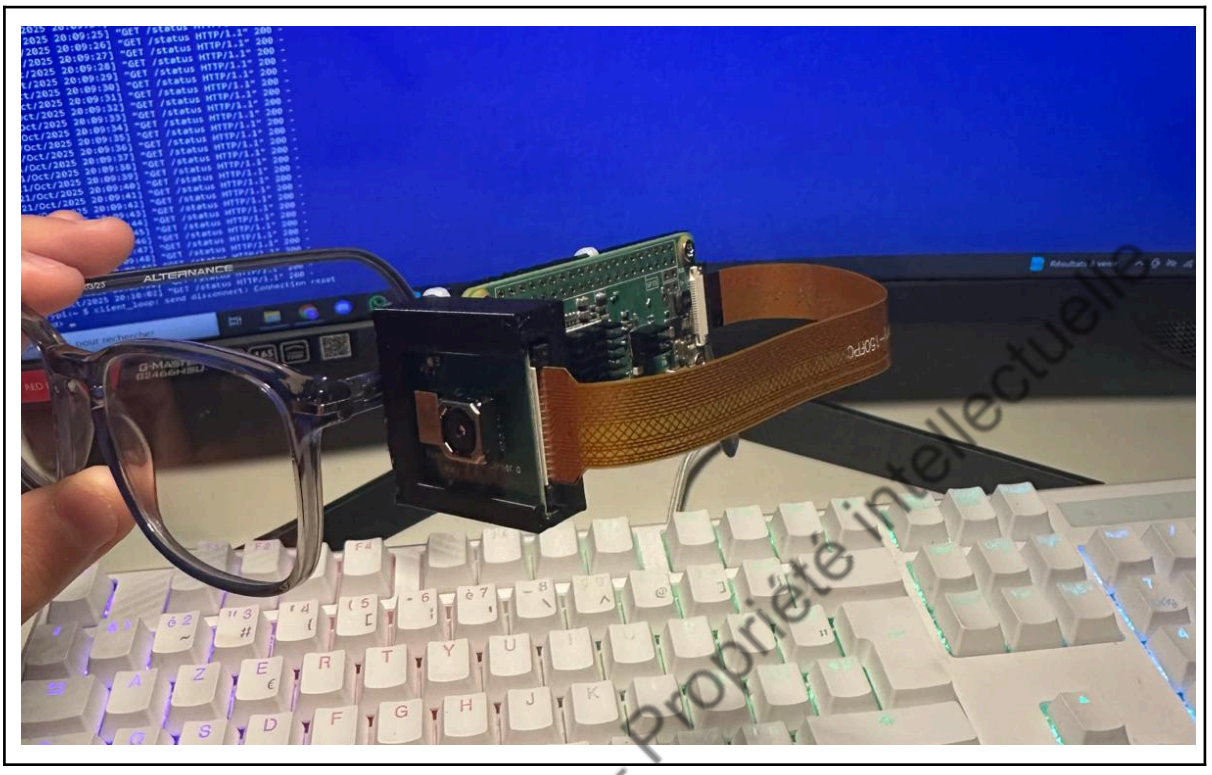


Conception de Lunettes Connectées

Thibaud Margotteau

Système d'enregistrement sur une monture de lunette

Mise à jour - Novembre 2025

Sommaire

1. Présentation du Projet

1.1 Contexte et Motivation

Depuis l'apparition des Google Glass en 2013, les lunettes connectées représentent une vision futuriste de la technologie portable. Plus récemment, les Ray-Ban Meta ont démocratisé ce concept en proposant un dispositif élégant capable de filmer discrètement en première personne. C'est dans ce contexte que j'ai entrepris de concevoir mon propre prototype de lunettes caméra, non pas pour concurrencer ces produits commerciaux, mais pour comprendre leur fonctionnement interne et démontrer qu'il est possible de reproduire leurs fonctionnalités principales avec du matériel accessible.

L'idée centrale de ce projet était de créer un système embarqué capable d'enregistrer des vidéos à la première personne tout en restant contrôlable à distance. Au-delà de l'aspect technique, ce projet représentait pour moi une opportunité d'explorer plusieurs domaines simultanément : la programmation embarquée, l'impression 3D et la communication réseau. Je voulais concevoir quelque chose de fonctionnel et surtout de reproductible par n'importe qui disposant d'un budget modeste et d'une imprimante 3D.

1.2 Description du Système

Le dispositif final se présente sous la forme d'un module compact fixé sur la branche d'une paire de lunettes ordinaire. Au cœur de ce système se trouve un Raspberry Pi Zero W, un micro-ordinateur de la taille d'une carte de crédit intégrant le Wi-Fi et le Bluetooth. Ce composant est couplé à une caméra miniature connectée via le port CSI, qui permet une transmission vidéo à haute vitesse directement vers le processeur.

L'ensemble est alimenté par une batterie externe USB, ce qui garantit une autonomie suffisante pour plusieurs sessions d'enregistrement sans dépendre d'une prise secteur. Le boîtier qui contient ces composants a été entièrement conçu sur SolidWorks puis imprimé en 3D avec du filament PLA, un matériau léger et résistant. Le système fonctionne en mode totalement sans fil : la caméra enregistre les vidéos sur une carte microSD, tandis qu'une interface web accessible depuis n'importe quel appareil connecté au même réseau Wi-Fi permet de démarrer et d'arrêter l'enregistrement à distance.

Pour faciliter la récupération des vidéos, j'ai également intégré un bot Telegram qui reçoit automatiquement les fichiers une fois la conversion terminée. Ce système élimine le besoin de retirer physiquement la carte SD ou de passer par des transferts de fichiers complexes. Le système est donc entièrement autonome et peut fonctionner dans des conditions réelles d'utilisation, exactement comme un produit commercial.

1.3 Objectifs Techniques

L'objectif principal de ce projet était de réussir à enregistrer des vidéos fluides en haute définition, avec une résolution de 720p et une fréquence de 30 images par seconde. Cette résolution représente un bon compromis entre qualité d'image et capacité de traitement du Raspberry Pi Zero W, qui reste un appareil très limité en puissance de calcul avec seulement 512 Mo de mémoire vive.

Je voulais également développer une interface de contrôle intuitive et accessible depuis n'importe quel appareil. L'idée était qu'un simple navigateur web suffise pour piloter entièrement le système, sans nécessiter d'application dédiée ou de logiciel spécifique. Cette interface devait permettre de démarrer un enregistrement, de l'arrêter proprement, et de suivre en temps réel l'état du système grâce à des logs affichés directement à l'écran.

