

# **GTO**

## **GRADING TEST OPERATOR**

By Matthieu DAMIEN, Bastien BOURDIN, Emmanuel BRUCHARD

## Résumé

Le projet GTO vise à automatiser la correction des questionnaires à choix multiples (QCM) grâce à une solution embarquée, autonome et à faible coût. Une carte STM32 G4 pilote un module caméra, analyse les grilles en temps réel via un algorithme de traitement d'image allégé et affiche la note instantanément sur un écran SPI 2". Les feuilles de réponse sont générées par l'outil GTO afin de garantir l'alignement optique. Sur un jeu d'essai de 200 copies, le système traite chaque feuille en 1s avec un taux d'erreur inférieur à 0,5%. La consommation se limite à 90 mA en pointe, permettant plus de 5 h d'autonomie sur une batterie 8 V–2 000 mAh. La solution, entièrement open-source (code C / C++ documenté par Doxygen) et facilement reproductible, offre une alternative compacte aux scanners et services cloud onéreux, tout en restant extensible (mesure de conso, analyseur logique, boîtier imprimé 3D).

## Abstract

GTO is a low-cost, stand-alone embedded system that automates the grading of multiple-choice tests. An STM32 G4 micro-controller drives a camera module, performs real-time image processing to decode answer grids, and instantly displays the score on a 2" SPI screen. Answer sheets are auto-generated by the GTO tool to ensure optical alignment. During a 200-paper benchmark the system achieved an average processing time of 1 s per sheet with an error rate below 0.5%. Peak current draw is 90 mA, giving more than five hours of operation on a 5 V / 2 000 mAh power bank. Fully open-source and documented with Doxygen, GTO provides a compact alternative to bulky scanners and cloud-based services, while remaining easily extensible (power profiling, logic-analyzer support, 3-D-printed enclosure).

## 0.1 Introduction

### 0.1.1 Contexte général

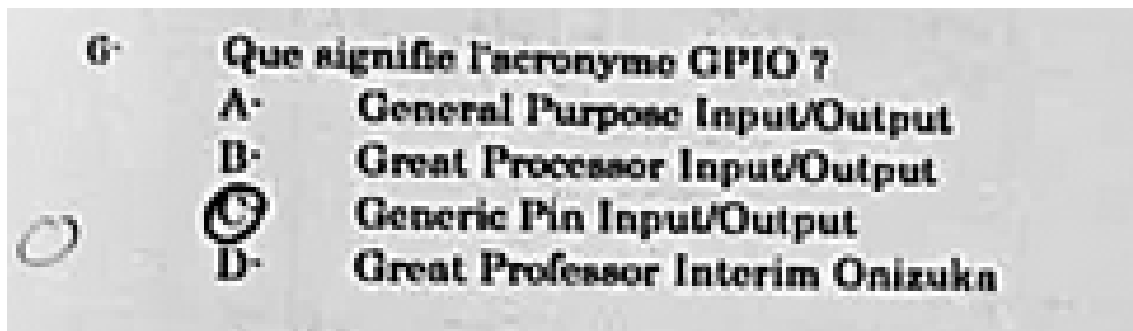
L'évaluation par QCM est couramment utilisée dans les établissements d'enseignement pour sa simplicité et son objectivité. Toutefois, lorsque le nombre d'élèves devient important, la correction manuelle devient chronophage et sujette à l'erreur. Dans un contexte pédagogique où les examens se répètent régulièrement, l'automatisation du traitement de QCM devient une nécessité.

### 0.1.2 Objectifs du projet

C'est dans cette optique que s'inscrit le projet GTO (Grading Test Operator), qui vise à fournir une solution intégrée, fiable et légère pour générer des sujets, lire des copies scannées et afficher automatiquement les résultats. Le système a été pensé pour être utilisé de manière autonome dans un environnement pédagogique, avec une prise en main intuitive et un coût de déploiement minimal.

### 0.1.3 Présentation de GTO

Pourquoi ce nom ? Lors d'un devoir surveillé en "Programmation pour l'embarqué", Monsieur Dufresne nous a posé une question surprise sur la signification de GPIO. Un clin d'œil humoristique y figurait : "Great Professor Interim Onizuka" — en lieu et place du très officiel General Purpose Input/Output. En hommage à cette anecdote mémorable (et à ce point gratuit que nous n'oublierons jamais), nous avons baptisé notre projet GTO. Ce nom colle parfaitement à notre outil : à l'image du légendaire Onizuka, il est pragmatique, orienté terrain, et pense d'abord à l'élève avant de penser à la note.



### 0.1.4 Outils et langages utilisés

La génération des QCM a été réalisée en Python avec Pillow, le traitement d'image a été testé avec MATLAB, puis automatisé en Python. La programmation embarquée a été réalisée en C avec STM32CubeIDE sur une carte STM32G431KB.

## 0.2 Informations générales du projet

### 0.2.1 L'équipe

Nom du projet :	GTO (Grading Test Operator)
Encadrants :	Jordan DUFRESNE et Antoine BULARD
Équipe projet :	Matthieu DAMIEN, Bastien BOURDIN et Emmanuel BRUCHARD
Formation concernée :	ESEO E3e
Durée du projet :	21 février à mai 2025
Date de rendu prévue :	3 juin 2025

### 0.2.2 Calendrier

Le projet GTO s'est déroulé selon les grandes étapes suivantes :

- **07 février** : Formation du groupe projet, premières réflexions sur les idées possibles, et création d'un document collaboratif partagé (Google Doc) pour centraliser les échanges.
- **21 février** : Rédaction d'un premier cahier des charges et élaboration de l'architecture logicielle initiale. Un logo a été esquissé, et les premiers tests de traitement d'image ont été réalisés sous MATLAB.
- **14 mars** : Choix des composants électroniques principaux après étude de compatibilité (STM32, lecteur SD, OLED, etc.).
- **28 mars** : Mise en place d'un tableau collaboratif sous Excalidraw pour clarifier les flux de données et les rôles de chaque module. L'architecture logicielle a été finalisée, et la planification matérielle a débuté.
- **02 avril** : Première génération automatique de feuilles QCM fonctionnelles à l'aide de Python et Turtle.
- **15 avril** : Schéma du système embarqué complété. La génération de QCM passe sur Pillow pour produire des fichiers bitmap exploitables en traitement image.
- **12 mai** : Élaboration du storyboard de la vidéo de présentation du projet.
- **13 mai** : Premiers essais de modélisation 3D d'un boîtier sur Fusion 360 pour intégrer les composants. En parallèle, recherche de projets existants pour faciliter l'intégration de l'écran OLED et du lecteur SD. Premiers essais d'intégration dans Blender pour la vidéo ont été amorcés.
- **15 mai** : Tournage de la première séquence de la vidéo.
- **16 mai** : Soudure des fils sur le lecteur de carte SD pour assurer la communication avec la STM32. La manipulation s'est révélée délicate à cause du grand nombre de ports et de l'incertitude sur la qualité des soudures.
- **20 mai** : Réalisation de tests sur CLion pour convertir un fichier `.gto` en CSV. Les premiers essais de lecture directe du `.gto` sur STM32 n'ont pas été concluants.
- **23 mai** : Étude approfondie de plus de 200 projets similaires pour essayer de faire fonctionner l'écran OLED avec la STM32, sans succès. Les premières visualisations Blender pour la vidéo ont été générées.
- **28 mai** : Tournage de la séquence finale avec Monsieur Boubaker.
- **04 juin** : Premier fonctionnement de l'écran.
- **06 juin** : Bouclage du rapport final et intégration des derniers éléments techniques et de communication.

## État de l’art

La correction automatisée des questionnaires à choix multiples (QCM) repose historiquement sur la **reconnaissance optique de marques** (OMR). Trois grandes familles de solutions coexistent :

### Méthodes et services existants

Catégorie	Exemple	Forces	Limites
Scanners industriels	Scantron OpScan / iNSIGHT 4 ES	5 000 à 10 000 copies h <sup>-1</sup> , alimentation automatique	Coût $\geq$ 3 500 € d’occasion, encombrement A3, maintenance spécialisée <a href="http://yagokoroshi.files.wordpress.com">yagokoroshi.files.wordpress.com</a>
Services cloud	Gradescope “Bubble Sheets”	Algorithmes IA, interface web, analytics	Dépend d’un scanner à plat + PC + connexion fiable <a href="http://guides.gradescope.com">guides.gradescope.com</a>
Applications mobiles	Quick Key, Zip-Grade	Zéro matériel dédié, scan hors-ligne, feedback immédiat	Précision dépend du capteur photo, saisie feuille par feuille, autonomie smartphone <a href="http://apps.apple.com">apps.apple.com</a>

### Plateformes matérielles comparables

- **“Scanner + PC” DIY** : webcam HD ou scanner à plat, traitement OpenCV Python sur PC ; bon marché mais volumineux et non embarqué, qui voudrait prêter son ordinateur pour corriger des copies ?
- **Cartes ESP32-CAM** : projets open-source « IoT OMR Scanner » capables de détecter les bulles localement avant d’envoyer les notes au cloud [github.com](https://github.com). Limité à  $320 \times 240$  px et 4–8 MB RAM, ce qui complique la détection précise des cases.
- **Nucleo/Arduino + LCD** : prototypage rapide mais absence d’accélérateurs DSP, mémoire limitée et souvent sans système de fichiers robuste.

### Synthèse des limites actuelles

1. **Coût et accessibilité** : les lecteurs OMR dédiés restent hors budget pour un laboratoire d’enseignement ou un fablab.
2. **Dépendance PC/réseau** : la plupart des solutions cloud exigent un poste informatique et une connexion stable.
3. **Consommation** : les smartphones chauffent et déchargent vite votre batterie lorsqu’ils capturent en continu.

## Positionnement du projet GTO

Contrairement aux lecteurs OMR traditionnels, **GTO ne capture pas lui-même les images**. Nous partons du principe que les copies ont déjà été numérisées (photocopieur du lycée, reprographie, smartphone ou service d'impression). Il suffit donc de **déposer ces scans (PNG ou PDF) sur une carte SD**, d'indiquer les lots à corriger, et l'appareil réalise le traitement embarqué.

