Étape 2 : Hardware x Software

La deuxième étape dans la fabrication du système consiste à mettre en place le système de capture. Dans cette section, nous allons créer un environnement virtuel, installer les bibliothèques nécessaires pour le travail,et d'autres étapes additionnelles pour la configuration des composants.

Pour se cultiver

pip est utilisé pour télécharger et installer des paquets directement depuis PyPI (Python Package Index), hébergé par la Python Software Foundation. Il s'agit d'un gestionnaire de paquets spécialisé exclusivement pour les paquets Python.

apt-get, en revanche, est utilisé pour télécharger et installer des paquets depuis les dépôts Ubuntu hébergés par Canonical.

Voici quelques différences clés entre l'installation des paquets Python avec apt-get et pip :

1. Disponibilité des Paquets: Canonical fournit des paquets pour un ensemble sélectionné de modules Python. En revanche, PyPI héberge une gamme beaucoup plus large de modules Python. Par conséquent, de nombreux modules Python ne peuvent pas être installés via apt-get mais sont disponibles sur PyPI.

2. Contrôle des Versions : Canonical héberge généralement une seule version de chaque paquet (généralement la plus récente ou celle sortie récemment). Avec apt-get, vous ne pouvez pas choisir quelle version d'un paquet Python installer. En revanche, pip vous permet d'installer n'importe quelle version d'un paquet précédemment téléchargé sur PyPI. Cette flexibilité est essentielle pour gérer les conflits de dépendances.

3. Portée de l'Installation : apt-get installe les modules Python de manière globale sur le système, ce qui signifie qu'ils sont disponibles globalement. Cela peut entraîner des conflits si différents projets nécessitent différentes versions du même paquet. pip résout ce problème en installant les paquets dans l'environnement virtuel du projet (virtualenv). Cette isolation garantit que les dépendances sont gérées indépendamment pour chaque projet.

4. Conventions de Nom de Paquet : Canonical nomme généralement les paquets Python 2 comme `python-<nom_du_paquet>` et les paquets Python 3 comme `python3-<nom_du_paquet>`. En revanche, pip simplifie l'installation des paquets en utilisant `<nom_du_paquet>` pour les paquets Python 2 et Python 3.

Choix entre apt-get et pip:

apt-get et pip sont tous deux des gestionnaires de paquets matures qui gèrent la résolution des dépendances lors de l'installation. Vous pouvez utiliser l'un ou l'autre en fonction de vos préférences. Cependant, envisagez d'utiliser **pip** dans les scénarios suivants :
- Lorsque vous avez besoin d'installer une version spécifique d'un paquet Python.
- Lorsque vous souhaitez installer des paquets dans l'environnement virtuel d'un projet.
- Lorsque le paquet dont vous avez besoin n'est disponible que sur PyPI.

Si vous êtes à l'aise avec l'installation de paquets globalement et qu'il n'y a pas de conflits de versions, apt-get ou pip conviendront pour l'installation générale de paquets Python.

C'est quoi un Environment Virtuelle :https://www.geeksforgeeks.org/python-virtual-environment/

Installation d'un Environment Virtuelle

On utilisera **pip** depuis un environnement virtuel pour installer les bibliothèques.

Une fois connecté à votre Raspberry Pi via SSH:

1 : Naviguer vers votre répertoire de projet

Accédez au répertoire où vous souhaitez créer votre environnement virtuel, ex :

```
/home/pi/Desktop/scripts
```

2 : Créer l'environnement virtuel

Utilisez la commande 'python3 -m venv venv' pour créer un environnement virtuel. Remplacez 'venv' par le nom que vous souhaitez donner à votre environnement virtuel.

```
python3 -m venv venv
```

3 : Activer l'environnement virtuel

```
source venv/bin/activate
```

4 : Désactiver l'environnement virtuel

Vous ne pourriez pas lancer le code si environnement virtuelle est désactiver

```
deactivate
```