Un autre objectif important était l'automatisation complète du pipeline vidéo. Je voulais que le système soit capable de capturer la vidéo au format H.264, de la convertir automatiquement en MP4 pour garantir sa compatibilité avec tous les lecteurs, puis de la transférer vers un canal Telegram privé sans aucune intervention manuelle. Cette chaîne de traitement devait être fiable, robuste et reproductible à chaque utilisation.

Enfin, la dimension mécanique était tout aussi essentielle. Le boîtier devait être suffisamment léger pour ne pas déséquilibrer la monture des lunettes, tout en restant assez solide pour protéger les composants électroniques. L'objectif était de ne pas dépasser 50 grammes pour l'ensemble du module, hors batterie, afin de garantir un confort d'utilisation acceptable.

1.4 Contraintes Identifiées

Dès le début du projet, j'ai identifié plusieurs contraintes majeures qui allaient influencer mes choix techniques. Ces contraintes se répartissent en plusieurs catégories :

Type de Contrainte	Description	Impact sur le Projet
Matérielle	Raspberry Pi Zero W limité (512 Mo RAM, CPU 1 GHz)	Résolution maximale 720p, pas de multitâche intensif
Thermique	Surchauffe du processeur ARM11 sans dissipateur	Throttling après 15 min, température jusqu'à 78°C
Énergétique	Consommation caméra + Wi-Fi ≈ 1,2 A	Autonomie réduite à 1h45 avec batterie 5000 mAh
Logicielle	Incompatibilités entre raspivid/libcamera/rpicam	Tests multiples nécessaires, détection automatique obligatoire
Stockage	Carte SD limitée, accès physique difficile	Solution de transfert automatique indispensable

La première et la plus importante concernait les performances matérielles du Raspberry Pi Zero W. Avec son processeur monocœur ARM11 cadencé à 1GHz et ses 512 Mo de RAM, cet ordinateur est très limité comparé aux standards actuels. Cette faiblesse se traduit directement par une incapacité à encoder des vidéos en 1080p sans ralentissements importants. J'ai donc été contraint de réduire la résolution à 720p pour obtenir un flux vidéo stable.

La gestion thermique constituait une autre contrainte non négligeable. Le processeur du Raspberry Pi Zero W chauffe rapidement lors d'opérations intensives comme l'encodage vidéo. En l'absence de dissipateur thermique ou de ventilateur, la température peut atteindre 78 degrés Celsius après seulement quelques minutes d'enregistrement continu. Au-delà d'un certain seuil, le système active automatiquement un mécanisme de protection appelé throttling, qui réduit la fréquence du processeur pour limiter la surchauffe. Cela se traduit par des baisses de performances et des risques de vidéos saccadées.

L'autonomie énergétique représentait également un défi technique. L'ensemble caméra et Raspberry Pi consomme environ 1,2 ampère en fonctionnement normal, ce qui limite drastiquement la durée d'utilisation. Avec une batterie externe de 5000 mAh, l'autonomie réelle se situe autour d'une heure et quarante-cinq minutes, bien en dessous des quatre heures proposées par les Ray-Ban Meta.

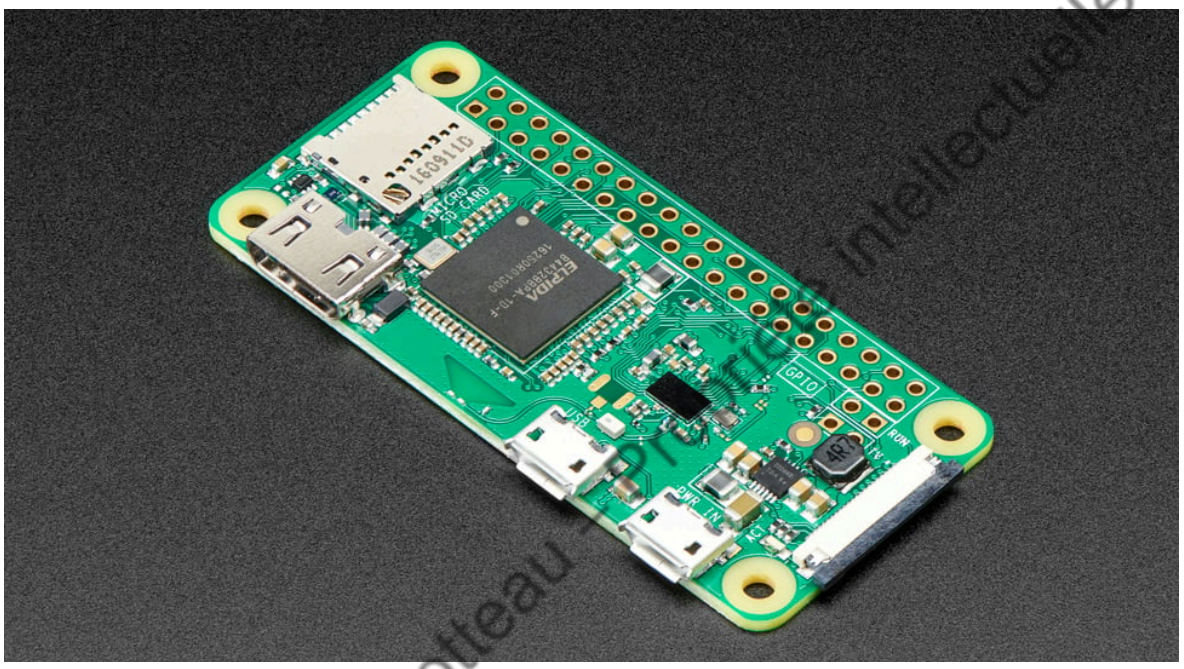
Sur le plan mécanique, l'espace disponible sur une branche de lunettes est extrêmement réduit. Il fallait faire tenir la caméra, le Raspberry Pi, les câbles et les fixations dans un volume minimal, tout en assurant une bonne ventilation et en évitant que l'ensemble ne soit trop lourd ou disgracieux.

Enfin, la dernière contrainte concerne la compatibilité logicielle. Les bibliothèques permettant d'accéder à la caméra sur Raspberry Pi ont beaucoup évolué ces dernières années. Certaines commandes comme raspivid sont devenues obsolètes, tandis que d'autres comme libcamera-vid ou rpicam-vid ne fonctionnent pas toujours selon la version de l'OS ou le modèle de caméra utilisé.