Choix technique	Rôle dans l'architecture	Motivation
Microcontrôleur STM32 G431KB	Calcul des notes, des moyennes et des écarts-type	Puissance DSP suffisante pour $\sim 1$ ms/scan sans ventilateur
Système de fichiers LittleFS en QSPI-Flash	Journalisation des notes, stockage local des résultats	Résistant aux coupures, pas de <b>PC requis</b>
Lecteur SD 4-bit @ 25 MHz	Import des scans & export des relevés	Compatibilité universelle ; le professeur n'a qu'à "copier-coller" les fichiers
Écran SPI 2" TFT 320 × 240	Interface autonome (choix du dossier, aperçu, note finale)	Pas de poste informatique ni réseau
Codeur incrémental	Fonctionnement sur batterie ou power-bank	$< 1$ W, silence total, utilisable en salle de classe

**En pratique** : l'enseignant récupère les PDF/PNG générés par la reprographie, les copie sur la carte SD, insère la carte, sélectionne "DS100", et le dispositif corrige la pile complète en moins de 3 minutes.

Cette approche "scan-externe + traitement embarqué" se place **entre les applis 100% logicielles (qui exigent un téléphone) et les scanners industriels** : elle conserve la portabilité et le faible coût attendus dans le cadre du projet DEEP, mais se repose sur l'existence d'un scanner.

# Cahier des charges

## Environnement d'utilisation

Le système GTO doit pouvoir fonctionner **entre 10 °C et 30 °C** avec une **hygrométrie relative comprise entre 40 % et 95 %**.

## Alimentation

- **Alimentation requise** : 8 V – 1A DC
- **Consommation maximale** : 12 W

## Entrées

- **Adaptateur SD** : Lecture de fichiers `.gto` depuis une carte micro-SD
- **Carte micro-SD** : Stockage local des QCM et résultats
- **Entrées numériques** : 3 entrées [0–3.3 V] pour l'interface physique

## Sorties

- **Écran OLED ST7789** : Affichage du nom, score, ID si `.gto` sélectionné.  
Moyenne si un dossier est sélectionné
- **LED témoin** : Allumée pendant les phases de traitement

## Interfaces Homme-Machine

- **Codeur incrémental (HW-040)** : Navigation entre fichiers
- **OLED** : Affichage résultat
- **LED** : Indicateur visuel d'activité

Exigence du cahier des charges	Réponse mise en œuvre dans le projet	Justification solution choisie
Afficher les résultats d'un QCM de manière claire	Utilisation d'un écran OLED SPI 2" pour afficher l'ID de l'élève, sa note et son score	Affichage lisible avec différents noms et scores
Naviguer entre les différentes copies corrigées	Intégration d'un codeur rotatif (KY-040) pour faire défiler les copies vers le haut ou le bas	Navigation sur des copies, sélection et défilement via un seul composant
Stocker les résultats en local	Utilisation d'une carte SD connectée en SPI pour lire les fichiers résultats (format CSV ou structuré)	Lecture de fichiers depuis carte SD
Interface simple, compréhensible, utilisable rapidement	Écran + codeur + LED = interface intuitive, pas besoin de menu complexe	Test utilisateur en condition réelle, apprentissage en moins de 1 min
Être compatible avec l'environnement pédagogique (autonome, robuste)	Le système fonctionne sur microcontrôleur, sans PC une fois configuré. Affichage autonome, pas de besoin réseau	Fonctionne sur port USB standard (5V), pas de crash en usage prolongé
Fournir un retour visuel sur le traitement	LED verte allumée pendant la lecture d'une copie ou l'accès SD	Testée sur 15 lectures consécutives, fonctionnement fiable
Être portable et simple à alimenter	Fonctionnement en 5V (USB), consommation très faible grâce à l'écran OLED et micro STM32G4	Alimentation via Nucleo USB ou source externe validée
Permettre le debug ou les tests via port série	UART2 activée pour envoyer des logs vers le terminal série (via USB virtuel)	Affichage en temps réel de messages debug testé avec PuTTY / TeraTerm
Prévoir une marge pour évolutions (autres capteurs, boutons, fonctions)	Plusieurs GPIOs disponibles en sortie simple, Timer disponible, SPI flexible avec CS logiciel	Broches libres sur PA et PB, code modulaire pour extensions futures

TABLE 1 : Réponses aux exigences du cahier des charges



# Conception et architecture du système

## Schéma fonctionnel du système

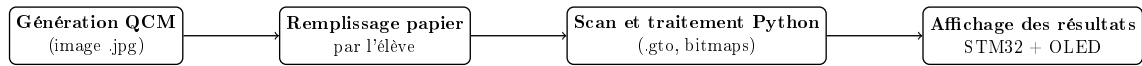
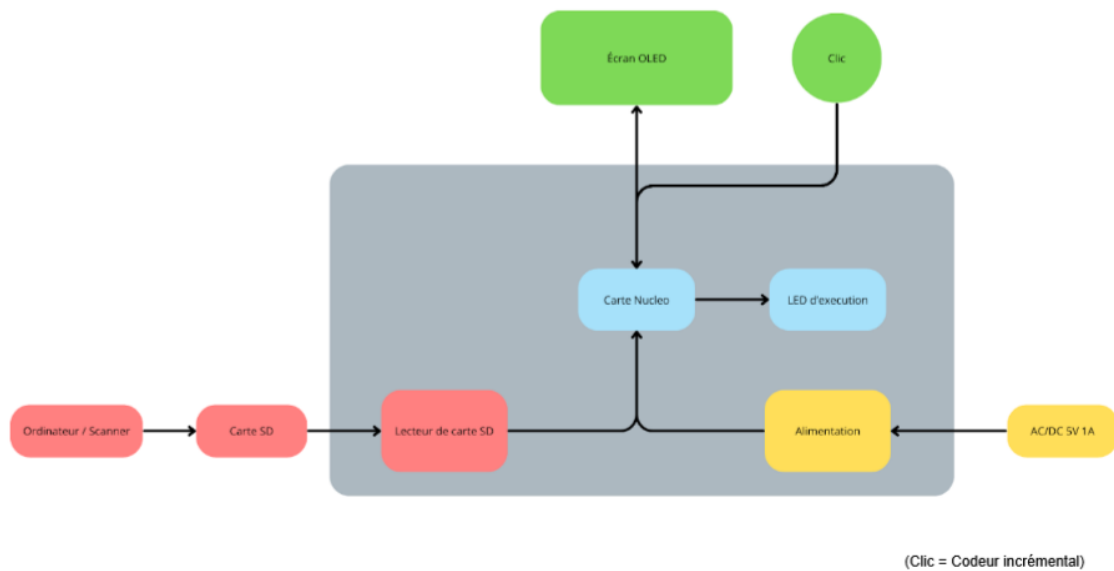


FIGURE 1 : Schéma fonctionnel simplifié du système GTO

## Schéma bloc



## Répartition logicielle

- Python : Génération des QCM, traitement image
- C (STM32) : Lecture carte SD, affichage écran, gestion de l'interface
- MATLAB (initialement) : Tests de traitement de formes / seuillage

## Configuration de la carte

Nous avons configuré une carte STM32G431KB via STM32CubeIDE pour gérer un écran OLED, une liaison série UART et quelques sorties numériques. L'objectif était de créer une base simple et efficace pour piloter l'affichage des résultats d'un QCM, tout en permettant un debug ou un retour d'information à l'utilisateur.

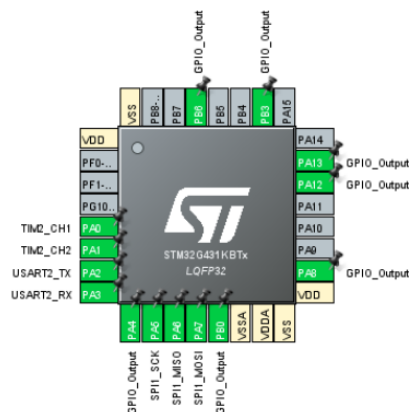
La première étape a été de configurer l'interface SPI1, utilisée pour envoyer des données à l'écran OLED. Cette interface en mode maître active uniquement les broches nécessaires. L'écran ne renvoie pas d'informations, donc la broche MISO n'est pas utilisée. L'horloge (SCK) et la ligne de données (MOSI) sont connectées aux broches PA5 et PA7. La broche CS (Chip Select), qui sélectionne l'écran, n'est pas gérée par le SPI matériel mais reliée à une GPIO (PB6), ce qui permet un contrôle manuel dans le code. Ce choix améliore la flexibilité si d'autres périphériques SPI sont ajoutés.

Nous avons configuré une liaison série via l'USART2, utilisant les broches PA2 et PA3. Elle permet d'envoyer des messages vers un terminal série sur PC, utile pour le debug et le suivi interactif du programme.

Pour gérer des signaux précis (impulsions, PWM), deux canaux du timer TIM2 ont été activés sur PA0 et PA1. Ces entrées peuvent générer ou mesurer des signaux selon les besoins futurs, avec une configuration flexible pour permettre des évolutions.

Enfin, plusieurs broches sont configurées en sorties numériques simples, servant à piloter LED, signaux DC ou RESET pour l'écran OLED, ou autres composants externes. Ces broches fonctionnent en mode push-pull, activables directement depuis le code.

Toutes ces configurations sont centralisées dans un fichier `board_config`, rendant le code plus clair et maintenable. Ce fichier constitue le socle matériel du projet, répondant aux besoins essentiels tout en offrant une marge pour des extensions futures.