5 : Installer les modules requis

Si `pip` n'est pas déjà installé sur votre système, vous pouvez l'installer en utilisant la commande adaptée à votre distribution Linux. Par exemple

```
sudo apt-get install python3-pip
```

puis

```
pip install adafruit-circuitpython-rgb-display
```

```
pip install adafruit-blinka busio digitalio
```

```
pip install opencv-python
```

```
pip install pillow
```

Vérifier les bibliothèques installées

```
(venv) pi@raspberrypi:~/Desktop/scripts $ pip list
Package                            Version
---------------------------------------- ---------
Adafruit-Blinka                      8.45.0
adafruit-circuitpython-busdevice     5.2.9
adafruit-circuitpython-connectionmanager 3.1.1
adafruit-circuitpython-requests      4.1.1
adafruit-circuitpython-rgb-display   3.12.4
adafruit-circuitpython-typing        1.10.3
Adafruit-PlatformDetect              3.71.0
Adafruit-PureIO                      1.1.11
binho-host-adapter                   0.1.6
numpy                                2.0.0
opencv-python                        4.10.0.84
pillow                               10.3.0
pip                                  23.0.1
```

```
pyftdi                  0.55.4
pyserial                3.5
pyusb                   1.2.1
RPi.GPIO                0.7.1
rpi-ws281x              5.0.0
setuptools              66.1.1
sysv-ipc                1.1.0
typing_extensions       4.12.2
```

Le code réel

Vous pouvez trouver une documentation de chaque classe et les références à la fin de cette page.

Vous pouvez installer les fichiers depuis le Git '/scripts/ras_pi_zero2W/os5' ou bien depuis la section Fichier sources , sauvegarder tous les fichiers dans un répertoire ex : os5

puis **rendez les fichiers exécutable** avec par exemple :

pour tous le répertoire :

```
chmod -R +x os5
```

ou bien chaque fichier individuellement :

```
chmod +x button.py
```

Explications des fichiers :

```
(venv) pi@raspberrypi:~/Desktop/scripts $ ls
camera_feed_test.py  camera_properties.txt  os1  os2  os3  os4  os5  venv
```

camera_feed_test.py est utilisé pour tester les différentes capacités de votre caméra. Nous allons apprendre ensuite comment les trouver et les stocker dans un fichier, par exemple : camera_properties.txt. Les fichiers os1...5 sont les différentes itérations du système. Nous allons nous concentrer uniquement sur os5.Pour plus d'informations à propos des autres systèmes d'exploitation, veuillez consulter le rapport complet.

```
(venv) pi@raspberrypi:~/Desktop/scripts/os5 $ ls
button.py  camera.py  draw.py  main.py  menu.py  __pycache__  rotary_encoder.py  screen.py
```

button.py :

Ce fichier contient la classe responsable du contrôle du bouton. Dans cette version, on peut détecter des clics simples et des clics longs de 6 secondes, accédez au fichier pour modifier cette durée. Plus d'infos dans le rapport.

> Veuillez n'utiliser que le pin GPIO3 pour le bouton, car il servira à allumer et éteindre le système avec le même bouton.Assurez vous que votre interface I2C est éteinte (Elle est par défaut)

rotary_encoder.py :

Ce fichier contient la classe responsable du contrôle de l'encodeur rotatif.Plus d'infos dans le rapport. Pas de préférence sur les GPIO pins.

> Il est toujours recommandé d'ajouter des résistances lorsque vous utilisez des capteurs pour limiter le courant en entrée dans les broches GPIO, que ce soit des résistances de pull-up ou de pull-down.