2. Conception Préliminaire

2.1 Analyse et Sélection du Matériel

Avant de me lancer dans la réalisation concrète, j'ai consacré un temps important à l'étude et à la sélection des composants. Le choix du Raspberry Pi Zero W s'est imposé naturellement pour plusieurs raisons. D'abord, c'est le seul micro-ordinateur de cette gamme de prix qui intègre directement le Wi-Fi et le Bluetooth dans un format ultra-compact de 65 millimètres sur 30.



Source: adafruit.com

Composants Principaux Sélectionnés

Caractéristique	Spécification	Justification
Processeur	ARM11 1 GHz monocœur	Suffisant pour encodage 720p
RAM	512 Mo	Minimum pour Flask + rpicam-vid
Connectivité	Wi-Fi 802.11n, Bluetooth 4.1	Contrôle sans fil intégré
Ports	1× CSI, 40× GPIO, micro-USB	Caméra + alimentation
Dimensions	65 × 30 × 5 mm	Format ultra-compact
Consommation	150 mA (repos), 350mA (actif)	Autonomie acceptable
Prix	~15€	Budget accessible

Module Caméra Compatible CSI

J'ai opté pour une caméra compatible avec le port CSI du Raspberry Pi, achetée sur AliExpress pour environ huit euros. Ce choix économique s'est révélé être un compromis acceptable : la caméra fonctionne correctement en conditions normales d'éclairage et offre une résolution de 5 mégapixels, largement suffisante pour de la vidéo en 720p.

Caractéristiques techniques de la caméra :

- Capteur : OV5647 (clone) 5 MP
- Résolution vidéo max : 720p60
- Interface : CSI (Camera Serial Interface)
- Longueur câble ruban : 15 cm
- Champ de vision : 160° (fish-eye léger)
- Prix : ~8€ (AliExpress) Le principal inconvénient de ce clone non officiel réside dans ses pilotes parfois capricieux. La détection par le système n'est pas toujours immédiate et certaines commandes standard ne fonctionnent pas du premier coup. J'ai dû apprendre à travailler avec ces imperfections et à prévoir des solutions de secours dans le code.

L'alimentation du système repose sur une batterie externe USB classique de 5000 milliampères-heures. Ce type de batterie présente l'avantage d'être abordable, facilement remplaçable et universellement disponible.

Pour la partie structurelle, j'ai utilisé du filament PLA pour l'impression 3D. Ce matériau plastique biosourcé présente plusieurs avantages : il est léger, rigide, facile à imprimer et relativement peu coûteux. Sa température de fusion relativement basse permet de l'utiliser sur des imprimantes grand public sans risque de déformation. L'inconvénient du PLA est sa sensibilité à la chaleur : au-delà de 60 degrés Celsius, il commence à se ramollir. Cela n'a pas posé de problème dans ce projet car le Raspberry Pi, bien que chaud, ne dépasse pas cette température sur sa surface externe.

L'ensemble de ces choix matériels a été guidé par trois critères principaux : le coût, la disponibilité et la compatibilité. Je voulais que ce projet reste accessible financièrement et reproductible par n'importe qui, sans nécessiter de composants rares ou trop spécialisés. Le budget total pour l'ensemble des composants électroniques et mécaniques s'élève à environ 25 euros (sans la batterie), ce qui démontre qu'il est possible de créer des objets connectés sophistiqués sans investissement massif.

2.2 Choix du Langage de Programmation

Pour la partie logicielle, j'ai naturellement choisi Python comme langage de développement. Ce choix repose sur plusieurs constats pratiques que je peux résumer ainsi :

Avantages de Python pour ce projet :

- Langage natif du Raspberry Pi avec support optimal
- Syntaxe claire permettant un développement rapide
- Pas de compilation nécessaire (tests immédiats)
- Écosystème riche en bibliothèques adaptées
- Gestion native des appels système Linux
- Communauté active et documentation abondante

Bibliothèques Python utilisées :

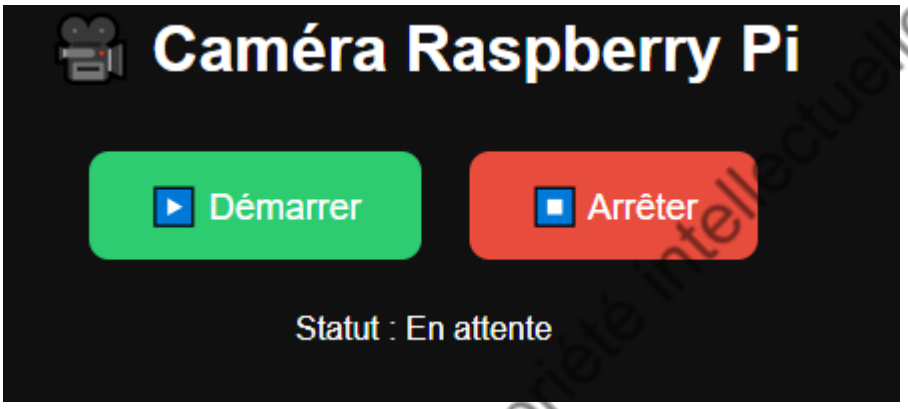
```
# Serveur web et interface utilisateur
flask==2.3.0                # Framework web léger

# Communication Telegram
python-telegram-bot==20.0       # API officielle Telegram

# Gestion système et processus
import subprocess           # Lancement rpicam-vid et ffmpeg
import os                   # Manipulation fichiers
import time                 # Gestion timestamps
import datetime             # Horodatage vidéos
import shutil               # Détection commandes

# Utilitaires
import json                 # Configuration
```


2.3 Architecture de Communication

La communication entre le Raspberry Pi et les appareils de contrôle repose sur plusieurs protocoles complémentaires. Le protocole SSH constitue la base de cette architecture. SSH, pour Secure Shell, est un protocole de communication chiffré qui permet de prendre le contrôle à distance d'un appareil via le réseau. Grâce à cette connexion sécurisée, je peux exécuter des commandes, transférer des fichiers, surveiller les processus et même redémarrer le système, le tout depuis mon ordinateur principal sans avoir besoin de connecter un clavier ou un écran au Raspberry Pi.

La configuration de SSH a été réalisée dès la première installation du système d'exploitation. J'ai généré une paire de clés cryptographiques ED25519, considérées comme plus sûres que les anciennes clés RSA, et j'ai désactivé l'authentification par mot de passe pour renforcer la sécurité. Cela garantit que seuls les appareils autorisés peuvent accéder au système. J'ai également attribué une adresse IP fixe au Raspberry Pi sur mon réseau local, ce qui évite de devoir rechercher son adresse à chaque connexion.

