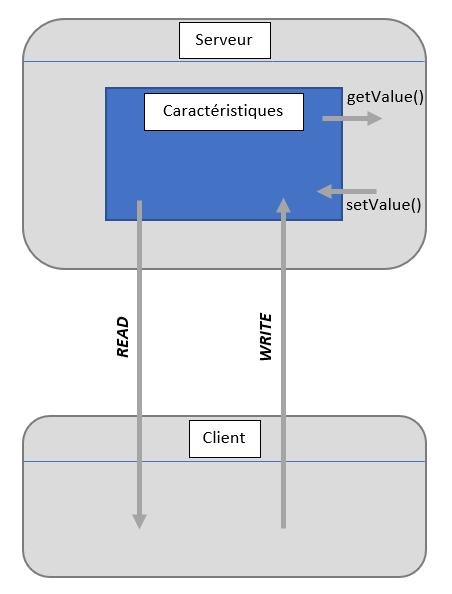
Lors de la première réunion avec Monsieur Lissarague, nous lui avons émis l’idée d’apporter une amélioration supplémentaire au projet afin de supprimer les anciens problèmes de synchronisation qui avaient pu exister au semestre 1. Nous voulions également passer sur microcontrôleur. Afin de maintenir l’accessibilité au code, qui était programmé par l’IDE arduino, et également de simplifier le Hardware, nous avons fait le choix d’utiliser un module de Développement ESP32. L’ESP32 étant un microcontrôleur programmable sous Arduino, le code reste compréhensible pour un habitué des arduino Uno et Nano. L’amélioration majeure apportée par les ESP32 consiste principalement dans l’existence d’une connexion Bluetooth directement intégrée. Notre objectif avec les ESP32 est donc de supprimer les HC05 précédemment présents et également généraliser toute la connexion en mode Bluetooth plutôt que d’avoir un mélange entre connexion par NRF (radio) et HC05.

Afin de gérer la nouvelle connexion, nous avons commencé par adapter la connexion entre une application Android. Le fonctionnement reste identique au HC05, mis à part le nom de la bibliothèque utilisée, les fonctions ayant, elles le même nom, et la logique du code pour obtenir le message ne changeant pas. On a donc une communication Bluetooth qui s’effectue directement sur le Port Série de l’ESP32, le tout étant récupéré caractère par caractère. Le traitement de l’information étant ensuite réalisée en fonction de la trame, il fonctionne en suivant un découpage selon les virgules puis une attribution à chaque variable permettant de contrôler le plateau. Cette étape nous a ainsi permis de supprimer le HC05.

Nous nous sommes ensuite penchés sur le NRF, l’ESP32 et l’utilisation du NRF n’était pas évidente. Premièrement, d’un ESP à l’autre, l’emplacement des Pins permettant la communication, qui est réalisée par SPI, peut varier. Il est ensuite difficile de percevoir l’origine d’une erreur lors de l’utilisation d’un NRF et ceux-ci étant très fragiles, ils peuvent être grillés sans que nous le sachions. Nous avons donc décidé, pour rendre la communication plus homogène et pour utiliser le Bluetooth interne de réaliser une connexion Bluetooth entre les deux ESP32, ce qui permet notamment une meilleure synchronisation.

Nous avons d’abord testé une connexion Bluetooth comme celle qui existe entre un ESP et une application Android. Nous nous sommes alors heurtés à un problème qui est que l’ESP32 est considéré comme un module esclave. C’est-à-dire qu’il ne peut pas faire le choix de s’appairer à un autre module Bluetooth mais doit attendre un ordre pour cela. Il n’est donc pas possible avec le Bluetooth passant par le port série d’appairer les deux ESP32 car il n’y a pas d’ordre d’appairage envoyé.

Afin de pallier ce problème, nous nous sommes penchés sur une connexion BLE (Bluetooth Low Energy) entre ces deux ESP32. Cette connexion est composée d’un serveur (l’esclave) et d’un client (le maitre). Afin de réaliser cette connexion, nous nous sommes d’abord penchés sur la manière dont les informations pouvaient transiter entre client et serveur. Après quelques recherches, nous avons identifié que le serveur hébergeait en réalité des caractéristiques, qui peuvent prendre la forme d’un string (chaine de caractères) qui lui sont accessibles. La particularité de la communication BLE est que le client, selon l’initialisation du serveur, peut lire ou écrire dans ces caractéristiques.



