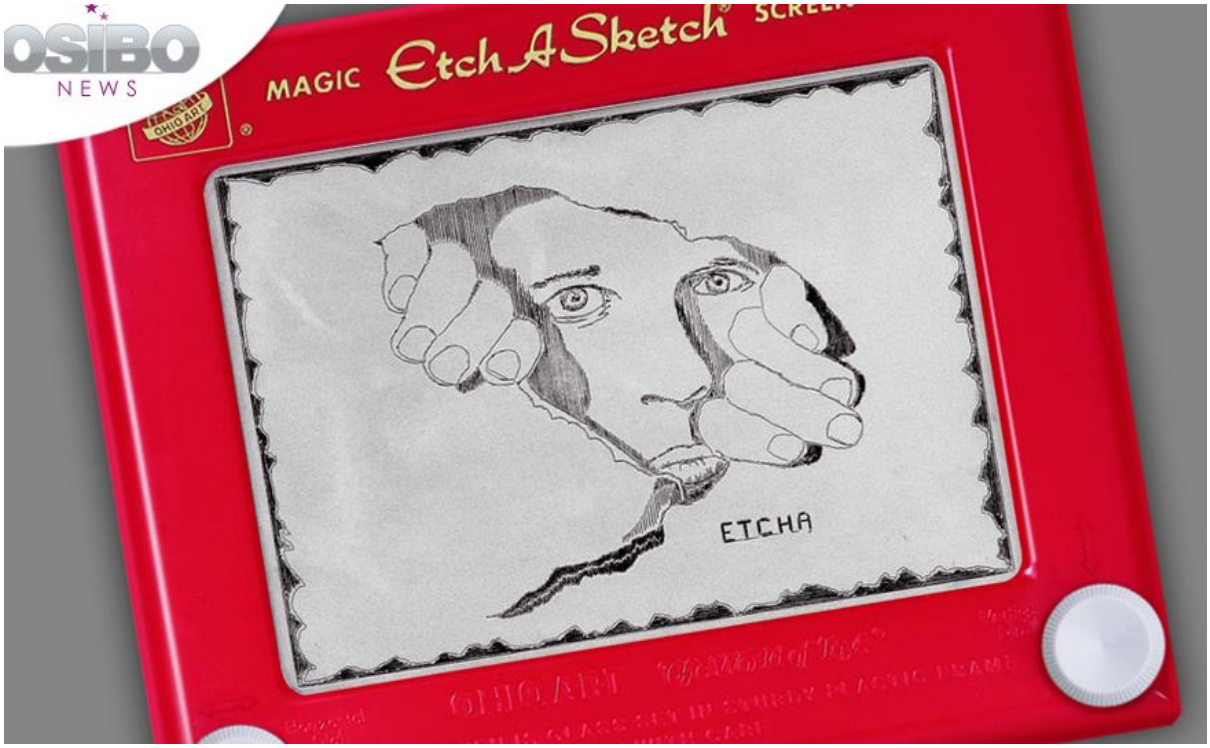
Université de **T**ours

École polytechnique de l'université de Tours

Informatique et Systèmes Intelligents Embarqué



Mini Projet C++

MagicScreen storm V10+

Mathis PALMA / Benjamin CHEROUGE

ISIE4A

Encadrant : Alexis ROLLAND

Table des matières

1. Contexte 2

2. Présentation du projet 2

3. Choix matériel 3

3.1. Capteur de distance 3

3.2. Écran I2C 3

3.3. Carte de programmation 4

3.4. Écran ordinateur 4

4. Environnement de travail 5

4.1. Développement embarqué 5

4.2. Affichage graphique 5

5. Fonctionnement attendu 6

6. Analyse amont 7

6.1. Analyse SART 7

6.1.1. DCD 7

6.1.2. DFDO 8

6.1.3. Dictionnaire des données 11

6.1.4. PSPEC/CSPEC 13

6.2. Analyse UML 14

6.2.1. Diagramme des cas d'utilisation 14

6.2.2. Diagramme des classes 15

7. Réalisation 16

7.1. Partie MBED studio (lien vers dépôt Github) 16

7.2. Partie Processing 16

7.3. Modélisation 3D 18

8. Vidéos démonstration rendu final 20

9. Bilan et difficultés rencontrées 20

Conclusion et pistes d'amélioration 21

1. Contexte

Dans le cadre des cours de C++ d'ISIE 4A nous avons pour objectif de réaliser un « mini-projet » dans le but de mettre en application les notions d'analyse amont et de codage en langage orientée objet. Pour ce projet, le sujet devait être définit par l'équipe et devait répondre à 2 contraintes. La première étant d'être compatible avec le matériel fournit par l'école et la seconde étant que le projet devait être le plus « fun » possible.

2. Présentation du projet

Notre équipe composée de Benjamin CHEROUGE et Mathis PALMA ont choisi de travailler sur le projet MagicScreen V10+. Ce projet au nom farfelu n'est autre qu'une version améliorée et plus fun à l'utilisation des fameuses ardoises magiques plus connus sous le nom de « Télécran ». Voici-ci-dessous une image d'un télécran pour vous rappeler votre jeunesse :

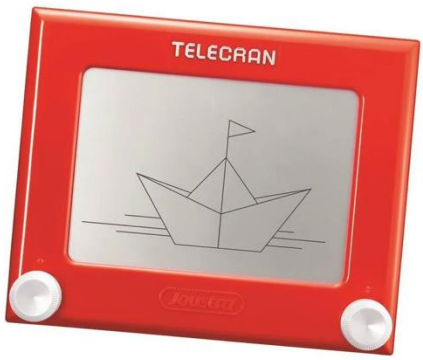


Figure 1 Télécran originel

Pour rappel le principe du télécran est de réaliser des dessins en utilisant uniquement 2 molettes situées aux extrémités de l'objet.

Notre projet a pour objectif de remplacer ces deux molettes par 2 capteurs et faire le rendu sur un écran. Ainsi l'utilisateur n'aura qu'à déplacer ses mains à plat de haut en bas au-dessus des capteurs pour réaliser ses plus grands chefs d'œuvre à en faire rougir Van GOGH.