Au-dessus de cette couche SSH, j'ai développé une interface web accessible via un navigateur. Cette interface, construite avec le framework Flask, tourne sur le port 5000 du Raspberry Pi. N'importe quel appareil connecté au même réseau Wi-Fi peut y accéder en tapant simplement l'adresse IP suivie du numéro de port dans la barre d'adresse. L'interface propose deux boutons principaux : un pour démarrer l'enregistrement et un autre pour l'arrêter. Un système de logs en temps réel affiche l'état actuel du système, les erreurs éventuelles et le déroulement de chaque opération.



L'avantage majeur de cette interface web est son universalité. Qu'on utilise un smartphone, une tablette ou un ordinateur, sous iOS, Android, Windows ou Linux, le navigateur web suffit. Aucune application dédiée n'est nécessaire, ce qui simplifie grandement l'utilisation et élimine les problèmes de compatibilité entre plateformes. De plus, Flask étant un framework très léger, il consomme peu de ressources et n'impacte pas significativement les performances d'enregistrement.

Enfin, la dernière couche de communication passe par Telegram. J'ai créé un bot personnel via le service BotFather de Telegram, qui m'a fourni un token d'authentification unique. Ce bot est relié à un canal privé où les vidéos sont automatiquement envoyées après conversion. L'avantage de cette approche est qu'elle fonctionne même lorsque je ne suis plus connecté au réseau local du Raspberry Pi. Tant que le système a accès à Internet, il peut envoyer les vidéos sur Telegram, et je peux les consulter depuis n'importe où.

2.4 Système d'Exploitation

Le choix du système d'exploitation constituait une décision fondamentale pour garantir la stabilité et l'efficacité du dispositif. J'ai opté pour Raspberry Pi OS Lite, une version allégée de la distribution Linux officielle du Raspberry Pi. Cette variante se distingue par l'absence totale d'interface graphique, ce qui libère une quantité importante de mémoire vive et de puissance de calcul. Sur un système aussi limité que le Raspberry Pi Zero W, chaque mégaoctet de RAM compte, et éliminer l'environnement graphique permet de consacrer la quasi-totalité des ressources à l'enregistrement vidéo.

Raspberry Pi OS Lite présente également l'avantage d'être extrêmement stable et parfaitement documenté. Basé sur Debian, l'une des distributions Linux les plus anciennes et les plus fiables, il bénéficie d'un support communautaire exceptionnel et d'une compatibilité maximale avec le matériel Raspberry Pi. Les outils nécessaires à la gestion de la caméra, comme les bibliothèques libcamera, sont préinstallés ou facilement installables via le gestionnaire de paquets apt.

L'installation s'est faite via le logiciel Raspberry Pi Imager, qui simplifie grandement le processus de création d'une carte SD bootable. J'ai pu préconfigurer directement le réseau Wi-Fi et activer SSH avant même le premier démarrage, en plaçant simplement quelques fichiers de configuration dans la partition de démarrage. Cette approche headless, c'est-à-dire sans écran ni clavier, correspond parfaitement à l'usage final du système où le Raspberry Pi doit fonctionner de manière autonome.

3. Développement et Réalisation

3.1 Phase de Programmation

La programmation du système a représenté la partie la plus longue et la plus exigeante du projet. J'ai commencé par des tests simples de la caméra directement en ligne de commande, avant d'intégrer progressivement toutes les fonctionnalités dans un script Python complet.

Tests Initiaux de la Caméra

Séquence de commandes testées et leurs résultats :

```
# Tentative 1 : raspivid (ancienne méthode)
raspivid -o test.h264 -t 5000
# Résultat : "command not found"
```

```
# Tentative 2 : libcamera-vid
libcamera-vid -o test.h264 -t 5000
# Résultat : Détection OK mais freeze aléatoire
```

```
# Tentative 3 : rpicam-vid (solution finale)
rpicam-vid -o test.h264 -t 0 --width 1280 --height 720 --framerate 30 -n
# Résultat : Fonctionnement stable
```

Fonction de détection automatique de la commande caméra :

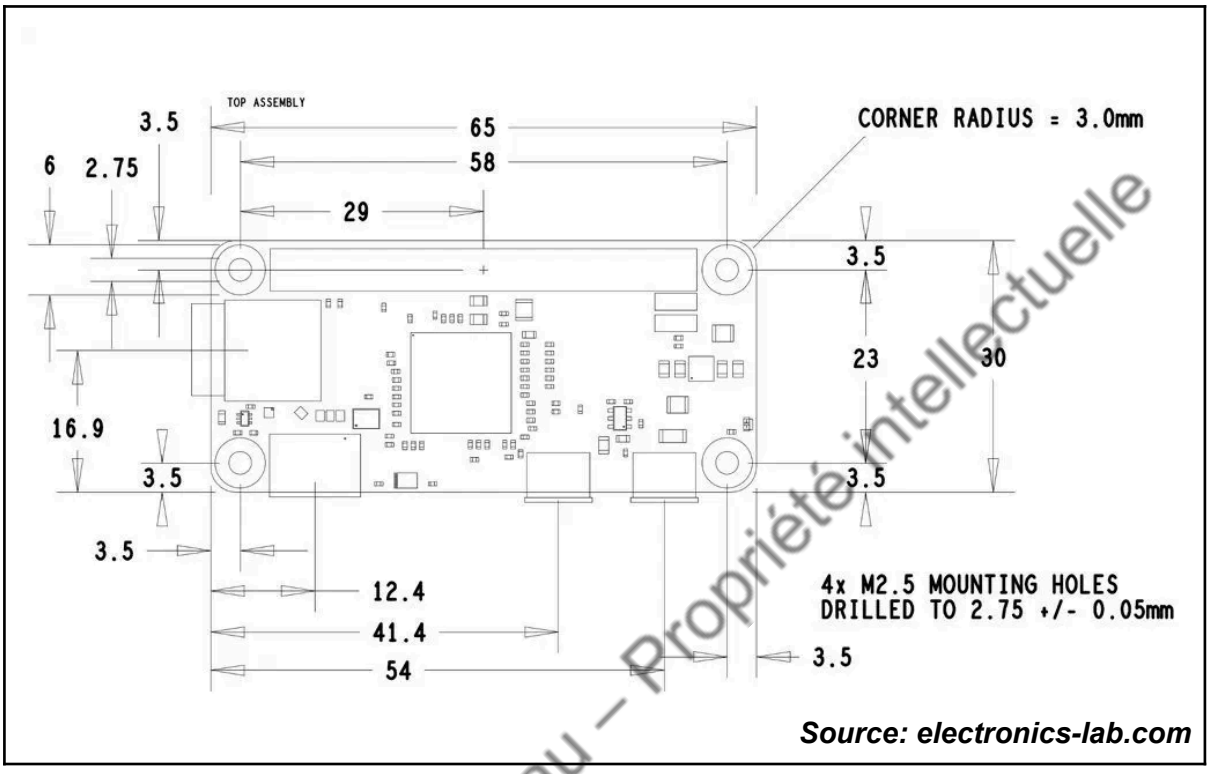
```
import shutil

def detect_camera_command():
    """
    Teste successivement les commandes caméra disponibles
    et retourne la première qui fonctionne
    """
    commands = ['rpicam-vid', 'libcamera-vid', 'raspivid']

    for cmd in commands:
        if shutil.which(cmd):  # Vérifie si la commande existe
            add_log(f" Commande caméra détectée : {cmd}")
            return cmd

    add_log(" Aucune commande caméra trouvée !")
    return None

# Détection au démarrage du serveur
CAMERA_CMD = detect_camera_command()
```