Nous avons donc adapté ce mode de fonctionnement à notre problème. Plutôt que d’envoyer un tableau de données comme c’était précédemment le cas dans notre NRF, nous avons fait transiter les données à travers la lecture et écriture des caractéristiques du serveur. Le boitier de commande a été programmé comme le maitre, puisque c’est seulement lorsqu’il reçoit une trame de l’application qu’il peut ensuite les transmettre au moteur. Le serveur est donc le plateau. Lorsque le boitier veut transmettre des données, il les écrit directement dans ses caractéristiques, s’il identifie qu’elles ont changé, il peut réaliser un déplacement. Pendant ce temps, le client lit en permanence les caractéristiques pour y apercevoir un signal de fin de déplacement pour déclencher les appareils photos.

-> Faire un schéma résumant la situation

Une fois ce mode de communication fonctionnel, nous avons regroupé les deux modes de communications Bluetooth : le programme étant linéaire, l’association des deux n’a pas posé de problème.

Le second changement majeur au niveau du hardware a été le remplacement des relais. Ceux-ci étant trop volumineux, et produisant un bruit à chaque activation, nous avons fait le choix de proposer une nouvelle configuration pour le déclenchement des appareils photo. En premier lieu, nous avons réfléchis à la possibilité de remplacer les relais par des transistors qui est un composant contrôlant le courant ou la tension de sortie en fonction d’une tension qu’on lui applique et qui peut donc être bloquant ou passant.

Après quelques recherches, il nous semblait risqué d’utiliser les transistors pour les appareils photo, nous nous sommes alors orientés vers les optocoupleurs qui sont des composants qui fonctionnent sensiblement de la même manière, qui fournissent donc une tension de sortie en fonction d’une commande. Ces composants étant composé de deux circuits (l’entrée et la sortie) complétement isolés, cela permet de s’assurer qu’aucun courant ne viendra court-circuiter l’appareil photo.

 Nos optocoupleurs fonctionnent donc par intermittence grâce à une pin de commande, qui est mise à l’état haut lorsque l’on souhaite activer un appareil photo, l’optocoupleur est donc capable de fournir une tension de sortie permettant la prise de vue, puis lorsque le pin est remis à l’état bas, l’optocoupleur est de nouveau bloqué et se comporte comme un interrupteur ouvert.

Prise de photo

Pin à 1



Attente

Pin à 0

***2) Concept de Focus Stacking***

Au vu de l’avancée du projet, il nous a été proposé d’ajouter une nouvelle fonctionnalité au projet : le focus stacking. Ce nouvel élément concerne principalement le déclenchement des appareils photos. Il s’agit d’un mode de fonctionnement qui permet à l’utilisateur de prendre plusieurs photos avec le même appareil photo en modifiant la mise au point entre chaque élément.

Ce qui nous a été proposé est de modifier un système existant permettant grâce à un moteur de pivoter la bague de mise en point afin que le moteur soit commandé par le boitier de commande et que la mise au point à chaque prise de vue se fasse automatiquement. Le fonctionnement général serait donc qu’à chaque arrêt du moteur de la table, les appareils connectés se déclenchent comme précédemment mais qu’avant que la table ne recommence sa rotation, la mise au point soit réglée sur chaque appareil pour une nouvelle photo jusqu’au nombre de photo défini, puis que la mise au point soit remise à son origine pour reprendre la rotation de la table.

Pour cela, nous avons réutilisé le système présent pour le moteur de la table en insérant un ESP32, qui soit un serveur en Bluetooth, et qui pourrait donc recevoir les informations du boitier de commande pour actionner le moteur.

Après discussions, il a été déterminé que chaque appareil photo devait au préalablement pouvoir être réglé manuellement pour obtenir les mises au point voulue avec un maximum de 9 photos par appareil supplémentaires. Ce réglage permettant d’obtenir les différents pas nécessaires que le moteur doit réaliser entre chaque photo sur chaque appareil. Les données devaient ensuite être rentrées dans l’application pour que le boitier de commande puisse envoyer les ordres à chaque moteur d’appareil photo au moment voulu et pour toute la durée de la prise de vue.