3. Choix matériel

3.1. Capteur de distance

Pour remplacer les deux molettes nous avons choisi 2 capteurs de distances infrarouges de références Sharp - GP2Y0A21YK0F. Ce sont des capteurs analogiques basés sur une technologie infrarouge. Ces derniers sont alimentés entre 4,5V et 5,5V en entrée et génère une tension de sortie comprise en 0 et 3VCC en sortie. Voici ci-dessous une image de ce capteur :

Figure 2 Sharp - GP2Y0A21YK0F

3.2. Écran I2C

Pour réaliser l'affichage graphique en raison des choix restreint dans la gamme d'afficheur compatible pour notre projet, au début du projet nous avons choisi de partir sur un afficheur OLED 0,96'' 128 x 64 Grove I2C alimenté en 3V. Cet afficheur se base sur un driver SSD1308. En effectuant des recherches nous avons pu trouver et commencé à travailler avec une libraire existante : **SSD1308_I2C_128X64** crée par **Wim Huiskamp** est disponible sur Mbed studio. Voici ci-dessous l'écran. Voici-ci-dessous une image de l'écran utilisé :

Figure 3 Ecran OLED 128x64 I2C sous base driver SSD1308

3.3. Carte de programmation

Pour s'interfacer avec tous les équipements nous avons décidé de partir sur une carte nucleo L073RZ. Voici-ci-dessous le PINOUT de la carte :

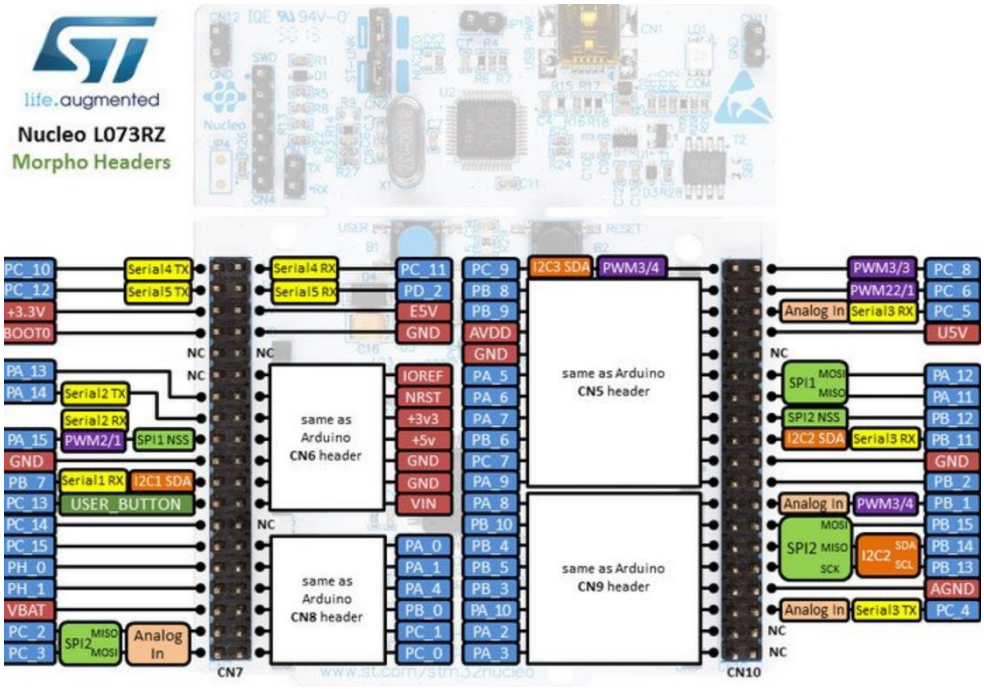


Figure 4 PINOUT nucleo L073RZ

Sachant que pour notre projet nous n'avons besoin que d'une seule connexion I2C, 2 entrées analogique cette carte répond largement à nos besoins. A noter que nous utiliserons le bouton poussoir présent sur la carte pour réaliser des commandes utilisateurs (initialisation / clear screen) détaillée par la suite.

3.4. Écran ordinateur

En raison de la taille de l'écran OLED et après des essais réalisés, nous avons préféré réaliser l'affichage directement sur un écran d'ordinateur en passant par l'interface logiciel « Processing 4.3 » et une connexion USB. Veuillez trouver une présentation d'avantage détaillé de Processing 4.3 par la suite.

4. Environnement de travail

4.1. Développement embarqué

Pour réaliser la programmation en C++ de notre carte nucleo L073RZ, nous avons décidé de travailler avec l'environnement MBED, sur la version local « MBED Studio ».

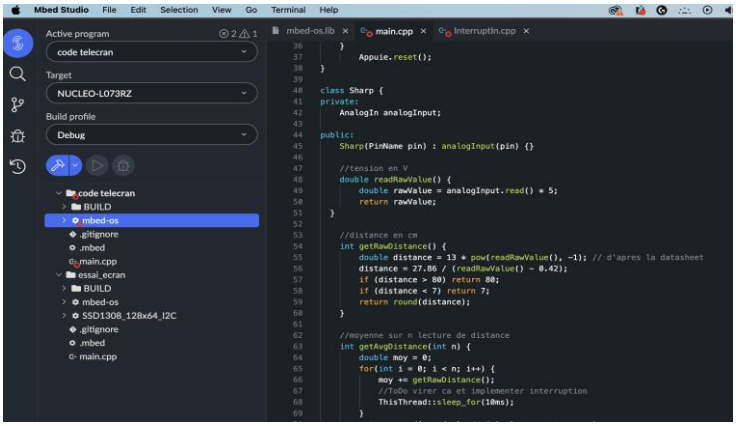


Figure 5 Illustration mbed studio

4.2. Affichage graphique

Pour réaliser l'affichage graphique sur l'écran d'ordinateur nous avons choisi d'interfacer notre carte en USB avec l'ordinateur en utilisant le logiciel Processing.

Processing est un environnement de développement open source conçu pour simplifier la création d'affichage graphique. Il utilise un langage de programmation simplifié basé sur Java qui permet diverses fonctionnalités (animations, ajout d'image de fond, etc.) Son fonctionnement repose sur deux fonctions essentielles : setup() pour l'initialisation et draw() pour le rendu continu.