Plus d'informations à propos du bouton et de l'encodeur rotatif plus d'informations dans la section 4.3 du rapport

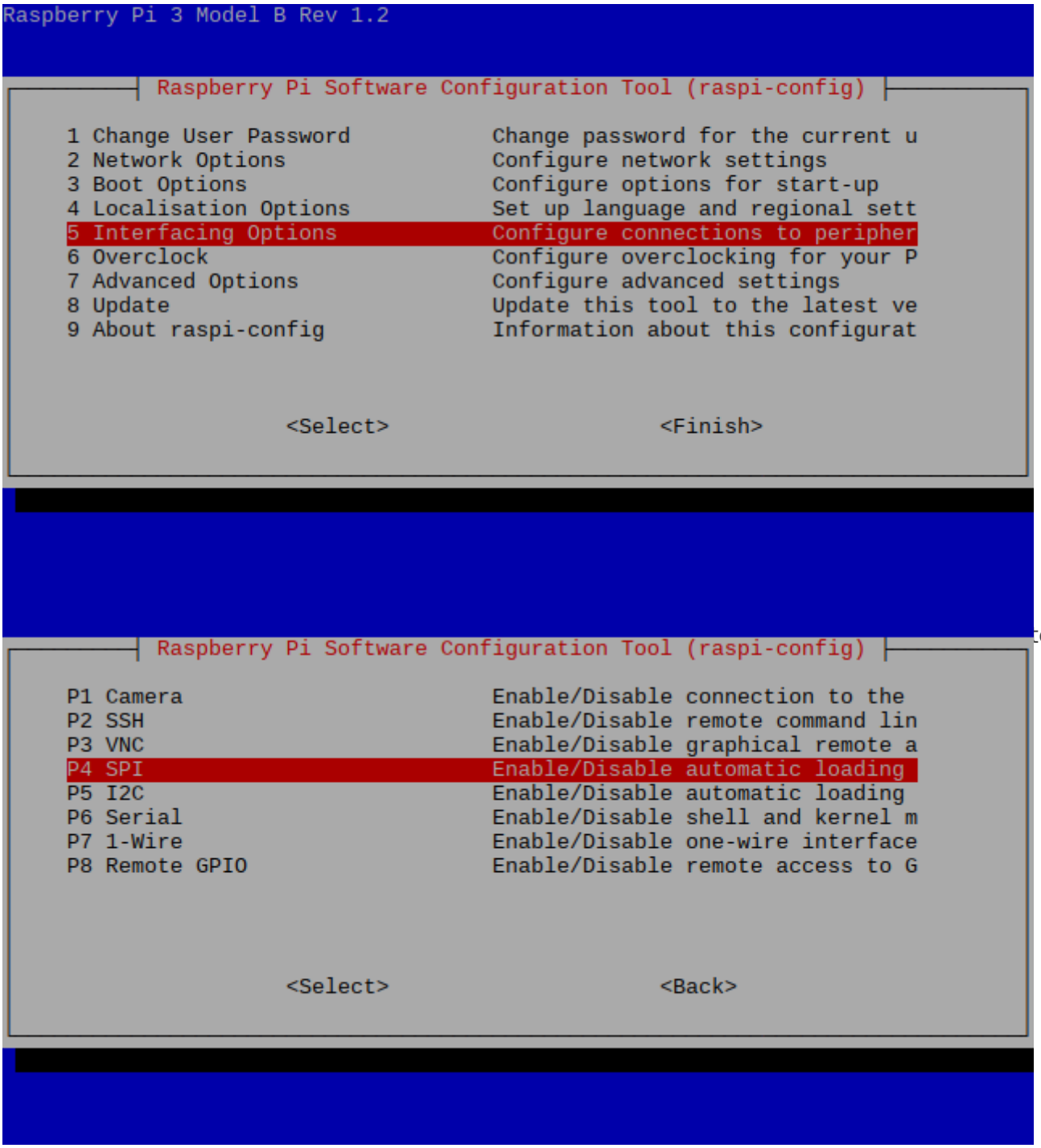
screen.py :

Ce fichier contient la classe responsable d'initialiser l'écran. On utilise l'interface SPI0 pour communiquer avec l'écran, c'est pourquoi nous avons besoin de l'activer au préalable. Pour changer les GPIO utilisés, vous devez accéder directement à la classe et les modifier. Cette classe ne prend pas les pins comme argument, contrairement aux classes `button` et `rotary_encoder`.