***3) Modification de la trame d’envoi***

Le système de base ne fonctionnait que selon un mode précis qui consistait à prendre un nombre de photo défini durant un tour. Au cours de la présentation des objectifs du projet par Monsieur Lissarague, celui-ci a émis l’idée d’ajouter différentes fonctionnalités et ainsi différents modes. La table rotative devait tout d’abord fonctionner comme lors du projet du premier semestre, c’est-à-dire programmer un nombre de photos à effectuer par un nombre d’appareils photos donnés durant un tour de plateau. Le second mode est en mode continu, la table doit tourner soit pendant un nombre de tours définis soit durant un temps défini. Enfin, une fonctionnalité sous forme de touches de magnétoscopes permettant de bouger comme l’utilisateur le souhaite grâce à des touches d’avance, de recul faisant tourner la table le temps de l’appui ou selon le nombre d’appui de l’utilisateur. Il nous a également fallu ajouter le focus-stacking au fonctionnement de base.

1. ***Communication Téléphone-Boitier de Commande***

Afin de pouvoir intégrer plus facilement les fonctionnalités, un travail sur les trames d’envoi ont été réalisées ainsi que sur les communications. Plusieurs solutions ont été envisagées afin de faciliter la distinction entre modes et également de rendre le code plus ouvert à l’intégration de nouveaux modes. (Peut-être qu’il faudrait développer un peu en expliquant plus concrètement comment vous avez redu le code plus souple pour l’intégration de futurs nouveaux modes

Au niveau de la communication, nous sommes restés sur une trame unique, cela évitant tout problème de décodage au niveau du boitier de commande le nombre d’informations contenues dans chaque trame pouvant être différent. Le schéma de la trame envoyé est donc identique en mode temps réel (ou continu) et programmé. L’un des éléments de la trame permet de distinguer si l’on est dans tel ou tel mode.

ID\_COMMANDE, MODE, ACCELERATION, VITESSE, NOMBRE\_DE\_PAS\_TABLE, DIRECTION, CHOIX\_ROTATION, NOMBRE\_DE\_TOUR, TEMPS\_DE\_ROTATION, NOMBRE\_DE\_PHOTOS, NOMBRE\_DE\_CAMERAS, PAUSE\_ENTRE\_CAMERAS, FOCUS\_STACKING

Accélération, Vitesse, Nombre\_Photos, Nb\_Pas\_Table, Degré\_Désiré, Durée\_Pause

***Comparaison entre l’ancienne trame et la nouvelle trame***

Lors de la récupération de la trame par l’ESP32, tous les éléments de la trame sont récupérés sous forme de String (chaine de caractère) comme représentée ci-dessus. Les éléments sont ensuite récupérés grâce à un découpage selon la virgule et stockés dans un tableau de valeurs. Les valeurs sont ensuite interprétées selon le mode.

La trame est composée des éléments suivants :

-ID\_COMMANDE : ID qui est généré automatiquement dans l’application afin de pouvoir différencier plusieurs envois de rotation différents

-MODE : Différence entre les modes pour savoir si l’on est en mode Programmé et Temps Réel

-ACCELERATION, VITESSE : correspondent à la vitesse de la table pendant une rotation et de remise en route après un arrêt.

-NOMBRE\_DE\_PAS\_TABLE correspond au nombre de pas nécessaire pour le moteur pour que la table rotative réalise un tour complet.

-CHOIX\_ROTATION : permet de faire une distinction entre le NOMBRE\_DE\_TOUR et le TEMPS\_ROTATION.

-NOMBRE\_DE\_CAMERAS correspond au nombre de caméras utilisées pour la prise de vue avec un maximum de 9 caméras se déclenchant successivement selon la valeur de PAUSE\_ENTRE\_CAMERAS durant un arrêt moteur. Le NOMBRE\_DE\_PHOTOS correspond au nombre de prise de vue (indépendant du nombre d’appareil) au cours d’un tour.

-FOCUS\_STACKING : Le nombre de prises de vues avec niveau de focus différents par appareil photo et par arrêt. (A voir demain)

Lors de la réception des informations par l’application, le boitier de commande crée une liste d’instructions, qui regroupe toutes les instructions qui vont être envoyées aux différents périphériques pour effectuer l’action requise par l’application. La liste d’instruction comporte donc tous les envois de rotation, les prises de photo et permet donc d’avoir une séquence préparée avant toute communication. Chaque instruction ayant un ID spécifique.