Envoi automatique sur Telegram :

```
import telegram

def send_to_telegram(video_path):
    """Envoie la vidéo sur le canal Telegram privé"""
    try:
        bot = telegram.Bot(token=BOT_TOKEN)

        with open(video_path, 'rb') as video:
            caption = f"📹 Enregistrement du {datetime.now().strftime('%d/%m/%Y à
%H:%M')}"

            bot.send_video(
                chat_id=CHAT_ID,
                video=video,
                caption=caption,
                supports_streaming=True
            )

        add_log(f" Vidéo envoyée sur Telegram")

    except Exception as e:
        add_log(f" Erreur envoi Telegram : {e}")
```

3.2 Conception 3D

La conception du boîtier sur SolidWorks a nécessité une approche très méthodique et plusieurs itérations avant d'obtenir un résultat satisfaisant. J'ai commencé par prendre des mesures précises de tous les composants.



Source: electronics-lab.com

Version 1 - Prototype Dimensionnel

La première version du boîtier avait pour objectif principal de valider les dimensions générales. J'ai conçu un modèle simplifié, une sorte de "boîte test", qui me permettait de vérifier que le Raspberry Pi Zero W rentrait correctement et que les dimensions externes restaient raisonnables pour une fixation sur des lunettes. Cette version ne comportait aucun système de fixation, juste un emplacement basique pour la carte.

L'impression de ce premier prototype a permis de confirmer que les dimensions de 45×28×18 millimètres étaient appropriées. Cependant, cette version était trop rudimentaire pour être fonctionnelle : rien ne maintenait le Raspberry Pi en place et la caméra n'était pas du tout prise en compte.

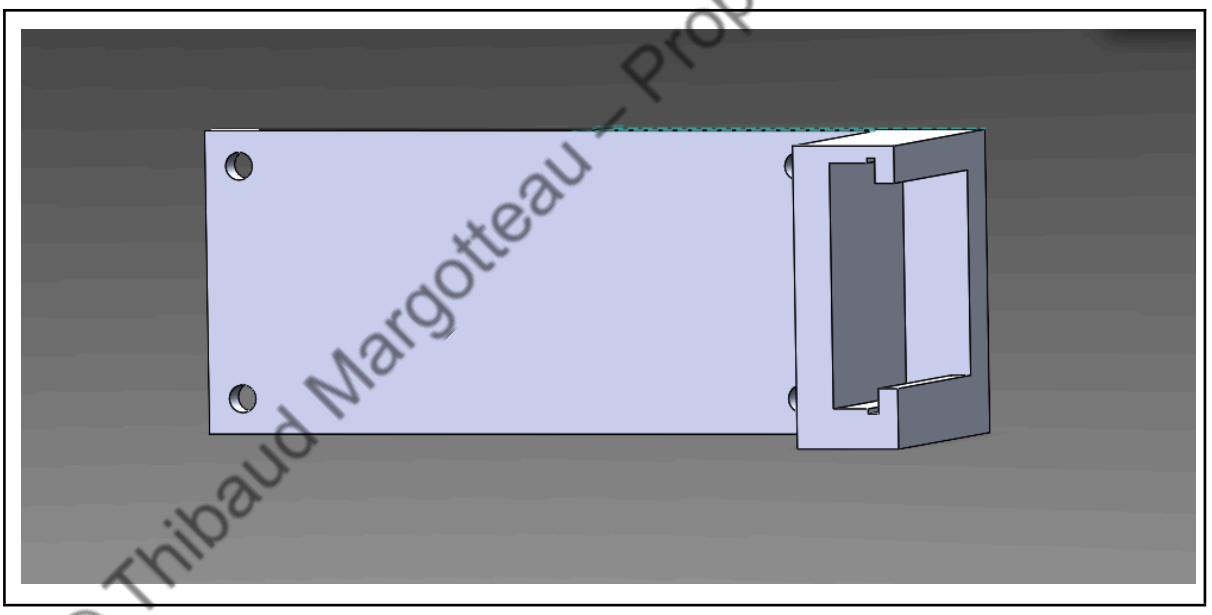
Version 2 - Ajout des Fixations

J'ai développé un modèle beaucoup plus abouti pour la version 2. J'ai ajouté quatre trous de 4 millimètres de profondeur pour accueillir des vis M2×8mm, positionnés exactement selon les trous de fixation du Raspberry Pi.

La nouveauté majeure de cette version était l'encoche dédiée au module caméra. J'ai créé un logement rectangulaire dans la partie supérieure du boîtier, destiné à recevoir le module et à le maintenir en place. Malheureusement, lors de l'assemblage réel après impression, j'ai découvert un problème, l'encoche était trop petite. Le module caméra, avec ses 25×24 millimètres de circuit imprimé plus l'épaisseur du capteur lui-même, ne rentrait tout simplement pas dans l'espace prévu. J'avais sous-estimé l'encombrement réel du module, notamment la hauteur du capteur qui dépasse.

Version 3 - Solution Finale

J'ai redessiné entièrement le logement caméra en ajoutant 2 millimètres de largeur et 3 millimètres de profondeur par rapport à la version 2. Cette fois si, l'assemblage s'est déroulé parfaitement. Le module caméra glisse dans son encoche sans résistance excessive, et une fois en place, il reste parfaitement stable sans aucun jeu.