Comment activer l'interface SPI

Depuis un terminal tapez cette commande

```
sudo raspi-config
```



tème

camera.py :

Ce fichier contient la classe Camera, responsable de la capture vidéo et de son enregistrement.

comments trouvez les caractéristiques de votre caméra

v4l2 documentation : https://www.mankier.com/1/v4l2-ctl

Listez tous les périphériques vidéo :

'v4l2-ctl --list-devices'

```
(venv) pi@raspberrypi:~/Desktop/scripts/os5 $ v4l2-ctl --list-devices
bcm2835-codec-decode (platform:bcm2835-codec):
        /dev/video10
        /dev/video11
        /dev/video12
        /dev/video18
        /dev/video31
        /dev/media2

bcm2835-isp (platform:bcm2835-isp):
        /dev/video13
        /dev/video14
        /dev/video15
        /dev/video16
        /dev/video20
        /dev/video21
        /dev/video22
        /dev/video23
        /dev/media0
        /dev/media1

HD Webcam C525 (usb-3f980000.usb-1.2):
        /dev/video0
        /dev/video1
        /dev/media3
```

Détectez votre appareil, puis :

Lister les formats vidéo supportés et les résolutions d'un périphérique vidéo spécifique :

'v4l2-ctl --list-formats-ext --device path/to/video_device'

```
(venv) pi@raspberrypi:~/Desktop/scripts/os5 $ v4l2-ctl --list-formats-ext --device /dev/video0
ioctl: VIDIOC_ENUM_FMT
        Type: Video Capture

        [0]: 'YUYV' (YUYV 4:2:2)
```

```
        Size: Discrete 640x480
            Interval: Discrete 0.033s (30.000 fps)
            Interval: Discrete 0.042s (24.000 fps)
            Interval: Discrete 0.050s (20.000 fps)
            Interval: Discrete 0.067s (15.000 fps)
            Interval: Discrete 0.100s (10.000 fps)
```

Maintenant copier toutes ces informations dans un fichier .txt et tester les avec **camera_feed_test.py**

```
def main():
    # Initialize the camera (0 is the default camera)
    cap = cv2.VideoCapture(0)
    #set the video dimensions(resolution)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 800)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 448)

    # Set the frame rate
    #cap.set(cv2.CAP_PROP_FPS, 24)

    if not cap.isOpened():
        print("Error: Could not open video capture.")
        return

    w=cap.get(3)
    h=cap.get(4)
    print('width=',w)
    print('height=',h)
    print('capturing fps = ',cap.get(5))
```

Quoi choisir et comment : Testez tous les formats, commencez par changer `cv2.CAP_PROP_FRAME_WIDTH` et `cv2.CAP_PROP_FRAME_HEIGHT` dans `camera_feed_test.py`. Ensuite, vérifiez à combien de FPS vous enregistrez. Notez tout cela dans une feuille quelque part, puis choisissez parmi toutes les options testées les plus pertinentes pour vous. Par exemple, nous avons choisi 5 options :

- Résolution de 800 x 448, enregistrement à 30 FPS

- Résolution de 800 x 600, enregistrement à 24 FPS

- Résolution de 800 x 448, enregistrement à 30 FPS

- Résolution de 1920 x 1080, enregistrement à 5 FPS

- Résolution de 1280 x 1080, enregistrement à 10 FPS

Enfin, ouvrez votre fichier `camera.py` et modifiez le chemin de sauvegarde de vos vidéos dans la méthode `set_rec_settings`. Par exemple, nous avons choisi : `/home/pi/Desktop/videos`.

```
def set_rec_settings(self,wfps,dimensions):
        self.video.set(cv.CAP_PROP_FRAME_WIDTH,dimensions[0])
        self.video.set(cv.CAP_PROP_FRAME_HEIGHT, dimensions[1])
        self.result = cv.VideoWriter(
                        os.path.join('/home/pi/Desktop/videos', f'{self.file}__{dimensions[0]}x{dimensions[1]}.avi'),
                        cv.VideoWriter_fourcc(*'XVID'),wfps, dimensions)
```

Pour plus d'informations à propos de la caméra, des choix de captures, ou de l'enregistrement de fichiers, veuillez consulter le rapport complet.