1. ***Communication Boitier de commande-Moteur/périphériques externes***

La liste d’instruction ayant été créée, le boitier peut alors communiquer avec les différents périphériques.

1. Cas du moteur

Lorsque le boitier de commande a récupéré et décodé la trame comme décrit dans

le paragraphe précédent, le boitier de commande communique au moteur afin que celui-ci réalise le déplacement souhaité.

La communication se fait par le biais de la communication BLE déjà explicitée. Chaque côté peut lire et écrire dans les caractéristiques du serveur BLE qui est le moteur.

Lorsque le moteur n’effectue aucune action, sa caractéristique est à 0, ce qui permet au boitier de commande de savoir que ce dernier est libre et peut recevoir de nouvelles instructions. En fonction du paramètre MODE reçu par le boitier de commande, deux cas se distinguent dans l’envoi du datagramme :

* *Cas Mode temps Réel (MODE à 1)*

Dans le cas du temps réel, seuls les paramètres : Accélération, Vitesse, Nombre\_de\_pas\_table, Direction, Choix\_Rotation, Nombre\_De\_Tour et Temps\_Rotation, sont utilisées.

L’élément Choix\_rotation permet de comprendre si l’utilisateur souhaite réaliser une rotation en continu selon le nombre de tours souhaité (dans ce cas Temps\_Rotation ne servira pas) ou selon une durée précise. Il est inutile de préciser le nombre d’appareils photos puisque ce mode est destiné à une prise de vue par caméra par exemple.

Lorsque ce mode est sélectionné, le datagramme écrit dans les caractéristiques du moteur est la suivante :

CHOIX\_ROTATION = 0

ACCELERATION, VITESSE, DIRECTION, CHOIX\_ROTATION, NOMBRE\_DE\_TOUR

OU

CHOIX\_ROTATION = 1

ACCELERATION, VITESSE, DIRECTION, CHOIX\_ROTATION, TEMPS\_ROTATION

Les valeurs envoyées TEMPS\_ROTATION et NOMBRE\_DE\_TOUR étant quelque peu modifiées pour permettre d’envoyer une valeur utilisable telle qu’elle par le moteur. Ainsi le TEMPS\_ROTATION est multiplié par 1000 pour correspondre à un temps en secondes (auparavant millisecondes) et NOMBRE\_DE\_TOUR est multiplié par le nombre de pas total de la table.

* *Cas Mode Programmé (MODE à 0)*

Dans le cas du mode Programmé, seuls les éléments Accélération, Vitesse, Nombre

De pas table, Direction, Nombre\_De\_Photos, Nombre\_De\_Caméras, Pause\_entre\_Camera sont utilisés.

Lorsque ce mode est sélectionné, la trame envoyée au moteur par bluetooth est la suivante :

ACCELERATION, VITESSE, DIRECTION, 0, NOMBRE\_DE\_PAS

Ce datagramme ne diffère pas beaucoup de celui du temps réel. Il s’agit en réalité d’un cas particulier dans lequel CHOIX\_ROTATION = 0. Ce qui explique la 4ème valeur qui permet de forcer la lecture par le moteur d’un cas de rotation en fonction d’un nombre de pas à effectuer.

Concernant le nouveau paramètre NOMBRE\_DE\_PAS, celui-ci est un calcul entre le nombre de photos et le nombre de pas total pour effectuer un tour complet de la table. Il s’agit donc du nombre de pas entre chaque déclenchement.

Indépendamment du mode, une fois l’action reçue les caractéristiques du moteur sont mise à « 1 » par le moteur pour que le boitier de commande sache qu’il n’est pas disponible. Une fois la rotation effectuée, le moteur se met ses caractéristiques à « 2 » pour que le boitier de commande comprenne qu’une instruction a été terminée. Celui-ci se charge alors de remettre les caractéristiques à « 0 », de déclencher les appareils photos et de renvoyer la trame. Il est important de comprendre que dans le cas du mode programmé, il y aura autant d’envoi de datagramme que de photos à prendre, tandis qu’en mode programmé il n’y aura qu’un envoi unique.

Le moteur peut également avoir ses caractéristiques correspondant au chiffre « 3 », ce qui signifie que le boitier de commande lui envoie une pause dans son fonctionnement.