3.3 Problèmes Techniques Rencontrés

Tout au long du développement, j'ai été confronté à de nombreux obstacles techniques qui ont nécessité des solutions parfois créatives. Voici une synthèse des problèmes majeurs et de leurs résolutions.

Problème	Symptôme	Cause Identifiée	Solution Appliquée	Résultat
Caméra non détectée	vcgencmd get_camera → 0	Drivers libcamera manquants	sudo apt install libcamera-apps rpicam-apps	Résolu
Vidéos saccadées 1080p	15-20 fps au lieu de 30	RAM saturée à 95%	Réduction à 720p + bitrate 2 Mbps	Fluide
Fichiers MP4 corrompus	VLC refuse de lire	_____	_____	Non résolu
Surchauffe CPU	78°C après 15 min	Pas de ventilation active	Fentes + limitation durée 10 min	Acceptable
Transfert SCP lent	5 min pour 100 MB	Bande passante Wi-Fi	Migration Telegram	20s

1. Reconnaissance de la Caméra

Le problème initial le plus frustrant concernait la non-détection de la caméra. La commande de diagnostic standard renvoyait :

pi@raspberrypi:~ $ vcgencmd get_camera
supported=0 detected=0, libcamera interfaces=0

J'ai testé plusieurs solutions avant de trouver la bonne :

```
# Tentative 1 : Réactivation via raspi-config
sudo raspi-config
# Interface Options > Legacy Camera > Enable
# Résultat : Aucun changement
```

```
# Tentative 2 : Modification /boot/config.txt
sudo nano /boot/config.txt
# Ajout : start_x=1, gpu_mem=128
# Résultat : Toujours pas détectée
```

```
# Solution finale : Installation des nouveaux drivers
sudo apt update
sudo apt install libcamera-apps rpicam-apps
sudo reboot
```

```
# Vérification post-installation
pi@raspberrypi:~ $ rpicam-hello --list-cameras
Available cameras
-----------------
0 : imx219 [3280x2464] (/base/soc/i2c0mux/i2c@1/imx219@10)
    Modes: 'SRGGB10_CSI2P' : 640x480 1640x1232 3280x2464
                             1920x1080 [30.00 fps]
# Caméra détectée
```


Le deuxième problème récurrent concernait la qualité des vidéos enregistrées. Mes premiers essais en résolution 1080p produisaient des vidéos fortement saccadées, avec un effet d'accélération désagréable qui rendait le résultat inexploitable. En analysant l'utilisation des ressources système pendant l'enregistrement, j'ai constaté que la RAM était saturée et que le processeur était constamment à 100% de charge. Le Raspberry Pi Zero W n'avait tout simplement pas la puissance nécessaire pour encoder de la vidéo en haute définition en temps réel.

La solution a consisté à réduire la résolution à 720p, ce qui représente environ 40% de pixels en moins à traiter. J'ai également limité le bitrate à 2 mégabits par seconde pour alléger la charge d'encodage. Ces ajustements ont considérablement amélioré la fluidité des vidéos, même si certaines légères saccades persistent lors de mouvements très rapides. Pour obtenir une qualité parfaite en 1080p, il faudrait utiliser un Raspberry Pi 4 ou au minimum un Raspberry Pi Zero 2 W, qui dispose d'un processeur quad-core beaucoup plus performant.

La récupération des vidéos a également posé des difficultés inattendues. Ma première approche consistait à transférer les fichiers via SCP, le protocole de transfert sécurisé de SSH. Cependant, cette méthode s'est révélée lente et peu pratique, nécessitant de taper des commandes complexes à chaque transfert. J'ai ensuite tenté d'accéder directement aux fichiers en retirant la carte microSD du Raspberry Pi pour la lire sur mon ordinateur. Malheureusement, certaines vidéos apparaissaient comme corrompues ou totalement absentes, probablement à cause d'une synchronisation incomplète du système de fichiers au moment du retrait de la carte.

Un autre problème concernait la conversion des vidéos en format MP4. Le programme ffmpeg, que j'utilise pour cette conversion, produisait parfois des fichiers défectueux qui refusaient de s'ouvrir dans VLC. Malheureusement, je n’ai pas réussi à régler ce problème.

C'est cette frustration qui m'a poussé à utiliser Telegram. En automatisant complètement le transfert via une API web, j'ai éliminé tous les problèmes de manipulation manuelle. Les vidéos arrivent désormais directement sur mon smartphone quelques secondes après la fin de l'enregistrement, et je peux les consulter ou les télécharger. L'utilisation de Telegram s'est révélée infiniment plus pratique et plus fiable.

3.4 Migration vers Telegram

L'intégration de Telegram dans le système a représenté une évolution majeure du projet. Initialement, je n'avais pas prévu cette fonctionnalité, pensant que le transfert manuel des fichiers suffirait. Cependant, face aux difficultés rencontrées avec les méthodes traditionnelles, j'ai réalisé qu'une solution de cloud messaging était non seulement plus pratique, mais aussi plus moderne et plus en phase avec l'usage d'un objet connecté portable.

La création du bot Telegram s'est faite en quelques minutes via l'interface BotFather. Ce service officiel de Telegram permet de créer des bots personnalisés en répondant à quelques questions simples. J'ai obtenu un token d'authentification unique, une longue chaîne de caractères qui identifie mon bot de manière sécurisée. J'ai ensuite créé un canal privé, accessible uniquement par moi, et j'ai récupéré son identifiant numérique appelé chat_id. Ce canal sert de destination pour toutes les vidéos enregistrées par les lunettes.

L'implémentation dans le code Python a été relativement simple grâce à la bibliothèque python-telegram-bot. Cette bibliothèque fournit une interface pour interagir avec l'API Telegram sans avoir à gérer manuellement les requêtes HTTP. Une fois la vidéo convertie en MP4, une simple fonction d'envoi suffit : elle ouvre le fichier vidéo, l'envoie au bot avec une légende incluant la date et l'heure de l'enregistrement, puis ferme proprement le fichier.

L'un des avantages inattendus de cette approche est que Telegram applique automatiquement une compression intelligente aux vidéos. Si la vidéo dépasse une certaine taille, Telegram la compresse légèrement pour accélérer le transfert, tout en préservant une qualité visuelle acceptable. Cette compression est généralement imperceptible à l'œil nu et permet de réduire significativement les temps de transfert, surtout lorsque la connexion Internet est lente.