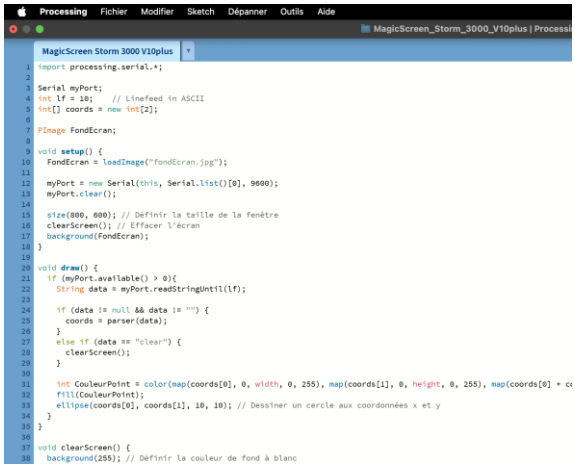


Figure 6 Illustration processing

5. Fonctionnement attendu

Avant de réaliser la phase d'étude nous avons défini de manière détaillée le fonctionnement de notre système.

- Lorsque l'utilisateur démarre le système, ce dernier commence par une phase de calibration durant laquelle l'utilisateur doit positionner ses mains au-dessus du système *(une main au-dessus de chaque capteur*) pour « lancer » le mode dessin.

- La calibration ne dure que quelque seconde et une fois finie l'utilisateur pourra voir s'afficher des points sur l'écran de l'ordinateur *(via l'interface graphique processing V4.3).* Le système est donc dans son état « normal » et les mouvements réalisés par la main situé au-dessus du capteur gauche du système générerons une coordonnée sur l'axe X et pour le capteur droit sur l'axe Y. Plus la distance des mains par rapport aux capteurs est importante plus les coordonnées sur l'axe X et Y sera important et inversement.

- Si l'utilisateur souhaite réinitialiser l'affichage, il n'aura qu'a réaliser un appui sur le bouton poussoir d'une durée supérieur à 2 secondes et le système repartira comme au démarrage.

6. Analyse amont

6.1. Analyse SART

La première étape de notre projet a été de mettre en place une phase d'analyse SART dans le but de décrire notre système dans sa globalité. Il est donc nécessaire de spécifier son modèle fonctionnel, son modèle de contrôle et son dictionnaire des données.

6.1.1. DCD

Dans un premier temps et dans le but de délimiter notre objet d'étude, nous avons réalisé notre diagramme DCD. Voici ci-dessous notre diagramme DCD :

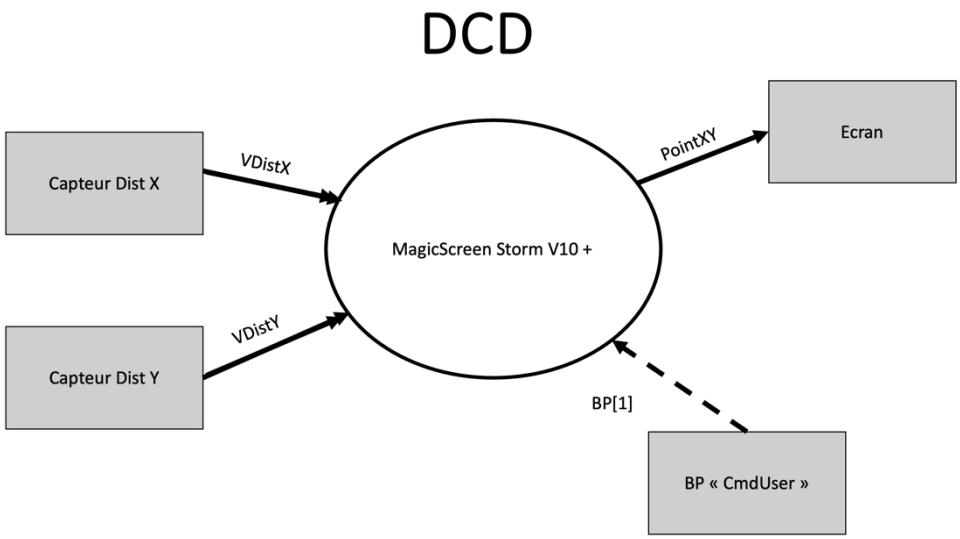


Figure 7 Diagramme DCD

Notre système est donc composé d'un processus maître qui sera le MagicScreen, ce dernier sera interconnecté aux 2 capteurs de distances dont il recevra de chacun un flot de donnée continue. Le système sera également connecté à un bouton poussoir dont il recevra un flot de contrôle. Enfin, notre processus maître fournira un flot de donnée continue vers un écran.

6.1.2. DFDO

Une fois notre Diagramme DCD réalisé, nous avons réalisé notre diagramme DFD0 pour détailler d'avantage le fonctionnement du système en décomposant notre processus maître en sous-processus et nos flots E/S en sous flots. Voici ci-dessous notre diagramme DFD0 :

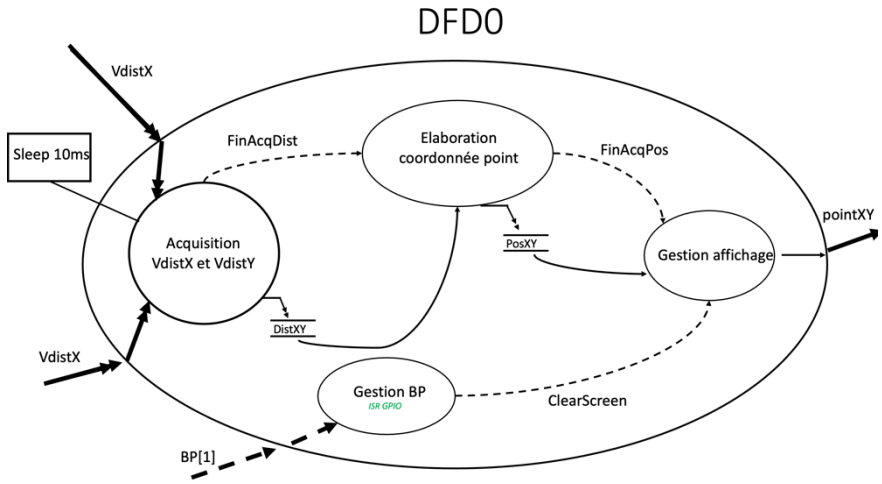


Figure 8 Diagramme DFD0

Comme nous pouvons le voir, notre analyse a permis de découper notre processus maître en 4 sous-processus. Voici ci-dessous le descriptif de ces processus. *Veuillez noter que le détail des signaux sera réalisé dans la partie « dictionnaire des données ».*