1. Cas des périphériques

FOCUS STACKING à ajouter

1. ***Communication Boitier de commande-Téléphone***

Afin de permettre à l’utilisateur d’avoir une vue sur les actions en cours, un feedback

a été créé entre le boitier de commande et l’application.

Trois types de datagrammes ont été créés. Le premier informe l’application qu’une instruction a été créée pour un périphérique. Cette étape a lieu avant tout envoi d’instruction aux périphériques, et permet à l’utilisateur de connaitre toutes les actions qui vont être réalisées. Il lui renvoie alors un datagramme composé :

* De la chaine de caractère « création »
* D’un ID\_COMMANDE : représente l’identifiant de la commande concernée, ID commande précédemment créée par l’application et envoyée au boitier.
* -D’un ID\_instruction ; représente l’identification de l’instruction créée (prise de photo, demande de rotation moteur) présente dans la liste d’instruction de la commande créée lors de la réception.
* Du nom du périphérique concerné (moteur par exemple)
* Le datagramme qui sera transmis au périphérique

Le second type de datagramme permet au boitier de commande de communiquer le

Démarrage de l’instruction. Cela permet notamment à l’utilisateur de savoir où en est la table rotative dans sa liste d’instruction (par exemple : première prise de photo en cours). Cela permet donc d’avoir une vision en temps réel. Le datagramme envoyé à l’application est le suivant :

* Chaine de caractère « en cours »
* ID\_commande
* ID\_instruction : correspond à l’ID de l’instruction qui vient de démarrer

Enfin le boitier de commande permet d’annoncer la fin d’une instruction, afin

de pouvoir clore afficher à l’utilisateur que l’instruction est terminée, la suite logique étant qu’à la suite un datagramme en cours pour l’instruction suivante sera envoyée. Le datagramme est composé des éléments suivants :

***(A VOIR DEMAIN)***

Schéma récapitulatif des com à faire

Focus Stacking à aborder plus tard

Compléter avec la partie moteur

***3) Réécriture du code***

Afin de rendre le code plus accessible et modifiable dans le futur, nous avons cherché à modifier la structure du code.

Tout d’abord, le code a été structuré selon différentes library qui sont en fait des fichiers regroupant différentes fonctions principales afin de simplifier le fichier principal. Ainsi, notre code fait appel à ces différentes library et donc aux différentes fonctions dans le code principal sans nécessité de coder plusieurs fois la même chose et également pour simplifier la lecture du code à un autre programmeur.

A titre d’exemple, pour communiquer avec un LCD, il faut l’initialiser, afficher sur l’écran, ou encore créer différents symboles à afficher. Une library LCD contient un fichier .h avec toutes les entêtes des noms de fonctions et un .cpp qui contient le code de chacune des fonctions. Donc on aurait une fonction initialiser, afficher, créer\_symbole\_bluetooth. Il suffit alors simplement de créer un objet dans le code principal de type LCD et d’utiliser les fonctions grâce à la synthaxe : nomObjet.fonction.

FAIRE UN SCHEMA EXPLICATIF

PARLER DE LA LISTE D’INSTRUCTION MAIS DEJA FAIT DONC PAS TROP

***4) Adaptation du PCB et du boitier de commande***

Un des objectifs principaux du projet était d’obtenir des PCB fonctionnels. Lors du premier semestre, seul l’un sur les deux fonctionnait en situation réelle. Nous avons donc mis un accent sur la conception rapide des composants et des PCB sur Altium, tout en menant nos recherches pour les améliorations à apporter, tels que les ESP32 et les optocoupleurs.

Pour chacune des parties, un schéma hardware a été réalisé, permettant d’identifier facilement la manière de refaire nos connexions, puisqu’il faut le rappeler le but de Monsieur Lissarague est de fournir un projet accessible et modifiable. Ces schémas nous ont également aidé à réaliser les différentes schématiques.

Au niveau du PCB du boitier de commande, celui-ci ayant été réalisé très vite car nous partions d’une base fonctionnelle, nous avons fait le choix de garder au départ les relais et les optocoupleurs dans le cas où la nouvelle solution ne fonctionnerait pas. Le but étant de pouvoir gérer les problèmes de routage au plus vite pour envoyer les PCB en fabrication en Chine.

# III/ Application Android

***1) Présentation Générale de l’application***

Pour réaliser l’application, nous avons choisi de travailler sur Android Studio, à la suite de nos cours Android du Second Semestre.