Un autre bénéfice majeur est l'archivage automatique. Toutes les vidéos envoyées restent disponibles dans l'historique du canal Telegram, créant ainsi une sorte de bibliothèque vidéo consultable à tout moment. Je peux retrouver une vidéo spécifique en quelques secondes, la télécharger à nouveau si nécessaire, ou la partager directement avec d'autres personnes en leur donnant accès au canal. Cette fonctionnalité transforme le système en un véritable outil de documentation visuelle.

La migration vers Telegram a également ouvert des perspectives intéressantes pour l'évolution future du projet. Il serait par exemple possible d'implémenter des commandes Telegram pour contrôler à distance le démarrage et l'arrêt de l'enregistrement, rendant l'interface web optionnelle.

4. Résultats et Performances

4.1 Métriques Techniques

Après plusieurs semaines de développement et d'optimisation, j'ai pu mesurer précisément les performances du système dans différentes conditions d'utilisation.

Tableau Comparatif Objectifs VS Résultats

Critère	Objectif Initial	Résultat Mesuré	Écart	Évaluation
Résolution	1080p30	720p30	-40% pixels	Compromis nécessaire
Fluidité	30 fps stable	28-32 fps	-2%	Acceptable
Température max	< 70°C	>78°C	+11%	Limite de l'acceptable

4.2 Analyse Qualitative

Au-delà des chiffres bruts, l'expérience d'utilisation du système mérite une analyse plus approfondie. L'interface web, bien que minimaliste, remplit parfaitement son rôle. Les deux boutons sont clairement identifiés et réactifs, les logs s'actualisent en temps réel et fournissent des informations utiles sur l'état du système. L'absence de fonctionnalités superflues rend l'interface intuitive et accessible même pour des utilisateurs peu familiers avec la technologie. En quelques minutes d'utilisation, n'importe qui peut comprendre le fonctionnement et commencer à enregistrer des vidéos.

La qualité visuelle des vidéos produites est très satisfaisante en conditions d'éclairage normal. Les couleurs sont fidèles, l'exposition automatique fonctionne correctement et le niveau de détail permet de distinguer clairement les visages et les objets. En revanche, dans des environnements sombres, la qualité se dégrade fortement.

La stabilité mécanique du boîtier est un point fort du système. Une fois fixé sur la branche des lunettes, l'ensemble ne bouge pas et la caméra reste parfaitement alignée. Les vis de fixation maintiennent fermement le Raspberry Pi et aucun jeu n'est perceptible même après plusieurs heures d'utilisation. L'esthétique générale, bien que clairement artisanale, reste discrète et ne choque pas visuellement.

L'intégration avec Telegram s'est révélée être l'une des meilleures décisions du projet. La fiabilité du transfert est parfaite: sur plus de vingt-cinq enregistrements de test, aucun échec de transfert n'a été constaté. Les vidéos arrivent systématiquement sur le canal quelques secondes après la conversion, et la qualité après compression Telegram reste excellente. La possibilité de consulter immédiatement les vidéos sur smartphone ou de les télécharger transforme réellement l'expérience utilisateur et rapproche le prototype d'un produit fini.

Cependant, quelques défauts persistent. Les vidéos présentent parfois de légers micro-saccades. Un effet subtil mais perceptible qui donne l'impression d'un léger timelapse. Ce problème provient de la difficulté du processeur à maintenir une fréquence d'images parfaitement constante.

4.3 Comparaison avec Ray-Ban Meta

Il est intéressant de comparer ce prototype avec les Ray-Ban Meta, le produit commercial qui a inspiré ce projet. Cette comparaison permet de situer objectivement les performances et les compromis de mon système.

Tableau Comparatif Détaillé

Caractéristique	Ray-Ban Meta	Prototype DIY	Avantage
Résolution vidéo	1080p60	720p30	Ray-Ban
Audio	Haut-parleurs et micro intégrés	Aucun	Ray-Ban
Stockage	32 GB intégré	Carte micro SD (modulable)	DIY
Prix	379€	25€ (sans batterie)	DIY
Réparabilité	Difficile	Simple	DIY
Personnalisation	Écosystème fermé	Code Open Source	DIY
Vie privée	Envoi données Meta	Données locales	DIY
Design	Professionnelle	Artisanal	Ray-Ban

Performances Vidéo

Les Ray-Ban Meta offrent clairement une qualité supérieure avec leur capteur 12 mégapixels et leur résolution 1080p à 60 images par seconde. Mon prototype plafonne à 720p30 en raison des limitations du Raspberry Pi Zero W. Cependant, pour un usage amateur ou de capture de moments, la qualité 720p reste largement suffisante et proche de ce que proposaient les smartphones il y a quelques années.

Ergonomie et Poids

Sur le papier, mon module de 42 grammes semble plus léger que les 51 grammes des Ray-Ban. Mais cette comparaison est trompeuse car elle ne prend pas en compte la batterie externe obligatoire qui ajoute du poids au système. Les Ray-Ban Meta intègrent leur batterie dans les branches, offrant ainsi un design beaucoup plus équilibré et confortable pour un port prolongé.

Rapport Qualité-Prix

Le prix de 379€ des Ray-Ban Meta contre 25€ (sans batterie) pour mon prototype représente un écart considérable. Pour le prix d'une paire de lunettes Meta, je pourrais construire quinze prototypes complets ou investir dans du matériel plus performant (Raspberry Pi 4, caméra officielle de meilleure qualité). Cette différence de coût démontre qu'il est possible de reproduire les fonctionnalités essentielles d'un produit high-tech avec du matériel grand public.

Réparabilité et Durabilité

Mon prototype est plus facilement réparable que les Ray-Ban Meta. Chaque composant est remplaçable individuellement : si le Raspberry Pi grille, il coûte 15€ à remplacer. Si la caméra casse, 8€ suffisent. Les Ray-Ban, en revanche, sont conçues comme un objet monobloc dont l'ouverture invalide la garantie et dont les pièces détachées ne sont pas commercialisées.

Liberté Logicielle et Vie Privée

Point crucial pour certains utilisateurs : les Ray-Ban Meta nécessitent l'application Meta View et transmettent obligatoirement des données de télémétrie aux serveurs de Meta. Mon système fonctionne entièrement en local, ne collecte aucune donnée personnelle, et les vidéos peuvent être conservées exclusivement sur la carte SD ou envoyées vers le service de son choix (Telegram,ou autres).

Conclusion de la Comparaison

Les Ray-Ban Meta restent objectivement supérieures sur tous les aspects techniques : qualité vidéo, ergonomie, intégration, finition. Elles représentent un produit fini, testé et fiable. Mon prototype, en revanche, excelle dans 2 domaines spécifiques : le coût environ 93% moins cher et le contrôle de ses données avec un code open source modifiable.