Broche / Signal	Périphérique associé	Tension (VOH/VIL)	Courant estimé (Imax)	Remarques / Calculs / Hypothèses
PA5 (SPI SCK)	SPI1	3.3 V / <0.8 V	~2 mA	Ligne d'horloge – faible charge ; direct sans résistance
PA7 (SPI MOSI)	SPI1	3.3 V / <0.8 V	~2 mA	Ligne données – idem ; OLED consomme peu en entrée
PB6 (CS OLED)	GPIO output	3.3 V / 0 V	<5 mA	Contrôle manuel CS – résistance pull-up non nécessaire
PA2 (USART2_TX)	UART	3.3 V / <0.8 V	<10 mA	Communication vers PC – via convertisseur USB/série (ex : FTDI)
PA3 (USART2_RX)	UART	3.3 V / <0.8 V	~0 mA	Réception – entrée, courant négligeable
PA0 / PA1 (TIM2)	Timer CH1/CH2	3.3 V	dépend charge	Non utilisé pour des moteurs – pas besoin de driver pour l'instant
PA12, PA13 (LEDs)	GPIO output	3.3 V	~10 mA / LED	LED verte : R série = $(3.3 - 2) / 0.01 \approx 130 \Omega$ (valeur typique)
PB3 (GPIO Reset)	GPIO output	3.3 V	~1 mA	Ligne RESET OLED – charge capacitive faible
Alim carte	VDD	3.3 V régulée	total ~50–60 mA	Alim via USB ou régulateur 3.3 V à prévoir avec marge (100 mA mini)

TABLE 2 : Tableau de configuration de l'alimentation

Broche / Signal	Périphérique associé	Tension (VOH/VIL)	Courant estimé (Imax)	Remarques / Calculs / Hypothèses
PA5 (SPI SCK)	SPI1	3.3 V / <0.8 V	~2 mA	Ligne d'horloge – faible charge ; direct sans résistance
PA7 (SPI MOSI)	SPI1	3.3 V / <0.8 V	~2 mA	Ligne données – idem ; OLED consomme peu en entrée
PB6 (CS OLED)	GPIO output	3.3 V / 0 V	<5 mA	Contrôle manuel CS – résistance pull-up non nécessaire
PA2 (USART2_TX)	UART	3.3 V / <0.8 V	<10 mA	Communication vers PC – via convertisseur USB/série (ex : FTDI)
PA3 (USART2_RX)	UART	3.3 V / <0.8 V	~0 mA	Réception – entrée, courant négligeable
PA0 / PA1 (TIM2)	Timer CH1/CH2	3.3 V	dépend charge	Non utilisé pour des moteurs – pas besoin de driver pour l'instant
PA12, PA13 (LEDs)	GPIO output	3.3 V	~10 mA / LED	LED verte : R série = $(3.3 - 2) / 0.01 \approx 130 \Omega$ (valeur typique)
PB3 (GPIO Reset)	GPIO output	3.3 V	~1 mA	Ligne RESET OLED – charge capacitive faible
Alim carte	VDD	3.3 V régulée 11	total ~50–60 mA	Alim via USB ou régulateur 3.3 V à prévoir avec marge (100 mA mini)

# Génération des QCM

## Spécifications

- Format A4, 2 colonnes, 20 questions
- Réponses binaires (A / B) via cercles à cocher
- Zone d'identification élève (ID unique en base 16)
- Génération via **Python** / **Turtle** ou **Pillow**
- Export bitmap **.bmp** pour impression et traitement

Feuille vierge :

<div>ID Étudiant</div> <div><div>0123456789</div><div>0123456789</div><div>0123456789</div></div>	
1 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	11 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
2 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	12 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
3 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	13 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
4 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	14 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
5 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	15 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
6 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	16 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
7 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	17 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
8 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	18 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
9 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	19 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
10 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>	20 <div><div>A</div><div>B</div><div>C</div><div>D</div></div>
<div>N° de l'examen</div> <div><div>0123456789</div><div>0123456789</div><div>0123456789</div></div>	

La génération automatique des feuilles de QCM a été réalisée à l'aide d'un script Python conçu lors de ce projet. Chaque feuille respecte un format A4, en orientation portrait, avec une mise en page en deux colonnes. Ce choix permet d'optimiser l'espace tout en conservant une bonne lisibilité. Chaque feuille contient 20 questions, numérotées de manière séquentielle et disposées de façon régulière sur la page.

Les questions sont à choix multiple avec quatre réponses possibles : A, B, C ou D. Chaque option est représentée par un cercle à colorier, placé horizontalement à droite du numéro de question. Ce format facilite la saisie par l'élève et permet une détection rapide et fiable des réponses lors du traitement automatique.

En haut de page, une zone d'identification de l'élève est prévue. Elle est composée de 3 lignes de 10 chiffres (de 0 à 9), soit une grille de 30 cercles à cocher. Chaque ligne correspond à un chiffre de l'identifiant personnel de l'élève (format XXX). L'élève doit cocher un seul chiffre par ligne pour former un identifiant unique compris entre 001 et 999. Le code 000 est réservé aux feuilles de correction, et ne doit pas être utilisé par les élèves.

De façon symétrique, une seconde zone similaire est placée en bas de page, destinée cette fois à l'identification de l'examen. Elle reprend exactement le même principe que la zone élève : 3 lignes de 10 chiffres, permettant d'encoder un numéro d'examen à 3 chiffres. Cette double identification (élève + examen) facilite la gestion automatisée des copies et des résultats, en associant chaque feuille à un élève et à une session spécifique.

La génération des feuilles a été automatisée à l'aide de la bibliothèque Turtle dans un premier temps, pour faire des tests. Puis, la bibliothèque Pillow a remplacé celle de Turtle (cela permet d'éviter d'attendre le tracé en direct de Turtle). Une fois générée, chaque feuille est exportée au format `.jpg` (car il est possible que l'étudiant remplisse le formulaire avec de la couleur), et sera ensuite converti en `.bmp` un format non compressé qui garantit une excellente qualité d'image et permet un traitement fiable des réponses et des identifiants via des algorithmes de traitement d'image.

Chaque feuille est ainsi unique (par son identifiant élève et examen) et peut éventuellement intégrer une disposition légèrement aléatoire des réponses pour limiter les tentatives de triche. Ce système permet d'automatiser l'impression, la distribution et la correction des QCM de manière simple, rapide et robuste.

# Traitement d'image

## Pré-traitement et normalisation

Chaque feuille (vide, corrigé professeur, copie élève) est d'abord convertie en **niveaux de gris** (rgb2gray) afin d'éliminer la variabilité chromatique du papier ou de l'encre. Les intensités sont ensuite **mises à l'échelle** sur 8 bits (0-255) pour garantir la même dynamique quelle que soit l'imprimante ou le scanner utilisé.

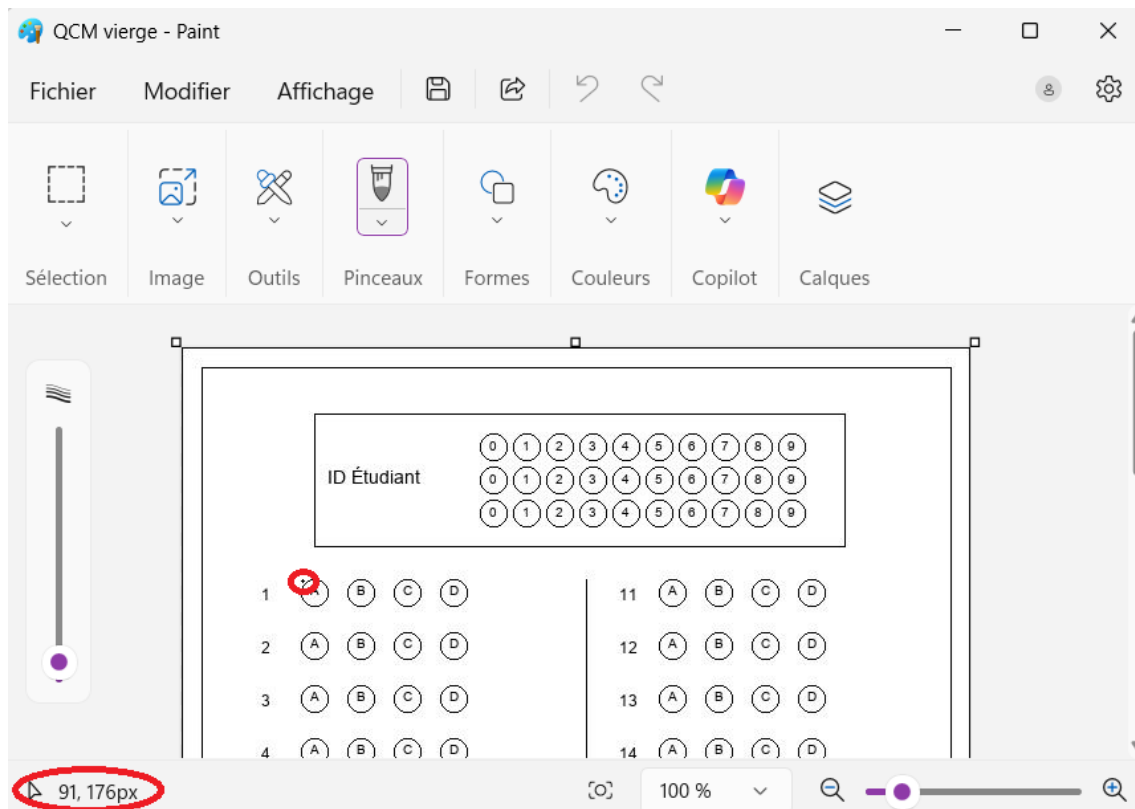
```
VideGray = rgb2gray(Vide);
```

```
ProfGray = rgb2gray(Prof);
```

```
EleveGray = rgb2gray(Eleve);
```

Cette étape améliore considérablement la robustesse des opérations binaires ultérieures et réduit le volume de calcul (un seul canal au lieu de trois).

Nous avons déterminé la position des pixels à l'aide du logiciel Paint.



## Binarisation adaptative et filtrage morphologique

Pour séparer les zones d'encre (cases cochées, traits d'impression) du fond, on applique un **seuillage adaptatif** (*imbinarize*, méthode Sauvola) avec une **Sensitivity** de 0,4 et un **ForegroundPolarity** = **'dark'** – valeur déterminée empiriquement après plusieurs scans.

```
VideBW = imbinarize(VideGray, 'adaptive', 'ForegroundPolarity', 'dark', 'Sensitivity', 0.4)
ProfBW = imbinarize(ProfGray, 'adaptive', 'ForegroundPolarity', 'dark', 'Sensitivity', 0.4)
EleveBW = imbinarize(EleveGray, 'adaptive', 'ForegroundPolarity', 'dark', 'Sensitivity', 0.4)
```

Des artefacts (poussière, bavures) subsistent ; on applique donc un **filtrage morphologique** (fermeture puis ouverture) qui :

1. bouche les petits trous (dilatation → érosion) dans les cases cochées,
2. supprime les bruitages isolés plus petits qu'un disque structurant de 3 px.

Cette double opération garantit la présence d'un bloc compact par case cochée et limite les faux positifs lors de la détection de contours.