Une application Android est composée de différentes activités qui représentent les différentes pages : connexion, envoi instructions etc… et qui ont chacune un layout, c’est-à-dire une mise en page spécifique, et une fonctionnalité. Dans la suite de cette partie, nous aborderons également le concept de fragment, il s’agit en réalité d’une sorte de « collage » sur une activité, un fragment est un nouvel élément qui vient se positionner au-dessus de l’activité (avec son layout, ses fonctionnalités) et qui est utilisé par exemple pour des fenêtres pop-ups ou lors de l’ouverture d’un élément d’un menu par exemple pour garder le lien avec l’activité principale qui a permis l’ouverture de fragment.

Notre application se compose de deux grandes activités. La première est l’activité de connexion qui est donc une page qui permet de sélection le périphérique avec lequel on souhaite communiquer. La seconde activité gère tout le dialogue avec la table rotative, elle est donc composée de différents fragments, chaque fragment correspondant aux différents modes ou éléments nécessaires au bon fonctionnement. Il y a donc un fragment de sélection des périphériques avec lequel le boitier de commande devra communiquer, le mode temps réel, le mode programmé.

Dialogue

Connexion

Mode programmé

Mode temps réel

Base de données

Sélection

***Représentation schématique de la structure de l’application***

***2) Communication Bluetooth grâce à l’application***

La première chose réalisée sur l’application a été de communiquer directement avec un ESP32. Le but premier étant de parvenir à se connecter directement avec le boitier de commande pour réaliser la connexion série Bluetooth.

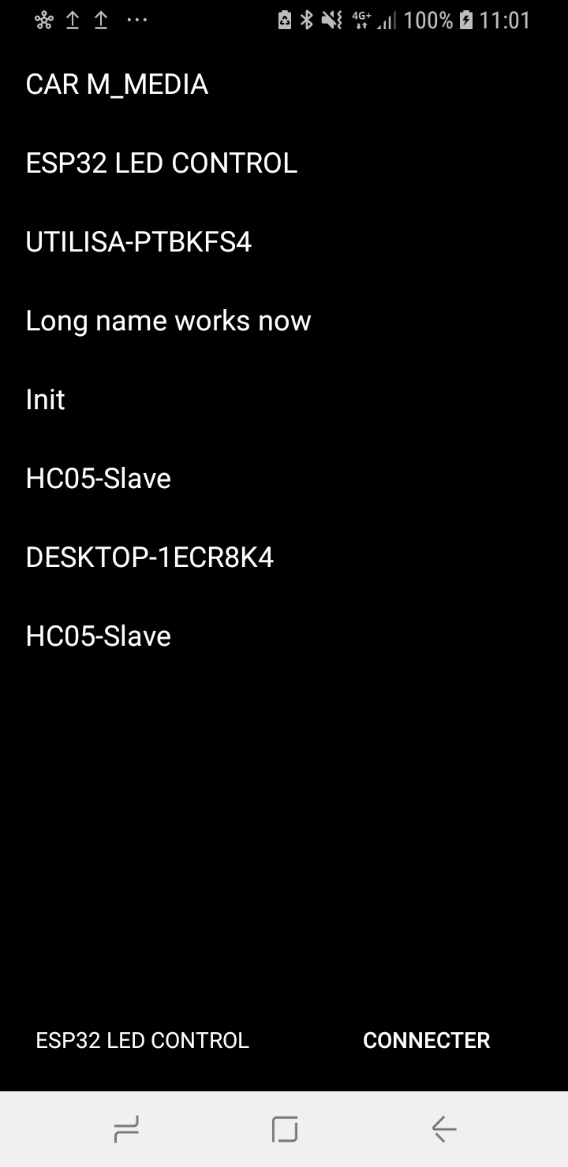
Lors d’une connexion Bluetooth, d’ordinaire, une liste des périphériques appareillable s’affichent pour sélectionner celui qui nous intéresse. Nous avons représenté le même système dans l’application. Une classe objet a été créée, c’est-à-dire une variable personnalisée contenant elle-même différentes variables et fonctions qui la caractérise. Cette classe représente un périphérique avec tous ses caractéristiques propres : son nom, adresse, s’il est connecté, les différents éléments permettant de s’y connecter. Les périphériques utilisent notamment des sockets pour la communication, qui permettent notamment une connexion durable sans nécessité d’envoyer l’adresse de destination à chaque envoi de données.