Processus « Acquisition de donnée » :

Fonction :

Ce processus a pour but de récupérer les signaux analogiques en provenance de deux capteurs de distance (X et Y) et de traduire ces signaux en une distance XY. Ce processus est réveillé par intervalle de temps régulier (toutes les 10ms). Une fois l'acquisition terminée, ce processus réveil le processus « Élaboration coordonnée point ».

Entrées :

- Valeur analogique image de la distance capteur X **« VdistX »** ***(****flot de donnée continu)*
- Valeur analogique image de la distance capteur Y **« VdistY »***((flot de donnée continu)*
- Signal de réveil à intervalle régulier de 10ms *(appel périodique par timer, flot de contrôle)*

Sortie :

- Valeur de distance XY **« DistXY »** *(puit de donnée)*
- Information de fin d'acquisition **« FinAcqDist »** *(flot de contrôle)*

Processus « Élaboration coordonnée point » :

Fonction :

Ce processus réveillé par l'information de fin d'acquisition de la distance XY permet de convertir la valeur issue du processus « Acquisition de donnée » sur la distance XY et de la convertir en une coordonnée d'un point XY. Une fois ce processus finalisé, il déclenche à son tour le processus « Gestion affichage ».

Entrées :

- **« DistXY »** calculée par le processus « acquisition de donnée » *(puit de donnée)*
- Signal de réveil indiquant la fin d'acquisition de la distance **« FinAcqDist »** *(flot de contrôle)*

Sortie :

- Position XY du point **« PosXY »** *(puit de donnée)*
- Signal de fin d'acquisition de la position **« FinAcqPos »** *(flot de contrôle)*

Processus « Gestion BP » :

Fonction :

Ce processus permet de traduire les commandes utilisateurs lors d'un appuie sur le bouton poussoir *(BP[1] situé sur la carte dans le cas présent)* . En fonction de la durée de l'appuie sur le bouton poussoir, ce processus va générer un signal vers le processus « gestion d'affichage »

- Si l'utilisateur réalise un appuie long (supérieur à 2 secondes), le processus interprète cela comme une requête de réinitialisation de l'affichage graphique et envoi alors un signal « ClearScreen » vers le processus « gestion affichage ».

Entrées :

- Interruption GPIO associé à l'appuie sur le bouton poussoir **« BP[1] »** *(flot de contrôle)*

Sortie :

- Signal reset de l'affichage graphique « **ClearScreen »** *(flot de contrôle)*

Processus « Gestion Affichage » :

Ce processus récupère la coordonnée X et Y générée par le processus « élaboration coordonnée point » et se charge de générer un point sur une interface graphique. Ce processus se lance à chaque fois qu'il reçoit une information de fin d'acquisition de la position du point à afficher sur l'écran par le processus « élaboration coordonnée point ». Ce processus prend en parallèle en compte le flot de contrôle en provenance du processus « Gestion BP ».

- Si le signal « ClearScreen » est actif, l'affichage graphique est réinitialisé et l'intégralité des points générés sur l'interface graphique sont supprimés.

Entrées :

- Signal de demande de reset « **ClearScreen »** *(flot de contrôle)*
- Signal informant que l'acquisition des coordonnées est finalisée **« FinAcqPos »** *(flot de contrôle)*
- Coordonnée XY du point à générer **« PosXY «** *(flot de donnée)*

Sortie :

- Génération d'un point à une coordonnée XY **« PointXY »***(flot de donnée)*

6.1.3. Dictionnaire des données

Afin de préciser et définir correctement chaque flots entrant et sortant des différents processus, nous avons réalisé un dictionnaire de donnée.

VdistX/Y :

- Flot de donnée continu
- Valeur (tension) image de la distance
- Nb de valeur possible : 65536 → 16 bits
- Non signé
    - Typage : *double*

FinAcqDist

- Flot de contrôle
- Signal de fin d'acquisition de conversion
- 2 états possibles : Booléen →1 bit
- Non signé
    - Typage : *BOOL*

DistX/Y

- Flot de donnée continu
- Distance en cm X et Y
- Nb de valeur possible : 65536 → 16 bits
- Non signé
    - Typage : *float*

FinAcqPos

- Flot de donnée continu
- Signal de fin acquisition et conversion de la mesure des capteurs
- 2 états possibles : Booléen →1 bit
- Non signé
    - Typage : *BOOL*

PosX/Y

- Flot de donnée continu
- Vecteur position XY
- Nb de valeur possibles : 1024 → 10 bits
- Non signé
    - Typage : *int*

BP[1]

- Flot de contrôle
- Appuie utilisateur (interruption GPIO)
- 2 états possibles : Booléen →1 bit
- Non signé
    - Typage : *BOOL*

PointXY

- Flot de donnée continu
- Coordonnée X et Y convertis en coordonnées écran
- Nb de valeur possibles : 128 → 7 bits
- Non signé
    - Typage : *int*

ClearScreen

- Flot de contrôle
- Commande de réinitialisation de l'affichage graphique
- 2 états possibles : Booléen → 1 bits
- Non signée
    - Typage : BOOL

6.1.4. PSPEC/CSPEC

Pour terminer notre phase d'analyse SART, nous avons réalisé une description plus détaillée de nos processus présentés précédemment en décrivant les algorithmes élémentaires des processus à travers des PSPEC que voici :

readRawValue()

- Entrée : Aucune
- Processus : Lit la valeur brute de l'entrée analogique, et retourne cette valeur.
- Sortie : La valeur brute lue de l'entrée analogique.