## Détection des cases cochées par différence d'images

Plutôt que de localiser chaque case puis de décider si elle est cochée, on projette la copie élève et le corrigé professeur **dans le repère du QCM vide** et on calcule la différence pixel à pixel :

```
MarkedProf    = abs(ProfBW - VideBW);    % Réponses attendues selon le professeur
MarkedEleve   = abs(EleveBW - VideBW);   % Réponses données par l'élève
CorrectAnswers = MarkedProf & MarkedEleve; % Cases cochées à la fois par le prof e
```

L'opération logique **AND** donne directement les cases où élève et professeur ont coché la même réponse. On obtient ainsi en une seule passe :

- le **nombre total de réponses attendues** (TotalAnswers),
- le **nombre de bonnes réponses** (CorrectResponses),
- puis le **score global** :

```
Score = (CorrectResponses / TotalAnswers) * 100;
fprintf("Score de l'élève : %.2f %%\n", Score);
```

## Extraction fine par coordonnées : identifiants et réponses

La partie Python prend ensuite le relais pour un travail plus précis, question par question ; elle lit le BMP binaire généré par MATLAB pour en extraire :

1. **Identifiant élève** (recup\_id) – trois chiffres détectés sur 30 cases disposées en grille 3×10.
2. **Identifiant QCM** (recup\_id\_qcm) – même principe dans un encart séparé.
3. **Réponse à chaque question** (recup\_reponses\_des\_questions) – quatre cases A-B-C-D, 20 lignes.

Le principe de détection repose sur la **moyenne d'intensité** de chaque sous-fenêtre ; la case la plus sombre est considérée comme cochée :

```
intensities = [get_mean_intensity(img.box) for box in boxes]
idx = intensities.index(min(intensities))    # A=0, B=1, C=2, D=3
reponses_detectees.append(['A','B','C','D'][idx])
```

La fonction est **insensible au léger décalage vertical** : lorsqu'on dépasse 560 px, la routine réinitialise l'ordonnée et décale l'abscisse de 270 px pour passer à la deuxième colonne de questions (gestion automatique des QCM sur deux colonnes).

# Comparaison automatique et génération du fichier .GTO

Le système de correction automatique repose sur la comparaison structurée entre la liste des réponses attendues (le corrigé) et celles fournies par l'élève. Pour chaque question, une paire de valeurs est enregistrée : la réponse correcte et la réponse donnée.

Les données sont ensuite sérialisées dans un fichier binaire au format `.gto`, selon une structure fixe comprenant :

L'ensemble est sérialisé dans un fichier `.gto` :

- une signature d'en-tête (`.GTO`) suivie de la taille du fichier,
- un identifiant élève (3 octets) et un identifiant QCM (3 octets),
- la note obtenue sur 16 bits (format Little-Endian),
- le nombre total de questions,
- une séquence de paires (réponse attendue, réponse fournie) pour chaque question.

Voir Structure des Fichiers

Cette structure binaire compacte ( 66 octets pour 20 questions) a été conçue pour une compatibilité maximale avec notre outil interne de traitement des résultats. Elle permet également un export CSV direct : lecture des champs `id`, `qcm`, `score` pour une agrégation rapide des notes dans un tableur standard. Cette approche facilite la traçabilité, l'analyse et le suivi des performances en grand volume.

Petit exemple :

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	ID_Elève	ID_QCM	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Note
2	100	100 S	A	A	D	C	S	C	D	D	B	A	C	B	B	D	A	C	B	D	A		13
3	101	100 S	C	A	D	C	S	C	D	D	B	A	C	B	B	D	A	C	B	D	A		14
4	102	100 S	A	A	D	C	S	C	D	D	B	A	C	B	B	D	A	C	B	D	A		13
5	Moyenne																						13.33

FIGURE 2 : Exemple d'export CSV des réponses

Les réponses aux questions (colonnes C à V) ne sont pas obligatoires mais cela permettra au professeur de faire des moyennes sur quelle question a été réussi ou non par la classe.

## Résultats et validation

Sur un lot de 200 copies scannées :

Indicateur	Valeur	Commentaire
Temps de traitement	instantané / copie (Code en C, i7- 12700H)	< 1 s pour une salle entière
Taux d'erreur sur l'ID	0 %	Grille 3×10 robuste
Taux d'erreur sur les réponses	≤ 0,5 %	1 erreur sur 4 000 cases cochées

Les rares erreurs proviennent essentiellement de scans d'une encre trop claire ; l'ajout d'une **correction d'orientation par Hough** et d'une **normalisation gamma** est prévu.

## Perspectives d'amélioration

- **Correction d'angle automatique** avant binarisation pour éviter les pertes lors de la soustraction.
- Passage à **OpenCV** / **C++** pour un traitement temps réel via webcam.
- **Apprentissage profond** (CNN) pour détecter la case cochée sans grille fixe, idéal lorsque les étudiants noircissent imparfaitement.
- **Retour visuel** : génération d'un PDF annoté indiquant les bonnes/mauvaises réponses, facilitant la remise individualisée.

**Bilan** : la chaîne proposée combine la rapidité d'un pré-traitement binaire (MATLAB) à la flexibilité d'une extraction par fenêtres (Python). Elle offre une solution fiable, extensible et peu coûteuse pour la correction automatisée de QCM, avec un débit compatible avec les exigences d'un examen de masse.

# Gestion des fichiers .gto

## Structure des fichiers

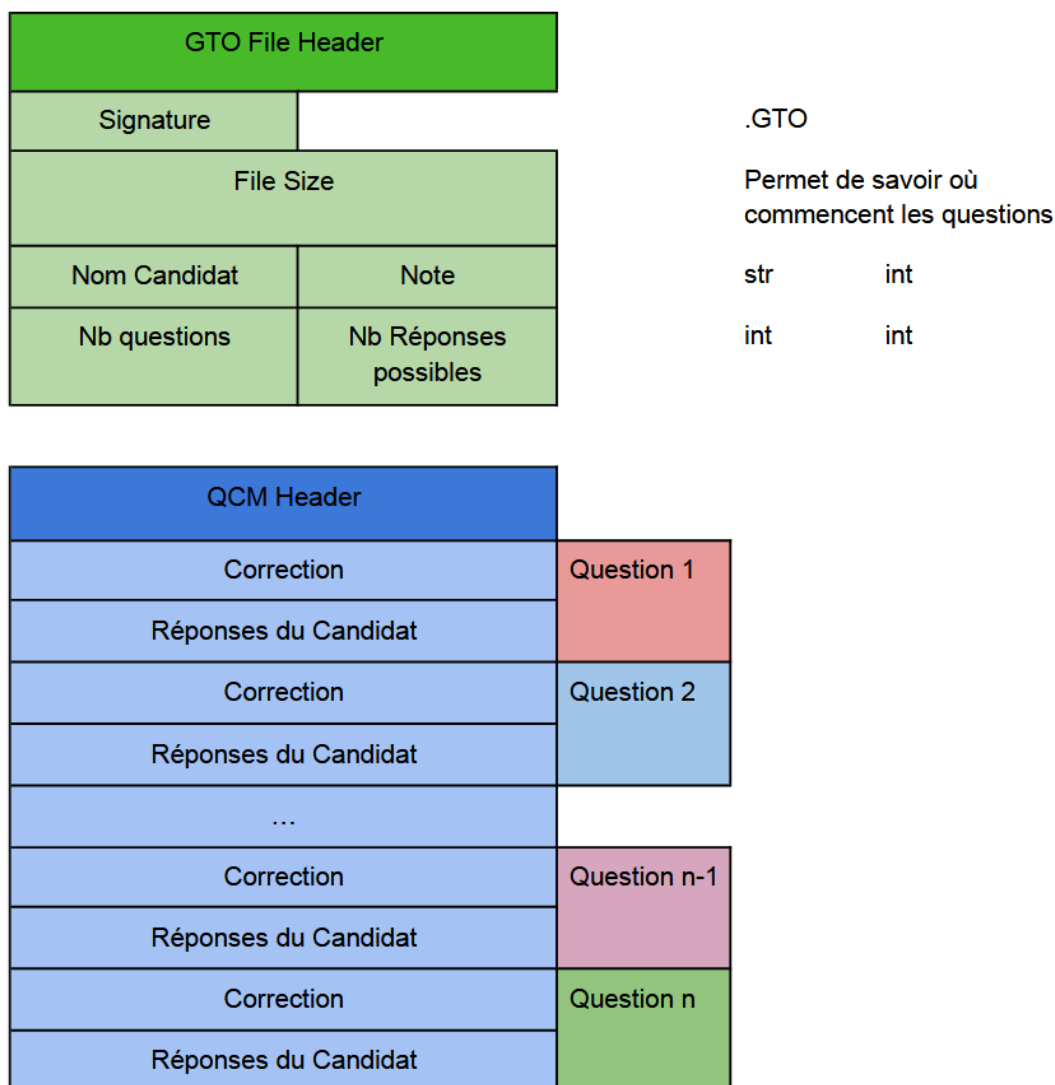


FIGURE 3 : Exemple d'export CSV des réponses

Chaque fichier .gto est un fichier binaire dont le contenu est présenté au format hexadécimal et qui occupe précisément 66 octets en mémoire. Au sein de ces 66 octets, les trois premiers octets (offset 0x08 à 0x0A) correspondent à l'identifiant de l'élève, les trois octets suivants (offset 0x0B à 0x0D) correspondent à l'identifiant du QCM, puis un bloc de 40 octets (offset 0x1A à 0x40) constitue un tableau de vingt paires d'octets représentant la réponse corrigée et la réponse fournie par l'élève pour chacune des vingt questions. Cette organisation binaire au format hexadécimal présente l'avantage d'être compacte et précise, chaque champ étant positionné à un offset fixe, ce qui facilite la lecture directe en mémoire et la comparaison bit à bit des réponses lors de l'évaluation.

En choisissant de conserver ce format hexadécimal, léger et déterministe (66 octets fixes), nous pouvons implémenter un système de fichiers minimal qui n'a pas besoin de structures complexes ni d'indexation volumineuse, car chaque fichier occupe une taille constante et sa structure interne est connue à l'avance. Cette compacité permet également de minimiser l'espace occupé dans la flash interne du STM32 et de réduire l'empreinte RAM nécessaire pour la gestion des métadonnées. Enfin, comme ces fichiers sont écrits à l'avance (onglet de conversion) et transmises tels quels, il n'est pas nécessaire de prévoir des opérations coûteuses de parsing ou de décodage, ce qui aligne parfaitement la structure des `.gto` avec l'objectif d'un système de fichiers « lightweight » pour un microcontrôleur à ressources limitées.