Lorsque la liste est récupérée, chaque élément est affiché suivant la mise en page suivante :



Les éléments étant tous stockés dans une liste, chaque élément est affiché (une ligne équivaut à un élément) selon la mise en page qui a été définie et qui est ici de récupérer le nom de chaque objet de la liste et de l’afficher.

Lors d’un appui sur un élément, et il est possible de se connecter en appuyant sur un bouton.







PERIPHS SUR LAPPLI

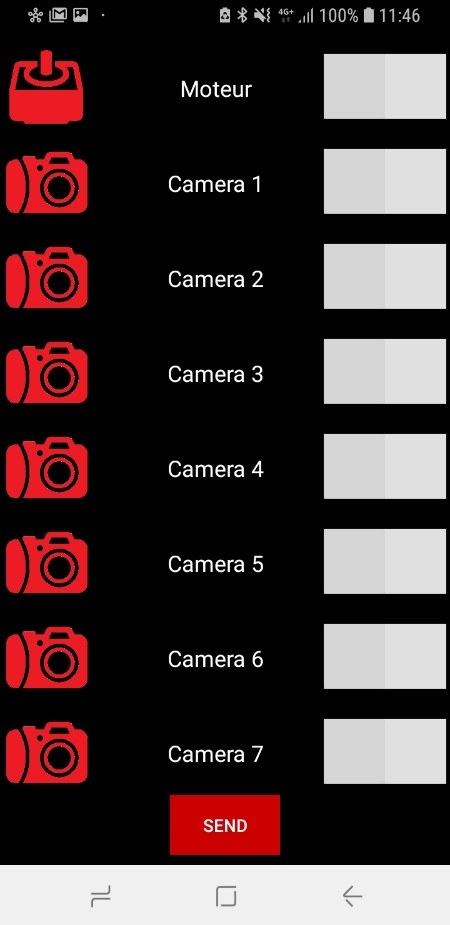
A COMPLETER COMMENT ELLE SAIT QUE PERIPH = ESP ?

***3) Envoi et récupération de données***

Une fois la connexion réalisée, l’application est appairée avec un périphérique et peut donc lui communiquer des informations.

En raison du focus stacking, une première page de sélection de périphérique à été ajoutée, on parle ici des périphériques qui communiqueront avec le boitier de commande.

Cette page se présente sous la forme d’une liste contenant tous les éléments que l’utilisateur souhaite utiliser et commander par le boitier de commande. Il y a donc le moteur, la sélection des différentes caméra utilisée (leur numéro dépendant du branchement sur le PCB) et si le mécanisme du focus stacking a été ajouté et quelle caméra peut donc l’utiliser.



L’utilisateur peut ainsi sélectionner les périphériques grâce à un switch. La sélection des différents éléments permet

***4) Mise en place d’une Base De Données***

Afin de rendre l’application plus ergonomique, une base de données a été mise en place pour permettre à l’utilisateur de sauvegarder certains modes de rotation sans être obligé de ressaisir les données.

Afin de différencier les deux modes de fonctionnement, deux BDD ont été créées, l’une contenant uniquement des données pour le mode programmé et l’autre uniquement des données temps\_réel, ce qui ne pose pas de problème au niveau de l’implémentation car la réalisation est identique.

La réalisation d’une BDD implique la création de fichier “objets” qui déterminent quelles sont les informations qu’un élément de la BDD contient, un objet représentant, par exemple, toutes les informations nécessaires dans le cas d’un mode programmé. Ainsi, lors de l’ajout d’un élément dans la BDD, tous les champs sont récupérés et ajoutés dans l’objet adapté qui contient également des champs correspondants.



Id

Acceleration

Vitesse

Direction

NbPas

NbCamera

NbPhoto

PauseEntreChaqueCamera

Focus Stacking

Objet ValeurProgrammé

de la BDD

***Champs utiles pour un envoi de***

***type Mode Programmé***

***Illustration de l’objet ValeurProgramme et de la récupération des données***

Concernant l’ID de l’élément de la BDD, il était impossible de donner un nom par défaut directement dans le programme car les objets avec le même id s’écrasent. Nous avons donc mis en place un système de pop-up qui permet à l’utilisateur de choisir le nom qu’il souhaite. Sachant qu’il n’est pas possible d’ajouter plus de 10 éléments dans chacune des bases de données.