Plus d'informations à propos de la camera dans la section 4.2 du rapport

draw.py

Ce fichier contient la classe Draw, responsable de créer les graphiques pour le menu et ses options. Chaque instance de cette classe ne peut créer qu'une seule image et la stocker. Ainsi, pour créer et stocker plusieurs images (plutôt que de créer une image à chaque fois), nous devons créer autant d'instances de cette classe que d'images souhaitées.

menu.py

La classe Menu est responsable d'intégrer toutes les parties du système ensemble. Vous pouvez trouver plus d'informations à ce sujet dans le rapport complet.

Plus d'informations à propos du menu et de l'écran dans la section 4.4 du rapport

main.py

Ce fichier contient la boucle principale de ce système. Ce fichier sera exécuté pour faire fonctionner le système.

```
def main():
    # Initialize the screen, draw, molette, and button
    screen = Screen()
    draw0=Draw(user_id="user 123",height=screen.height,width=screen.width)
    draw1=Draw(user_id="user 123",height=screen.height,width=screen.width)
    draw2=Draw(user_id="user 123",height=screen.height,width=screen.width)
    draw3=Draw(user_id="user 123",height=screen.height,width=screen.width)

    camera=Camera()
    button=Button(pin=3)
    molette=RotaryEncoder(clk_pin=board.D27, dt_pin=board.D22)

    # Initialize the menu
    menu = Menu(screen, draw0,draw1,draw2,draw3, molette, button,camera)

    # Add menu items
    menu.add_item("1 min",wfps=10,dimensions=(800,448),char='@ W I D E  V G A ',mode= 'S L O W  M O T I O N')
    menu.add_item("5 min",wfps=22,dimensions=(800,600),char='@ S U P E R   V G A ',mode='N O R M A L')
    menu.add_item("30 min",wfps=28,dimensions=(800,448),char='@ W I D E  V G A ',mode='E C O  -  N O R M A L')

    menu.add_item("1 h",wfps=100,dimensions=(1920,1080),char='@ F U L L  H D' ,mode=' T I M E - L A P S E ')

    menu.add_item("3 h",wfps=600,dimensions=(1280,720),char='@ H D',mode='E C O  -  T I M E - L A P S E ')

    #menu.add_item("C A M  V I E W ")

    menu.menu_0()
    #menu.menu_1()
    menu.menu_3()
```

Modifications à apporter :

1. Changez les options de votre menu comme indiqué. Les dimensions dépendent des options de votre caméra que vous avez choisies précédemment.

2. Attention à la fréquence d'écriture (w-fps) : W-fps n'est pas la fréquence de capture de votre caméra mais la fréquence d'écriture des images dans votre fichier.