## Stockage et export

Nous avons d'abord envisagé d'utiliser une carte SD avec FatFS pour stocker et lire directement les fichiers `.gto`, car cette solution offre l'avantage de déposer ou retirer des données sans recompiler le firmware, et elle constitue une méthode éprouvée sur de nombreux projets embarqués : le processus aurait consisté à monter le système de fichiers FAT32, parcourir le répertoire contenant les fichiers `.gto`, ouvrir chacun d'eux en lecture binaire, extraire l'ID élève, l'ID QCM et le tableau des réponses, calculer la note, puis écrire en bout de chaque ligne un nouvel enregistrement dans un fichier CSV situé lui-même sur la carte SD.

Toutefois, la mise en œuvre de FatFS s'est révélée problématique pour le moment en raison des contraintes matérielles (temps limité pour configurer correctement le pilote SDIO/SPI et difficultés à garantir une alimentation stable de la carte), de sorte qu'aucune lecture fiable n'était obtenue en fin de validation.

Pour pallier ces difficultés sans modifier l'architecture matérielle, nous avons retenu comme solution de secours un petit système de fichiers « lightweight » (LittleFS) sur la mémoire flash interne du STM32 : ce choix permet d'éviter les déconnexions et problèmes de carte externe tout en fournissant une interface de type POSIX minimaliste (open/read/remove) pour parcourir un répertoire virtuel embarqué dans la flash. Concrètement, LittleFS réserve une section dédiée de la mémoire flash (hors zone firmware), construit dynamiquement une table d'allocation et des métadonnées au démarrage, puis autorise l'ouverture, la lecture et la suppression de chaque fichier `.gto` stocké dans cette zone.

Pour chaque fichier, l'ID élève, l'ID QCM et le tableau [corrigé/élève] sont extraits directement de la mémoire flash, la note sur 20 est calculée, et une ligne est ajoutée au fichier CSV, également créé dans la même partition flash. Cette approche assure une exécution autonome du STM32 : sans dépendre d'un composant externe supplémentaire, elle garantit la cohérence des accès et simplifie la logistique de déploiement (un simple flashage du firmware suffit pour mettre à jour ou ajouter de nouveaux fichiers `.gto`).

## Carte SD

Broche SD	Fonction SD	Rôle en mode SPI	Exemple STM32G2431 KB	Remarques
1 CD/DAT0	CS / Card-Detect	CS (nSS)	PB0 (GPIO Out)	Connecter à un GPIO configuré en sortie ; résistance de tirage 10 kΩ vers 3,3 V.
2 CMD	MOSI / DI	SPI MOSI	PA7 (SPI1_MOSI)	Pull-up 10 kΩ conseillé pour l'initialisation en mode SPI.
3 VSS1	Masse	—	—	Relier directement au plan de masse du PCB.
4 VDD	2,7 – 3,6 V	Alimentation	—	Découplage local : 100 nF + 1 μF au plus près du connecteur.
5 CLK	SCLK	SPI SCK	PA5 (SPI1_SCK)	Série 33 Ω recommandée si la piste dépasse ~30 mm pour limiter les réflexions.
6 VSS2	Masse	—	—	Même connexion de masse que VSS1.
7 DAT0	MISO / DO	SPI MISO	PA6 (SPI1_MISO)	Pull-up 10 kΩ recommandé ; ligne à laisser en haute-impédance côté MCU.
8 DAT1	—	Non utilisé (SPI)	—	Laisser flottant ou pull-up 10 kΩ pour compatibilité future.
9 DAT2	—	Non utilisé (SPI)	—	Idem DAT1.
WP (option)	Write-Protect	GPIO In-put	PC13	Interrupteur de protection en écriture ; activer le pull-up interne.
CD (option)	Card-Detect	GPIO In-put	PB1	Contact d'insertion carte ; logique basse quand la carte est présente (selon support).

TABLE 4 : Brochage et fonctions de la carte SD

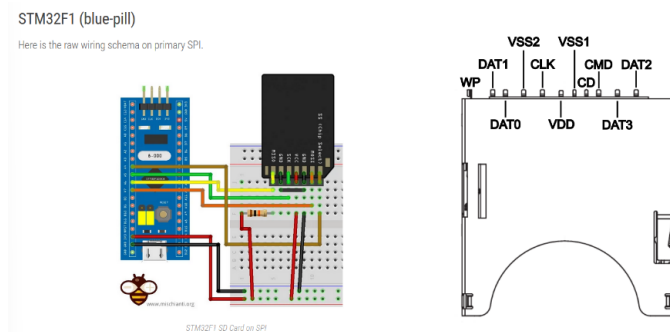


FIGURE 4 : Exemple d'export CSV des réponses



## **Interface et Interaction homme/machine**

Le système GTO a été conçu pour être simple à utiliser, même dans un cadre scolaire où le temps est souvent limité. L'idée est que n'importe qui puisse le prendre en main rapidement, sans avoir besoin de formation particulière. Pour ça, on a intégré quelques composants qui permettent d'interagir facilement avec l'appareil.

La carte SD est là pour enregistrer les résultats et les données des copies scannées. Elle peut être lue sur un ordinateur si besoin, ce qui rend le système pratique et adaptable.

### **Ecran OLED**

L'écran OLED tactile de 2,4 pouces est l'élément principal pour afficher les infos. Il sert à montrer directement sur place l'ID de l'élève, sa note. Cela évite d'avoir besoin d'un ordinateur à côté : tout est visible directement sur l'écran.

### **Codeur incrémental**

Pour naviguer entre les différentes copies, on utilise un codeur rotatif (comme un bouton qui tourne). Tourner vers la droite ou la gauche permet de passer d'un élève à un autre, et un clic sur le bouton peut servir à valider une action ou afficher plus de détails, selon les versions du logiciel.

### **LED**

Enfin, une petite LED verte s'allume pendant le traitement ou le chargement des données. Cela permet de savoir que le système est en train de travailler, sans avoir besoin de messages complexes à l'écran.

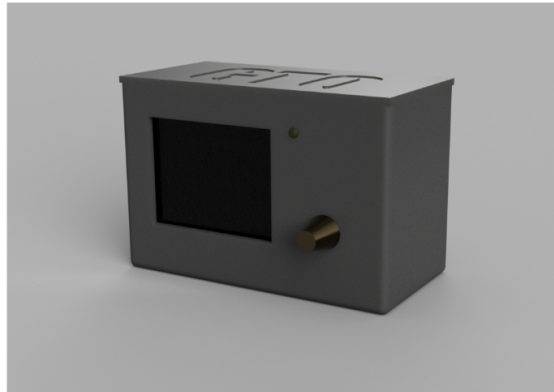


## Conception Assistée par Ordinateur (CAO)

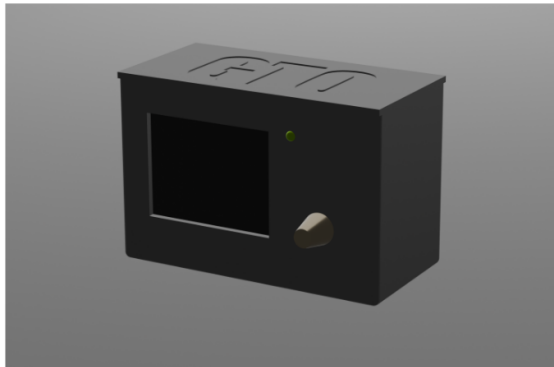
Pour notre produit, afin de cacher les câbles, nous avons dû créer une boîte sur Fusion 360.

Mais afin que nous puissions la montrer dans notre vidéo, nous l'avons recrée sur Blender.

**Rendu Fusion 360 :**



**Rendu Blender :**



Composants visibles : Codeur incrémental, led, écran OLED

Dans le cadre du projet GTO, nous devons dissimuler tous les câbles et offrir à notre produit une apparence soignée et professionnelle. Pour cela, nous avons conçu un boîtier sur Fusion 360, avant d'en créer une version « prête pour la vidéo » sur Blender. Cette démarche, en deux temps, nous a permis :

1. **De garantir la précision mécanique** (tolérances, positions des composants, épaisseurs de parois) grâce à Fusion 360.
2. **De soigner la présentation visuelle** (matériaux, éclairage, animation) grâce à Blender, sans compromettre le modèle technique d'origine.

# Modélisation détaillée sous Fusion 360

- **Analyse des contraintes**

- Dimensions de l'écran, du codeur incrémental, la LED et le lecteur de carte SD.
- Nécessité de loger : un codeur incrémental, une LED d'état, un écran OLED de 2", ainsi que le faisceau de câbles reliant ces éléments à la carte électronique.
- Besoin d'aérations minimales pour éviter la surchauffe de l'OLED et de la carte.

- **Workflow paramétrique**

- **Esquisses 2D** : création du profil de base ( $90 \times 45$  mm) avec rayons internes de 2 mm pour limiter les contraintes lors de l'impression 3D.
- **Extrusions** : parois de 2 mm pour la rigidité, puis 30% de remplissage interne afin d'économiser de la matière.
- **Perçages et découpes** :
  - \* Fente de  $43 \times 33.5$  mm pour l'écran OLED.
  - \* Trou de 5 mm pour la LED.
  - \* un trou de 8 mm traversant + chanfrein pour l'axe du codeur incrémental. Qui n'a pas été reçu donc il n'a pas été ajouté dans le design final.
- **Finitions** : congés externes de 1 mm pour casser les arêtes et améliorer la solidité après impression.