Enregistrement dans la Base De Données concernée



***StoryBoard de l’enregistrement d’un élément de la BDD***

La difficulté majeure d’une BDD réside dans l’appel des différentes fonctions. En effet, il est impossible de charger une BDD dans un code principal. Il faut passer par une tâche asynchrone qui permet de réaliser, par exemple, le chargement en tâche de fond. L’appel se fait donc dans le fragment dans lequel nous sommes situés, la tâche est ensuite réalisée en fond, et à la fin de la tâche, une fonction appelée postExecute est appelée, ce qui nous permet de faire le lien avec le fragment principal et garder une certaine logique. Cela permet de ne pas commencer à travailler sur les éléments d’une liste qui n’est pas encore créée dans la tâche de fond, par exemple.

Schéma simplifié

L’utilité de la BDD étant de pouvoir sélectionner un élément déjà entré et sauvegardé, nous avons mis en place un système de chargement et d’affichage de tous les éléments. Les éléments sont alors chargés, un nouveau fragment est ouvert contenant un RecyclerView qui permet en réalité d’afficher tous les éléments de la liste en suivant une mise en page identique pour chaque élément. Une fois chaque élément affiché, il peut être sélectionné ou supprimé, le RecyclerView nous permettant de savoir exactement avec quel élément on interagit.

Schéma

La suppression permet de supprimer l’élément directement dans la BDD, le fragment contenant la liste des éléments est alors fermé, pour éviter à l’utilisateur de pouvoir sélectionner un élément qui n’existe plus, ce qui permet à l’utilisateur de recharger la nouvelle BDD et ses changements.

La plus grande difficulté rencontrée lors de l’ajout de cette fonctionnalité concerne la sélection et le remplissage des champs dans le menu d’envoi en fonction de l’élément sélectionné. Finalement, nous avons considéré que lors d’un appui sur le bouton Sélection, les éléments de l’objet sélectionné sont sauvegardés, le fragment du menu pour l’envoi est ouvert, et ces éléments sont ajoutés en argument au début du code. La vérification de l’existence d’argument est donc vérifiée à chaque ouverture d’un menu d’envoi (le code ne faisant pas la différence entre une ouverture via le menu principal ou post-sélection).

StoryBoard

# IV/ Objectifs réalisés

Au terme de ce projet, nous avons réalisé tout ce qui nous avait été demandé par Monsieur Lissarague.

L’application permet d’envoyer les données au boitier de commande qui est propre au système. Nous ne passons plus par une application générique du store, ce qui permet donc à l’utilisateur d’identifier plus facilement quelles sont les informations à envoyer, sans se soucier de la trame d’envoi qui est réalisée en interne de l’application.

Concernant le système hardware, nous avons réglé les problèmes de synchronisation qui existaient et dérangeaient le bon fonctionnement par intermittence grâce à l’implémentation d’un module ESP32 plutôt que d’un Arduino. Les recherches sur les optocoupleurs nous ont également permis de pallier le problème du seul relais fonctionnant sur les trois originaux. Nous avons donc les 9 appareils qui se déclenchent lorsque le boitier de commande en donne l’ordre.

Les PCB, bien que des petits problèmes subsistent sur la version finale, sont désormais tous les deux fonctionnels et adaptés aux nouveaux composants. Le développement de la nouvelle version ayant été efficace, il a été possible de les commander en Chine pour obtenir une version plus propre et plus adaptée pour une utilisation régulière. Les problèmes observés lors du test final ont été corrigés pour une future commande.

Les objectifs ayant été remplis assez rapidement, ou en bonne voie, certaines améliorations ont rapidement été commencée. La première d’entre elle concerne l’accessibilité au code. Afin de permettre à d’autres étudiants de reprendre le code, tout à été pensé (exemple)afin de pouvoir ajouter des nouvelles fonctionnalités facilement sans devoir remanier l’intégralité du code.

Nous nous sommes également penchés sur l’optimisation de l’application, celle-ci envoie les données, comme souhaité, au boitier de commande, mais permet également une meilleure interaction avec l’utilisateur. L’intégration d’un feedback permet à l’utilisateur de suivre exactement la prise de photo ou les rotations du moteur, et la BDD permet une facilité d’utilisation dans le cadre de prises de vue identiques d’un objet à l’autre.