getRawDistance()

- Entrée : Aucune
- Constantes : <distanceMin> et <distanceMax>
- Processus : Calcule la distance en utilisant le coefficient et le décalage. Si la distance calculée est supérieure à <distanceMax>, retourne <distanceMax>. Si elle est inférieure à <distanceMin>, retourne <distanceMin>. Sinon, retourne la distance calculée arrondie et multipliée par 10.
- Sortie : La distance calculée, limitée par <distanceMin> et <distanceMax>, arrondie et multipliée par 10.

getAvgDistance(int n)

- Entrée : Un entier n qui représente le nombre de mesures à prendre pour calculer la moyenne.
- Processus : Prend n mesures de distance en utilisant getRawDistance(), en faisant une pause de 10ms entre chaque mesure. Calcule la moyenne de ces mesures, l'arrondit et la multiplie par 10.
- Sortie : La moyenne des n mesures de distance, arrondie et multipliée par 10.

sendCoordinates(int x, int y)

- Entrée : Deux entiers x et y.
- Processus : Convertit les entiers en chaînes de caractères, les concatène avec une virgule entre elles et une nouvelle ligne à la fin pour former un message.
- Sortie : Envoie le message via la communication série.

clearScreen():

- Entrée : Aucune.
- Processus : Crée une chaîne de caractères "clear\n".
- Sortie : Envoie le message "clear\n" via la communication série.

checkClearOk():

- Entrée : Aucune.
- Processus : Lit une chaîne de caractères de l'entrée standard et la compare à "clearOK".
- Sortie : Renvoie true si la chaîne lue est égale à "clearOK", sinon renvoie false.

6.2. Analyse UML

6.2.1. Diagramme des cas d'utilisation

La première partie de notre étude UML a été de concevoir le diagramme des cas d'utilisation que voici :

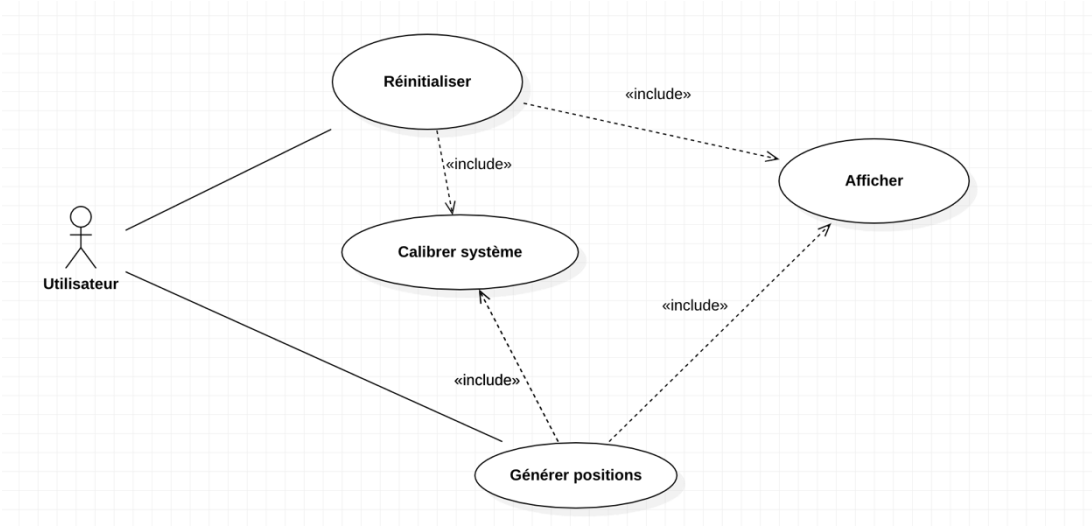


Figure 9 Diagramme UseCase

Selon le diagramme ci-dessus, l'utilisateur peut interagir avec le système de 2 manières. Il peut réinitialiser le système, ce qui entraine impact sur l'affichage (nettoyage de l'écran). Lorsque le système est réinitialisé cela entraine son passage en calibration. En réalisant des mouvements au-dessus des capteurs l'utilisateur peut générer les positions de différents points, pour ceci il faut qu'au préalable le système ait été calibrer. Une fois les points générer, cela déclenche donc un affichage sur l'écran.

6.2.2. Diagramme des classes

Pour continuer nous avons pu dresser le digramme des classes de notre système que nous avons affiné au fur et à mesure de la phase de réalisation du projet.

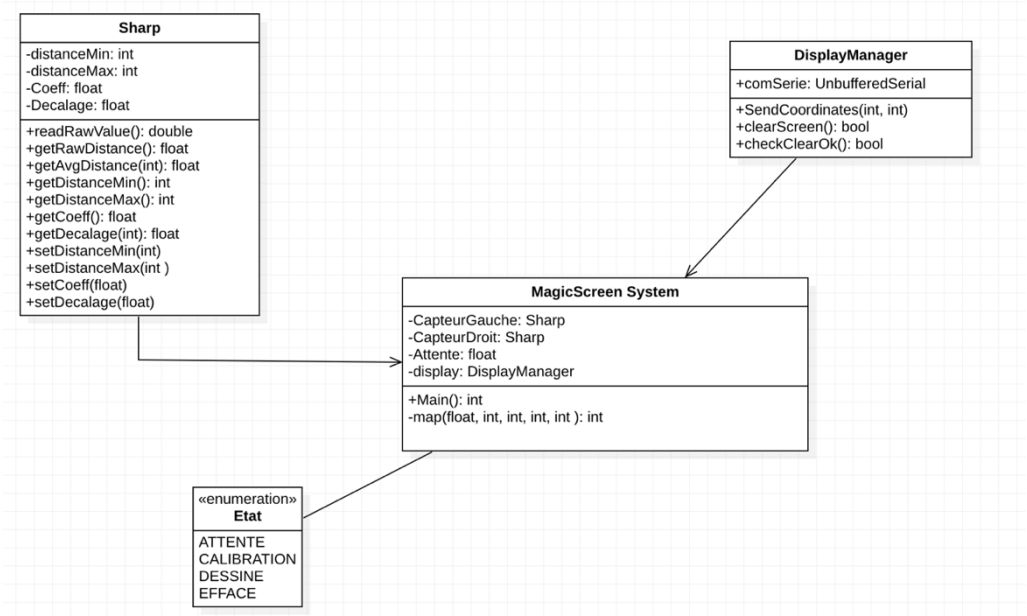


Figure 10 Diagramme des classes

Nous avons donc une classe dédiée à l'acquisition des données issues du capteur infrarouge Sharp. Celle-ci intègre deux constructeurs, un spécialement dédié à notre référence de capteur et un permettant d'instancier la classe avec des paramètres différents. Bien évidemment des accesseurs sont disponibles afin de modifier les paramètres. Trois méthodes sont disponibles pour l'utilisateur : une afin de récupérer la valeur brut issue du CAN, une retournant la mesure de distance à l'instant T et une retournant une valeur lissée sur n lectures