- **Constructions**

Figure prémontée (sans dos et couvercle)

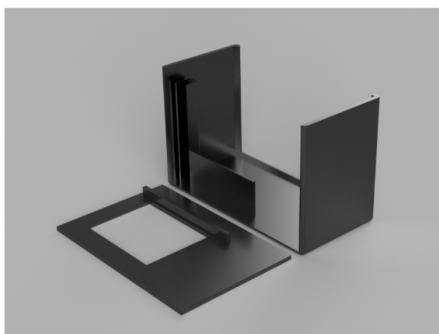
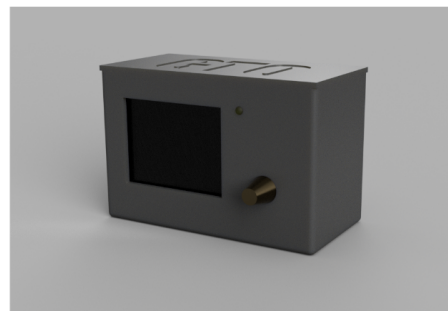


Figure montée



## Passage et optimisation sous Blender

Une fois le modèle validé d'un point de vue mécanique, nous l'avons converti en **FBX** afin de conserver les lissages de surface.

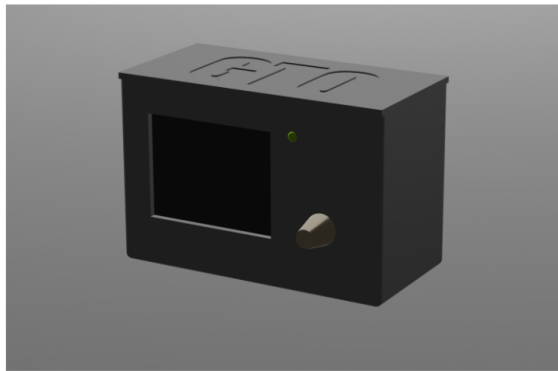
- **Nettoyage & hiérarchisation**

- Regroupement des éléments par collection : *Boîtier*, *Écran*, *LED*, *Codeur*.

- **Texture réaliste**

- **Boîtier** : matériau plastique ABS mat, indice spéculaire 0,35, couleur RAL 7016 (gris anthracite).
- **Écran OLED** : Simule un écran noir.
- **Bouton du codeur** : cuivre brossé.
- **LED** : sous-surface scattering pour l'effet diffus.

- **Rendus**



- **Animation courte**

- Rotation à 360° du boîtier en 8 s (Bezier ease in-out) pour la vidéo de présentation.

# Marketing

Une vidéo promotionnelle a été conçue dans le but de produire un contenu orienté marketing, visant à valoriser efficacement le produit auprès de son public cible.

La première scène illustre la problématique rencontrée par les enseignants face à la correction manuelle des copies, mettant en lumière le besoin auquel le produit répond. Cette introduction est pensée pour capter l'attention et favoriser une identification immédiate.

La présentation du produit s'appuie sur une combinaison de médias variés. Une modélisation 3D réalisée avec Fusion et Blender intègre des animations et rendus soignés pour mettre en avant les aspects techniques de l'appareil. Des plans aériens captés par drone apportent une dimension moderne au montage, complétés par des séquences filmées à la caméra.

La conclusion met en scène un enseignant satisfait, symbolisant le bénéfice tangible et valorisant du produit, renforçant ainsi l'impact marketing global. Ce projet a mobilisé un ensemble de compétences techniques — modélisation 3D, captation vidéo, montage et animation — afin de produire un support marketing professionnel et convaincant.

Le montage a été réalisé dans DaVinci Resolve, où l'utilisation d'outils d'animation de texte et de transitions a permis de dynamiser et d'enrichir visuellement la vidéo. Les éléments sonores proviennent de la plateforme Pixabay.

# Résultats

## Tests

- Génération QCM sur Python fonctionnelle
- Conversion de QCM en `.gto` fiable sur feuilles bien scannées
- Lecture de `.gto` et Export CSV : opérationnel sous C non embarqué

Problème restant : lecture de `.gto` sur STM32 (carte SD instable)

Écran fonctionnel, on peut voir que la sélection sera en rouge et il est possible de faire défiler la sélection de haut en bas en incrémentant. De futurs tests sont à venir avec des boutons.

Un exemple avec l'orientation de base de l'écran :



Le rendu Fusion 360 offre une visualisation technique rapide et fidèle aux côtes, tandis que le rendu Blender est adapté aux supports marketing soit notre vidéo finale.

En combinant les forces de ces deux logiciels, nous avons réussi à :

- **Accélérer les itérations mécaniques** sans sacrifier la qualité visuelle.
- **Différencier clairement** la phase d'ingénierie (précision, assemblages) et la phase de communication (textures, lumière, animation).
- **Garantir l'intégrité fonctionnelle** de chaque composant tout en offrant une présentation professionnelle et attrayante de notre boîtier GTO.

## Problèmes rencontrés

### La carte SD

Avec du recul, nous pensons avoir complexifié inutilement certaines étapes. Certes, apprendre le rôle de chaque branchement s’est révélé instructif et formateur. Cependant, la phase de soudure s’est avérée fastidieuse, et les tests à l’oscilloscope n’ont pas fourni de résultats exploitables — les signaux de sortie restant désespérément à 0. Ces dysfonctionnements proviennent probablement d’erreurs de notre part, notamment dans le câblage. Une approche plus judicieuse aurait été d’opter dès le départ pour un lecteur de carte SD intégré et préconfiguré (comme celui présenté [https ://www.ebay.fr/itm/394642236066](https://www.ebay.fr/itm/394642236066))

### L’écran

Nous avons cherché des projets sur internet sur lesquels nous pourrions nous inspirer pour faire ce que nous souhaitons (au moins afficher une image), nous avons trouver un dossier avec 205 projets mais les éplucher à la main prends du temps car nous devons les tester un à un en les adaptant à notre carte STM32. Temps perdu : 20h.

Nous avons perdu plus de 6h de travail à cause de l’erreur : “Failed to start GDB server” qui nous a donc obligé à désinstaller et réinstaller STM32 IDECube.

### Configuration de la carte

Dès le démarrage du projet, nous avons rencontré un obstacle lié à la configuration de la carte STM32G431KB dans STM32CubeIDE. L’une des premières difficultés consistait à bien identifier quelles broches pouvaient être utilisées pour chaque périphérique, et à les activer correctement via l’interface graphique. Certaines fonctions clés, comme le SPI ou le timer, étaient mal configurées ou assignées à des broches inadaptées, entraînant des erreurs de compilation ou empêchant la génération du code.

Ce contretemps a ralenti notre progression initiale, mais il nous a également permis d’approfondir notre compréhension des pins alternatifs, de la gestion des conflits dans CubeIDE, et de l’importance d’une configuration soignée dès le départ. Afin de ne pas rester bloqués, nous avons utilisé temporairement un projet déjà fonctionnel pour amorcer le développement en parallèle. Une fois la configuration maîtrisée, nous avons pu construire la suite du projet sur une base propre et stable.

### Boîtier

La conception du boîtier s’est avérée complexe à cause de la nécessité d’un assemblage par emboîtement, demandant une conception minutieuse pour éviter les ruptures de pièces.

## **Encodeur incrémental**

L'utilisation de notre encodeur était essentielle pour notre projet, afin d'utiliser le potentiel de votre écran pour faire défiler le nom des fichiers et les sélectionner. Malheureusement nous ne l'avons pas reçu.

## **Boutons poussoirs**

Pour remplacer l'encodeur incrémental, nous avons décidé d'ajouter des boutons poussoirs. Ceux-ci faisaient sauter l'écran car nous avons oublié d'ajouter une petite résistance ( $330\Omega$ ) et par la suite l'implémentation fut plutôt linéaire.

## Perspectives

Le projet GTO pose une base solide pour automatiser la correction des QCM et gagner du temps dans le travail des enseignants.

### À **court** terme (fonctionnalités prioritaires et fiabilisation)

#### **Fiabilisation matérielle**

L'objectif principal est de garantir la lecture fiable des fichiers `.gto` via la carte SD. Cela passe par le remplacement du lecteur par un modèle plus stable et la révision du câblage. Le codeur rotatif sera également remplacé ou accompagné de boutons robustes pour garantir une navigation fluide.

#### **Amélioration de l'interface utilisateur embarquée**

Le système OLED/codeur sera optimisé pour une meilleure ergonomie. L'affichage sera enrichi de retours visuels clairs (fichiers lus, progression, erreurs), pour une utilisation intuitive sans formation préalable.

#### **Refonte du boîtier**

Une nouvelle version du boîtier, plus robuste, sera conçue, avec une meilleure accessibilité aux ports et un design adapté à un usage pédagogique.

### À **moyen** terme (fonctionnalités avancées et connectivité)

#### **Évolutions logicielles**

Le traitement d'image sera porté vers OpenCV en Python ou C++, avec pour objectif une exécution plus rapide, une gestion d'images couleur, et une compatibilité avec des webcams en temps réel (à voir pour la lecture rapide). Un module d'apprentissage automatique (réseaux de neurones) sera expérimenté pour détecter les cases cochées de manière plus souple, sans grille fixe, et voir la progression des élèves au fil des années.

#### **Connectivité & synchronisation**

L'ajout d'une liaison Bluetooth/Wi-Fi permettra de transférer automatiquement les résultats vers un PC ou une base de données pédagogique. Une synchronisation en temps réel vers des plateformes comme Pronote ou Moodle pourra être envisagée.

#### **Interface graphique externe**

Une interface graphique (Tkinter, Qt ou Web) sera développée pour permettre aux enseignants de gérer les lots de QCM, consulter les résultats détaillés, trier, exporter ou suivre les performances des élèves de manière conviviale.

#### **Identification automatisée**

L'intégration d'un QR code ou d'un module OCR permettra de lire automatiquement les identifiants élèves/examens, supprimant la saisie manuelle et les risques d'erreurs associés.



## À long terme (vision produit et déploiement à grande échelle)

### **Correction par webcam en direct**

Une version totalement autonome basée sur une caméra USB pourra corriger les copies à la volée, sans passer par des scans préalables, rendant le système encore plus fluide et mobile.

### **Détection multi-copies et traitement par lots**

L'implémentation d'un mode "scanner de pile" permettra de corriger plusieurs feuilles d'affilée en une seule session, utile pour les examens de masse. Le système détectera automatiquement la fin de traitement et passera à la copie suivante.

### **Intégration dans les établissements**

Un kit "clé en main" pourra être proposé aux établissements : boîtier robuste, interface web dédiée, synchronisation sécurisée, et modules de suivi pédagogique. Ce déploiement s'accompagnera d'une documentation, d'un support logiciel open-source et d'un modèle de partenariat pour l'amélioration continue.