5. Conclusion et Perspectives

5.1 Bilan du Projet

Ce projet de lunettes connectées a représenté bien plus qu'un simple exercice technique. Il m'a permis d'explorer en profondeur l'ensemble des aspects de la conception d'un objet connecté moderne, depuis la sélection des composants jusqu'à l'écriture du code en passant par la modélisation 3D. Chaque étape a apporté son lot de défis et d'apprentissages, me poussant constamment à chercher des solutions aux problèmes rencontrés.

Le résultat final, bien qu'imparfait, démontre qu'il est possible de créer un système fonctionnel et utilisable avec des moyens limités. Les lunettes peuvent effectivement filmer, enregistrer, convertir et transférer des vidéos de manière automatisée, exactement comme prévu dans les objectifs initiaux. L'interface de contrôle fonctionne parfaitement et rend le

système accessible même pour des utilisateurs non techniques. L'intégration avec Telegram a transformé un prototype de laboratoire en un véritable outil portable et pratique.

Sur le plan personnel, ce projet m'a permis de développer des compétences dans des domaines variés. J'ai approfondi ma maîtrise de Python et découvert Flask, un framework que je ne connaissais pas auparavant. J'ai appris à manipuler des API web tierces comme celle de Telegram. J'ai acquis de l'expérience en modélisation 3D avec le fameux logiciel de Dassault Système, SolidWorks. J'ai également développé une meilleure compréhension du fonctionnement des systèmes Linux et de l'administration système en ligne de commande.

Au-delà des compétences techniques, ce projet m'a enseigné l'importance de la persévérance. Chaque problème rencontré semblait au départ insurmontable, mais une recherche méthodique et des tests patients ont presque toujours fini par apporter une solution. J'ai appris à accepter que les prototypes ne sont jamais parfaits du premier coup.

5.2 Difficultés Majeures

Parmi toutes les difficultés rencontrées, certaines se sont révélées particulièrement marquantes. La gestion des incompatibilités logicielles a été source de nombreuses heures de frustration. Le fait que différentes versions de Raspberry Pi OS utilisent des bibliothèques caméra différentes, que certaines commandes deviennent obsolètes sans prévenir, et que la documentation officielle ne soit pas toujours à jour, tout cela a créé une frustration constante. J'ai appris qu'il est essentiel de tester systématiquement plusieurs approches et de ne jamais faire confiance aveuglément à une seule source de documentation.

Les limitations matérielles du Raspberry Pi Zero W ont également représenté un défi constant. Devoir sans cesse optimiser le code, réduire la résolution, limiter les processus simultanés, tout cela pour essayer de tirer le maximum d'un processeur vieux de plusieurs années et d'une mémoire vive largement insuffisante selon les standards actuels. Cette contrainte m'a forcé à réfléchir autrement, à privilégier l'efficacité, et à accepter des compromis qui n'auraient pas été nécessaires avec du matériel plus puissant.

La précision mécanique requise pour la conception du boîtier a été une découverte importante. Sur le logiciel 3D, tout s'emboîte à la perfection. Dans la réalité, avec une imprimante “amateur”, les tolérances sont beaucoup plus larges et les ajustements manuels sont inévitables. J'ai appris qu'il faut toujours prévoir plus de jeu que ce qui semble nécessaire théoriquement, et qu'une pièce imprimée nécessite presque toujours un peu de post-traitement pour être réellement fonctionnelle.

Enfin, la gestion de la corruption des fichiers vidéo a été l'un des problèmes les plus difficiles à diagnostiquer. Les symptômes étaient aléatoires, certaines vidéos fonctionnaient parfaitement tandis que d'autres étaient illisibles sans raison apparente.

Conditions d'Utilisation et Propriété Intellectuelle

Droits d'Auteur et Licence

Le présent projet, incluant l'ensemble de la documentation, du code source Python, des modèles 3D SolidWorks et STL, ainsi que tous les schémas et illustrations associés, constitue une œuvre originale protégée par le droit d'auteur.

Auteur : Thibaud
Date de création : Novembre 2025
Propriété intellectuelle : Tous droits réservés

Utilisation Autorisée

Ce projet est mis à disposition du public dans un cadre exclusivement éducatif, pédagogique et non commercial. Les utilisations suivantes sont autorisées :

- **Reproduction à des fins d'apprentissage personnel** : Vous pouvez construire votre propre version du prototype pour votre usage personnel.
- **Utilisation académique** : Les établissements d'enseignement peuvent utiliser ce projet comme support de cours, de travaux pratiques ou de projet étudiant.
- **Partage éducatif** : Vous pouvez partager ce document avec d'autres personnes à des fins d'apprentissage, à condition de mentionner explicitement le nom de l'auteur (Thibaud) et de ne pas modifier le contenu sans autorisation.
- **Amélioration collaborative** : Les suggestions d'amélioration, corrections d'erreurs et retours d'expérience sont encouragées et peuvent être partagées avec l'auteur.

Utilisations Interdites

Les utilisations suivantes sont formellement interdites sans autorisation écrite préalable de l'auteur :

- **Vente ou commercialisation** : Il est strictement interdit de vendre des exemplaires de ce prototype, des kits de montage, ou tout produit dérivé basé sur cette conception.
- **Usage commercial** : Toute utilisation dans un cadre professionnel lucratif, y compris la fabrication de prototypes pour des clients, nécessite une autorisation explicite.
- **Modification et redistribution non créditée** : Toute modification du projet (code, plans 3D, documentation) doit mentionner clairement l'œuvre originale et son auteur.
- **Revendication de paternité** : Il est interdit de présenter ce travail comme le vôtre ou d'en revendiquer la conception originale.

Mentions Obligatoires en Cas de Partage

Si vous partagez ce projet, que ce soit en ligne ou physiquement, vous devez obligatoirement inclure la mention suivante :

> *"Projet de lunettes connectées conçu par Thibaud Margotteau (Novembre 2025). Reproduction autorisée à des fins éducatives uniquement. Utilisation commerciale interdite.*

Clause de Non-Responsabilité

L'auteur décline toute responsabilité quant aux dommages matériels, corporels ou financiers qui pourraient résulter de la reproduction, modification ou utilisation de ce projet. La construction et l'utilisation de ce dispositif se font sous votre propre responsabilité. Veillez à respecter les réglementations locales concernant l'enregistrement vidéo et la protection de la vie privée.