Il y a également une classe permettant de gérer la communication avec l'ordinateur et plus précisément le logiciel Processing. Le constructeur attend en paramètre un objet de la classe UnbufferedSerial de mbed-os, de cette manière notre classe n'a pas à s'occuper d'établir une liaison UART. Trois méthodes sont disponibles : la première permet d'envoyer la coordonnée au logiciel d'affichage, la seconde permet d'envoyer l'ordre d'effacement de l'écran et enfin la dernière s'assure que l'écran à bien été effacé.

MagicScreen System n'est pas réellement une classe étant donné qu'il s'agit du programme main tournant sur notre microcontrôleur. Nous l'avons représenté ici afin de mieux visualiser la façon dont s'imbrique les différentes parties de notre programme.

7. Réalisation

Dans cette partie nous détaillerons l'aspect réalisation du projet à travers les 3 grands axes, la partie programmation de la carte nucleo, la partie réalisation de l'interface graphique sur le logiciel Processing et enfin la réalisation de la modélisation 3D du support pour le système.

7.1. Partie MBED studio (lien vers dépôt Github)

Nous avons utilisé le logiciel mbed-studio afin de facilité la programmation de notre cible, car celle-ci étant compatible mbed, ceci nous a permis de ne pas coder en baremetal. Le debugger inclus dans l'IDE à été d'une grande aide dans la résolution de problèmes engendré par le développement de notre mini-jeu. Tout le code source a été intégré dans un repo github depuis l'IDE et est dorénavant disponible au lien ci-dessous :

https://github.com/Inserer-Pseudo/Telecran-magic-3000-turbo.git

7.2. Partie Processing

Pour la partie réalisation de l'interface graphique qu'en parallèle de la gestion de l'affichage sur le logiciel Processing, nous avons travaillé sur l'écran OLED 0,96'' 128 x 64 Grove I2C. Nous avons pu trouver une librairie que nous avons commencé à adapter pour notre application. Cependant nous avons rencontré quelques difficultés avec son utilisation. La connectique de l'écran étant assez complexe à interfacer avec notre carte (connectique grove) et surtout pour des raisons de confort pour l'utilisateur, nous avons préféré choisir de réaliser l'affichage graphique sur Processing V4.3, un logiciel sur lequel nous avions déjà des bases et qui est relativement simple à interfacer avec notre carte.

Sur Processing nous avons décidé de simuler un affichage graphique sur une fenêtre 800x600 pixels. Nous avons choisi de mettre un fond d'écran « Télécran » sur lequel s'effectue l'affichage des points.

Figure 11 Fenêtre rendu MagicScreen via processing

La communication mise en place entre Processing et notre carte nucleo L073RZ et bidirectionnel et s'effectue via une liaison série (UART) avec un bit/rate fixé à 9600 bits/seconde.

Notre interface processing lit donc en permanence sur la liaison série les données émises par la carte. Lorsque des coordonnées sont émises, le système parse les données pour récupérer la valeur X et Y des points à générer. Une fois cela fait les points sont ensuite généré sur l'écran au coordonnées traitées précédemment.

Nous avons ajouté un système de variation des couleurs des points affichées en fonction de leurs positions sur les axes X et Y pour un effet visuel plus sympathique pour l'utilisateur. Voici ci-dessous un exemple du rendu que nous avons sur l'interface.

Figure 12 Fenêtre rendu en cours MagicScreen via processing

Si notre interface lit une donnée contenant l'information « clear », nous supprimons l'intégralité des points généré et renvoi une information « clearOk » vers la carte nucleo pour notifier que l'action a été correctement réalisée.

7.3. Modélisation 3D

Bien que cela n'était pas demandé, nous avons décidé de réaliser un support pour intégrer notre système de façon plus esthétique pour l'utilisateur. Pour cela, nous avons une modélisation 3D de notre support sur le logiciel Fusion 360. Voici ci-dessous les différentes vues 3D du support :