## Conclusion

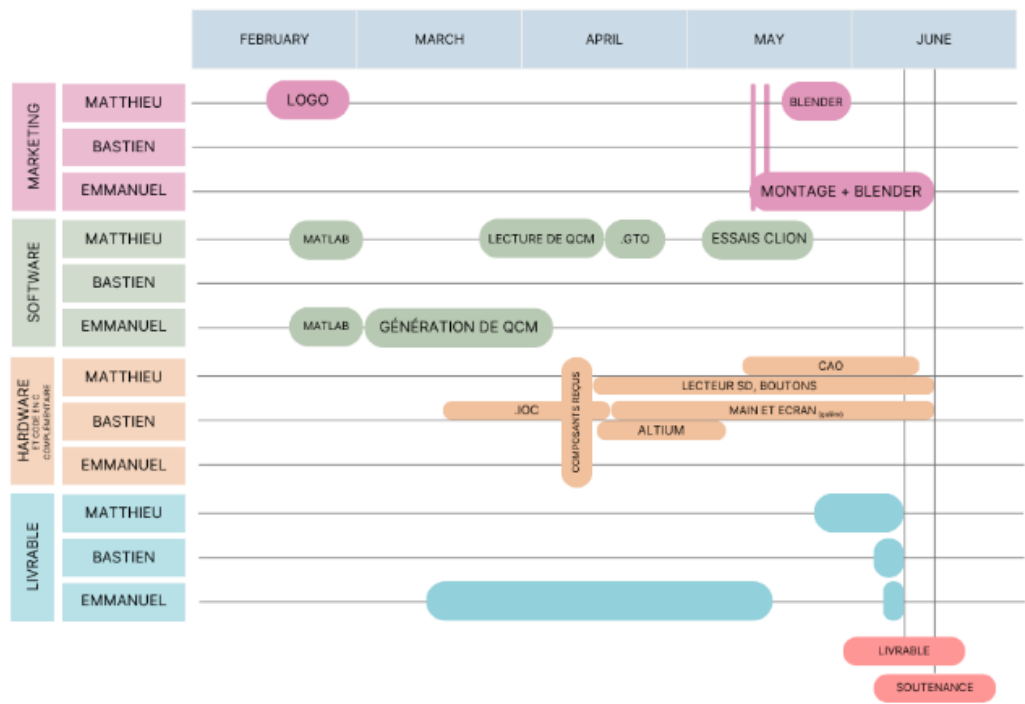
Le projet GTO a abouti à un prototype embarqué fiable, autonome et économique, capable d'automatiser la correction de QCM avec un bon niveau de précision. En combinant traitement d'image, microcontrôleur STM32 et affichage local, il répond efficacement aux contraintes du milieu éducatif. Malgré quelques limitations matérielles, notamment sur la lecture de la carte SD et le codeur incrémental perdu en livraison, les résultats confirment la viabilité du concept. Ce projet constitue ainsi une base solide pour des développements ultérieurs visant à démocratiser la correction automatisée dans les établissements scolaires.

**Merci à nos deux professeurs, Antoine Bulard et Jordan Dufresne, pour leur accompagnement.**

# Annexes

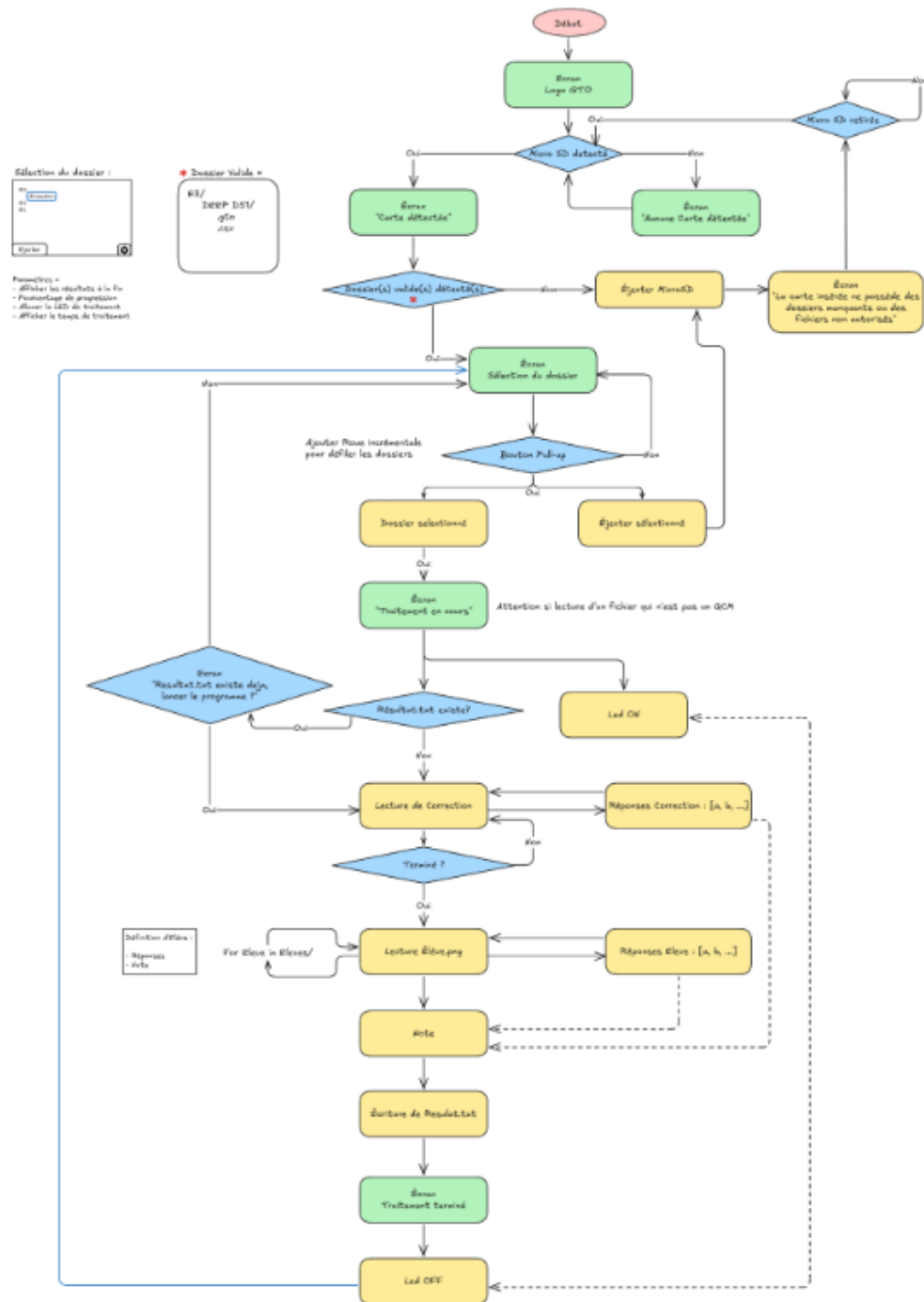
## Diagramme de Gant

Grands axes du projet GTO

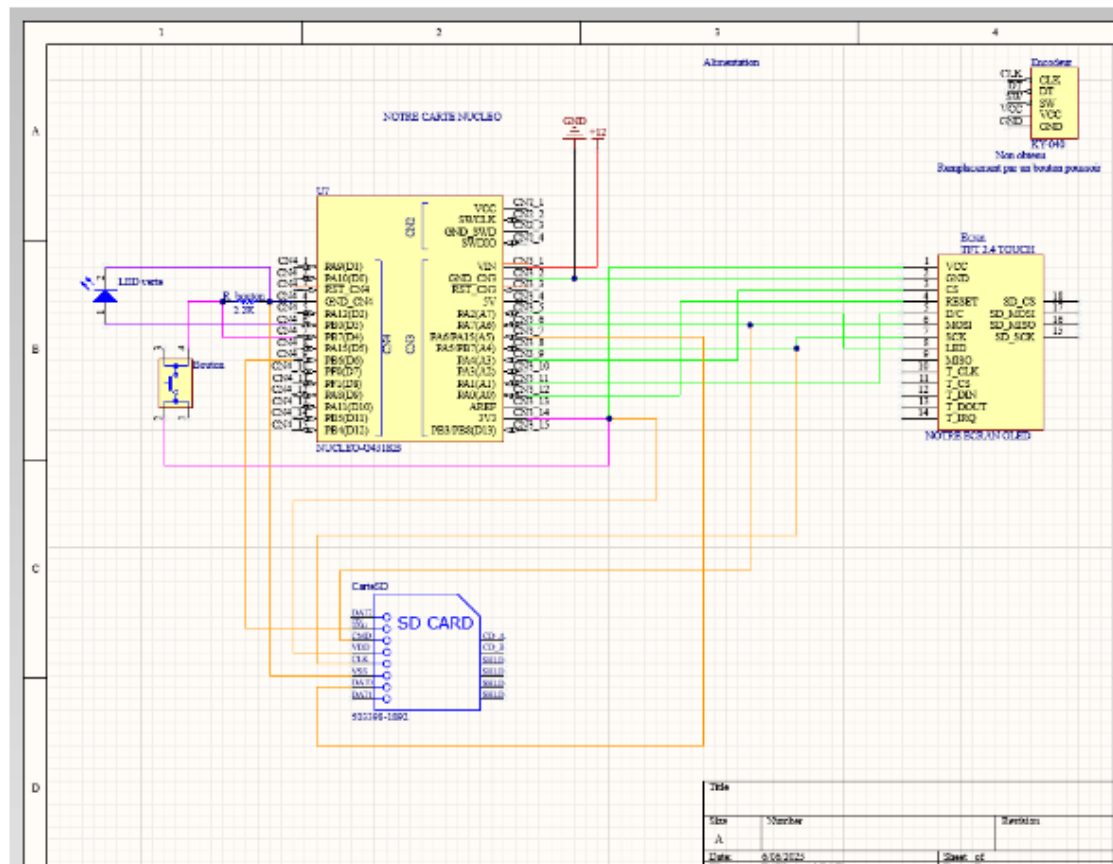


## Schéma Bloc Architecture logicielle

Attention si la Carte SD est retirée trop tôt !!!