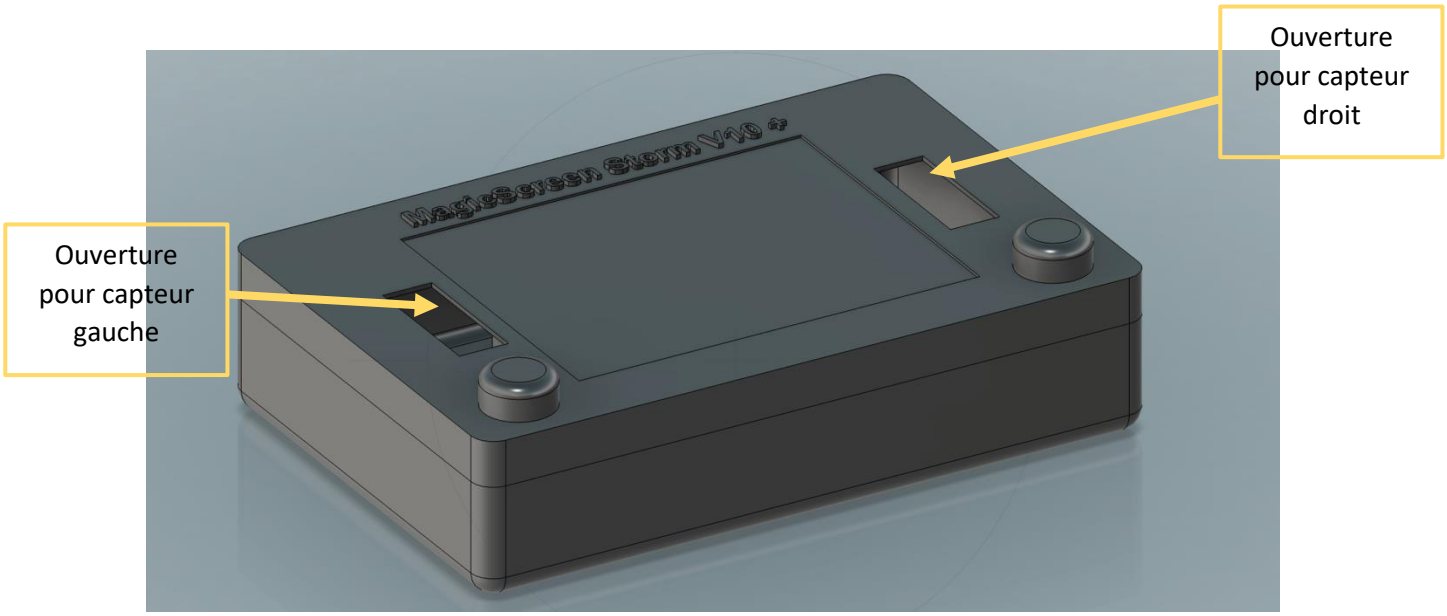


Figure 13 Vue 3/4 arrière support MagicScreen V10 +

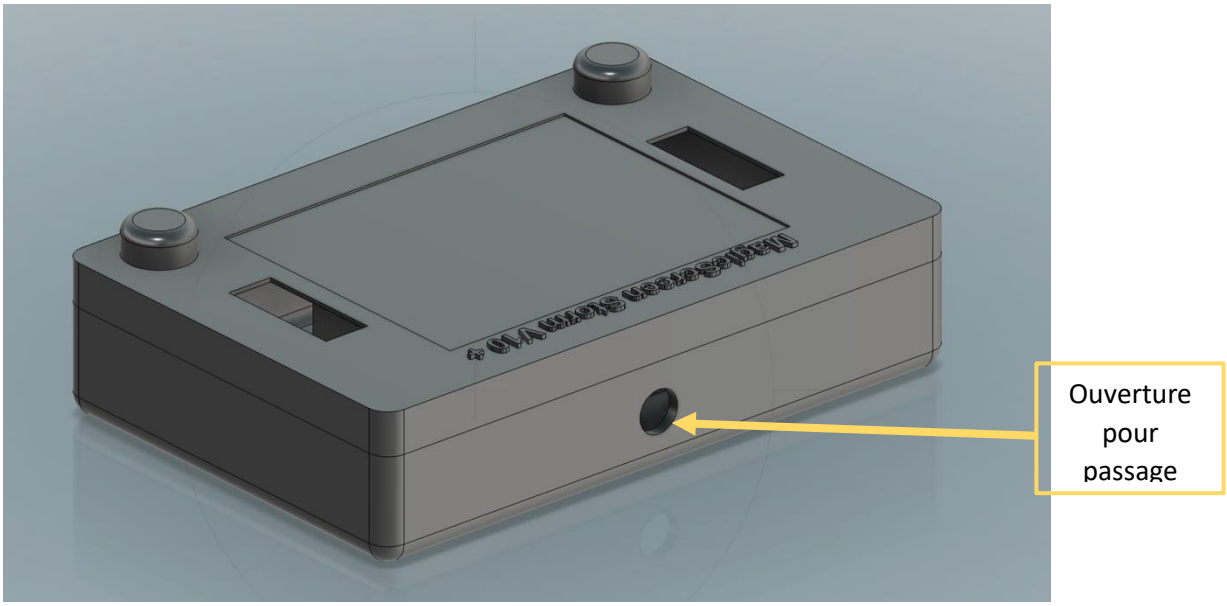


Figure 14 Vue 3/4 avant support MagicScreen V10+

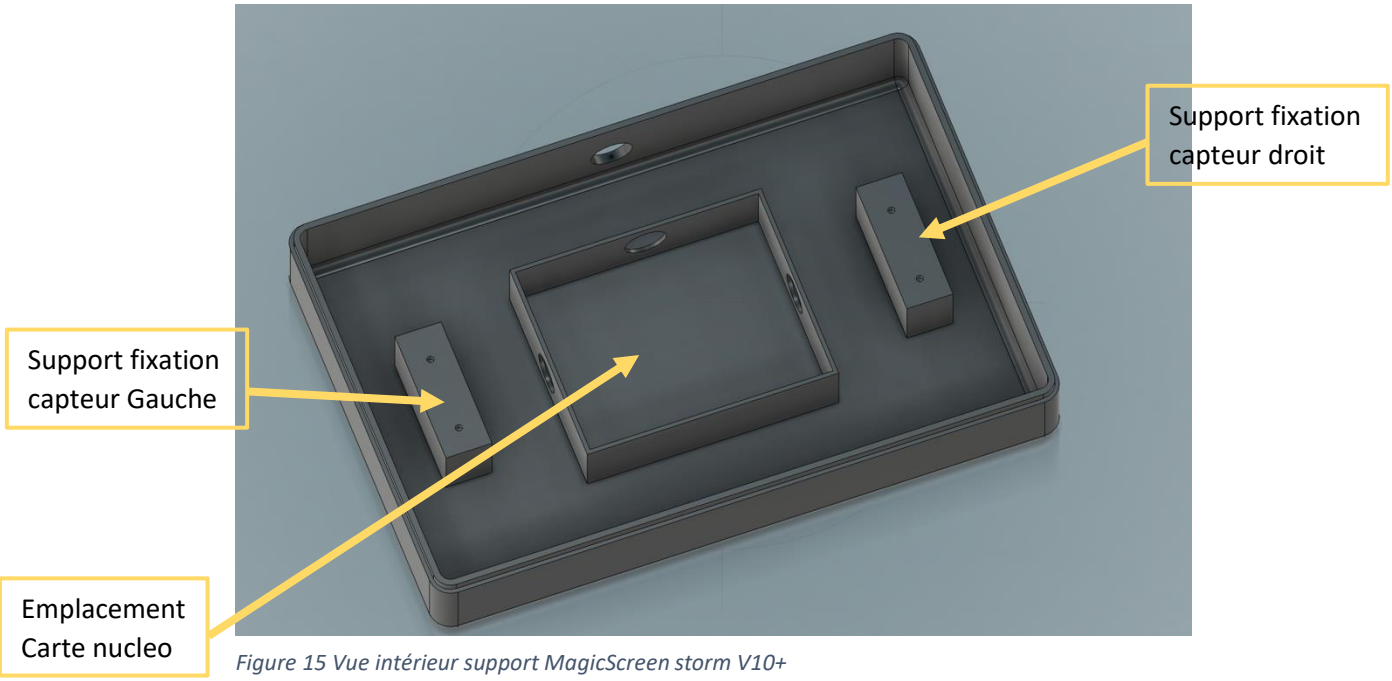


Figure 15 Vue intérieur support MagicScreen storm V10+

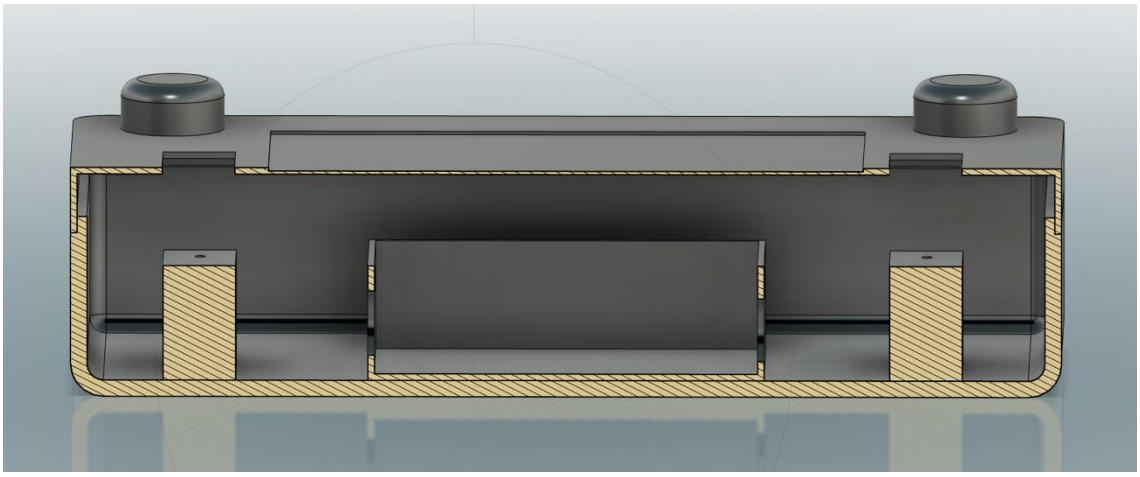


Figure 16 Vue de coupe support MagicScreen storm V10+

Cependant nous avons rencontré des problèmes lors de l'impression 3D du support, en effet nous avons eu un défaut sur le filament utilisé qui s'est avéré défectueux (trop humide → cassant donc mauvaise adhérence au plateau), or nous n'avions plus en stock personnel de quoi remplacer ce filament pour la date butoir, ce support pourra cependant être imprimé par la suite (post date butoir de rendu)

8. Vidéos démonstration rendu final

Veuillez trouver ci-dessous les vidéos démonstrative du résultat final de notre projet :

Démonstration de l'affichage des points :

https://youtu.be/_wcV_4fPN6I?feature=shared

Démonstration de l'effacement de l'écran via bouton :

https://youtu.be/hMct5oMfOTo?feature=shared

9. Bilan et difficultés rencontrées

Pour faire un bilan sur la réalisation du projet. Nous avons réussi de manière globale à répondre au cahier des charges que nous nous étions fixés. Notre système est opérationnel, l'utilisateur est capable de réaliser des dessins en bougeant ses mains. Ce projet nous à permit de mettre en applications nos cours de C++ et génie logiciel. Nous avons appris à programmer de façon concret un système embarqué en langage orienté objet chose que nous n'avons jamais fait par le passé. Nous avons pu voir comment mettre en place des interruptions, la gestion de Timer, utiliser les objets de la librairie STD, travailler sur le thread (même si pas mis en place sur le rendu finale). Nous avons également pu mettre en place les bonnes pratiques de réalisation de projet à travers Github.