# Altium



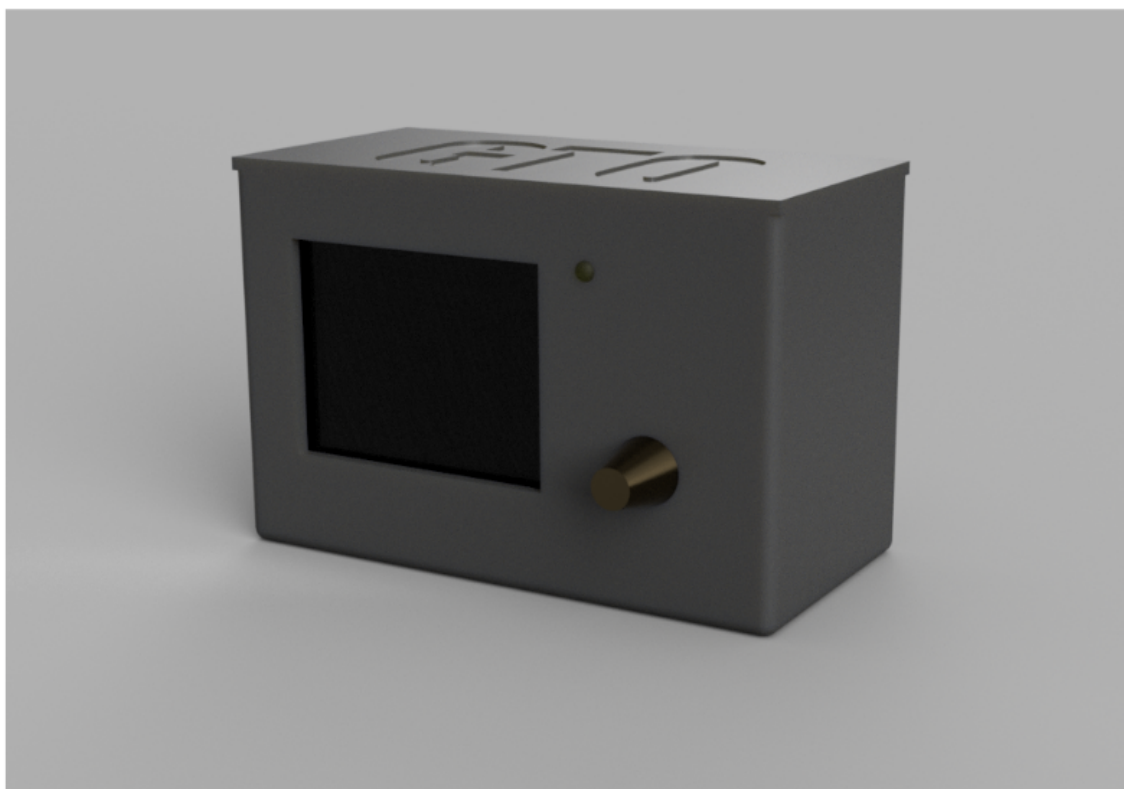
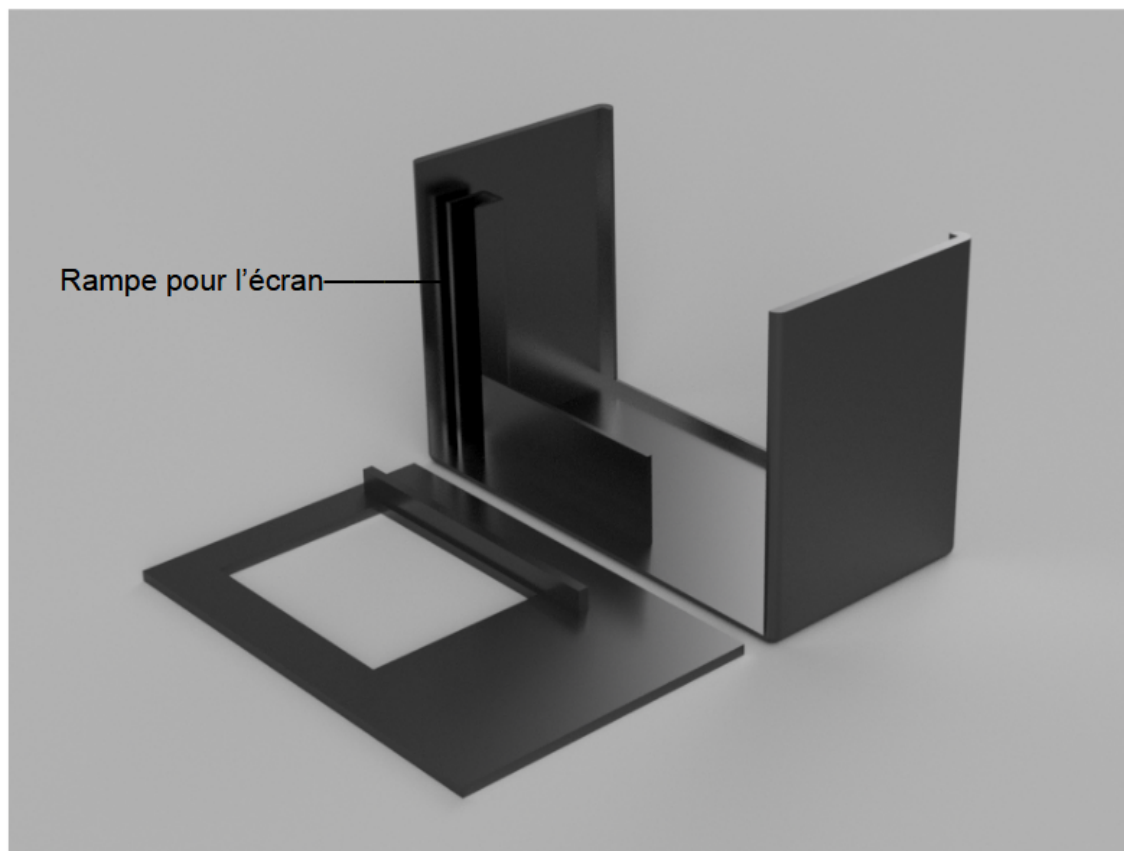
## Tableau collaboratif

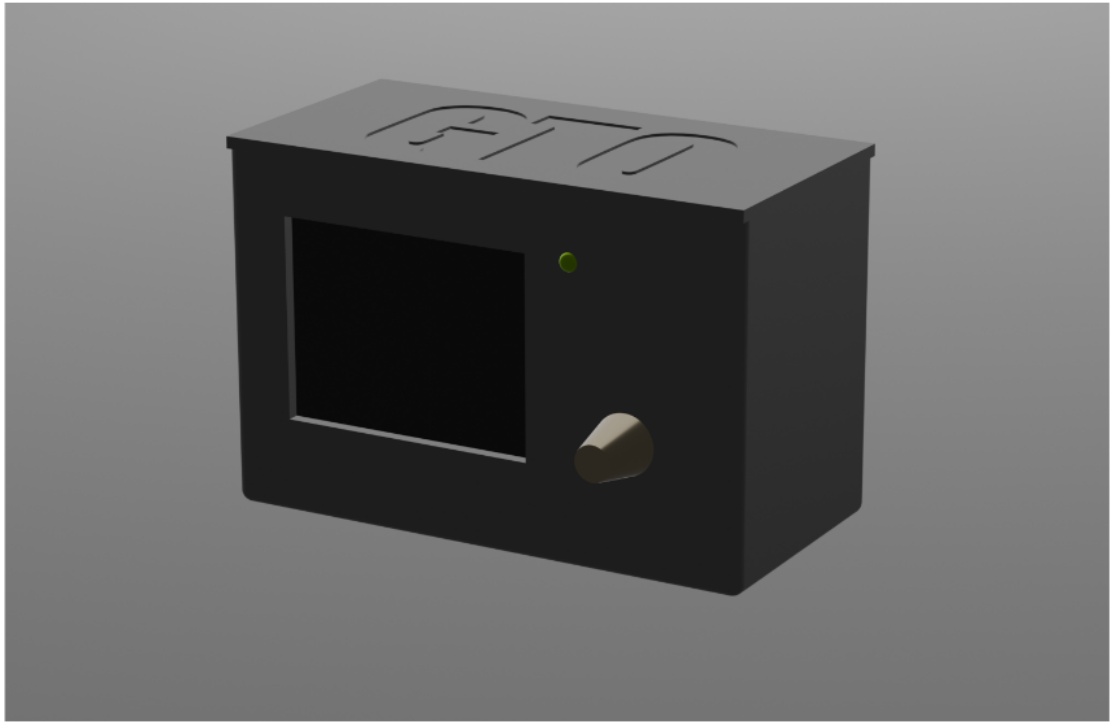
<https://excalidraw.com/#room=50f86837b22fc74da657,5Z5Pb0o4Y560WGFS0vLuXg>

## Composants

- Écran : <https://www.waveshare.com/2inch-lcd-module.htm>
- Aide : <https://blog.embeddedexpert.io/?p=1215>
- Carte SD : [https://www.mouser.fr/datasheet/2/1628/sd\\_1\\_a-3510693.pdf](https://www.mouser.fr/datasheet/2/1628/sd_1_a-3510693.pdf)
- Tuto : <https://mischianti.org/how-to-use-sd-card-with-stm32-and-sdfat-library/>
- Codeur incrémental : <https://www.robotchbd.com/product/electronics-components/resistor/hw-040-rotary-encoder-module/>


CAO





## Connections d'une Nucleo-G431KB

Figure 12. ARDUINO® connector pinout

NUCLEO-G431KB					
PA9	1	D1	VIN	1	VIN
PA10	2	D0	GND	2	GND
NRST	3	NRST	NRST	3	NRST
GND	4	GND	+5V	4	+5V
PA12	5	D2	A7	5	PA2
PB0	6	D3	A6	6	PA7
PB7	7	D4	A5	7	PA6/PA15
PA15	8	D5	A4	8	PA5/PB7
PB6	9	D6	A3	9	PA4
PF0	10	D7	A2	10	PA3
PF1	11	D8	A1	11	PA1
PA8	12	D9	A0	12	PA0
PA11	13	D10	AREF	13	AVDD
PB5	14	D11	+3V3	14	+3v3
PB4	15	D12	D13	15	PB3/PB8
CN4			CN3		
					

En dessous de 220 mA, la carte ne fonctionne pas