Malgré tout nous avons rencontré quelques difficultés notamment sur la communication dans le sens Processing vers Nucleo. En effet nous avons rencontré des difficultés du au formatage du message qui nous à empêcher de contrôler correctement que l'écran avait été correctement réinitialisé. Nous avons également eux des problèmes avec la connectique Grove utilisé sur les capteurs et pour s'assurer d'une connectique fiable et robuste nous avons été dans l'obligation de souder les connectiques des capteurs. Nous avons enfin rencontré des problèmes matériels qui nous ont empêché de réaliser le support 3D que nous avions modéliser.

Conclusion et pistes d'amélioration

Pour conclure ce mini-projet c++ aura été l'occasion de mettre en application toutes les notions d'analyse et de programmation en langage orienté objet (C++). Malgré les difficultés rencontrées nous avons tout de même réussi à répondre au cahier des charges que nous nous étions fixés au début. Nous beaucoup apprécié travailler sur ce format de projet plutôt ludique ou nous sommes maîtres du projet de A à Z.

Il serait possible d'améliorer notre rendu final en intégrant un écran plus grand directement dans le système, sans utiliser un écran d'ordinateur avec l'interface Processing utilisée actuellement. N'ayant pas de bouton déporté disponible à disposition nous avons utilisé le bouton de la carte mais il serait tout à fait envisageable d'utiliser un bouton déporté sur le support, ce qui rendrait plus simple la réinitialisation du système. Il serait également amusant d'intégrer un accéléromètre au système et ainsi l'utilisateur devrait secouer le système afin d'effacer l'écran, comme un vrai télécran.