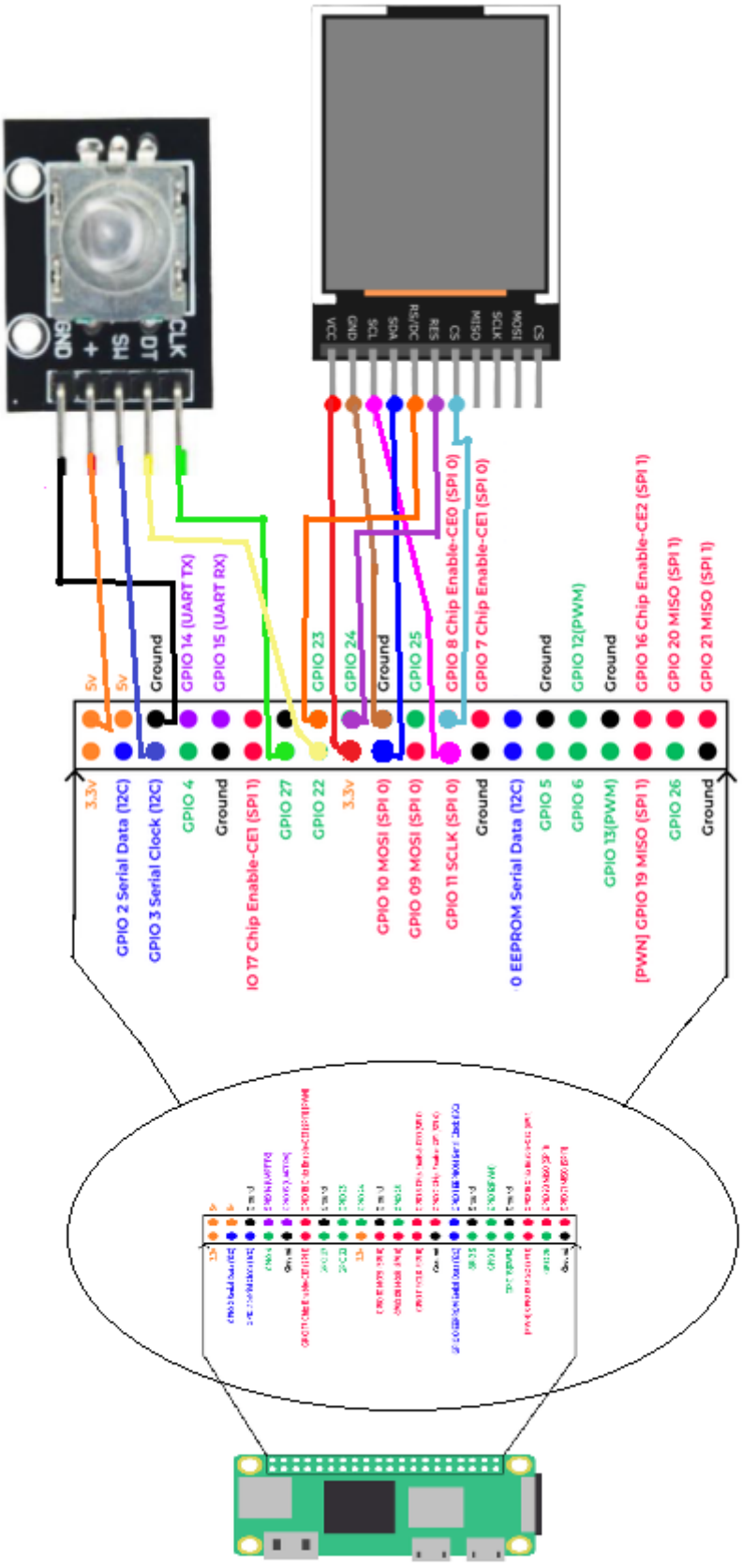
En résumé :
- Pour les modes Time-Lapse, vous voulez une fréquence d'écriture supérieure à la fréquence de capture. Par exemple, pour nous : 720x1280, fréquence de capture = 10 FPS, fréquence d'écriture = 600.
- Pour les modes normaux, recherchez une fréquence d'écriture approximativement égale à la fréquence de capture.
- Pour les modes slow-motion, cherchez une fréquence d'écriture inférieure à la fréquence de capture. Par exemple, chez nous : 30 FPS de capture et 10 FPS d'écriture.

3. Enfin, changez les broches de votre encodeur rotatif selon vos préférences.

Pour plus d'informations sur les fréquences de capture, consultez la section finale du rapport.

Assembler le Système

- **Rotary Encoder + Button:**
  - VCC = 5V
  - Ground = Ground
  - SW = GPIO3
  - CLK = GPIO27(votre choix)
  - DT = GPIO22(votre choix)
- **Screen:**
  - VCC = 3.3V
  - Ground = Ground
  - SCL = GPIO11 (SCLK) : Ligne d'horloge série pour la communication SPI. Ce pin fournit le signal d'horloge pour synchroniser la transmission des données.
  - SDA = GPIO10 (MOSI) : Master Out Slave In pour la communication SPI. Ce pin envoie des données du Pi vers l'écran.
  - DC = GPIO23(votre choix) : Pin d'entrée/sortie à usage général, utilisé ici pour la ligne de sélection de registre/commande de données (RS/DC) pour différencier les commandes et les données.
  - RES = GPIO24(votre choix) Pin d'entrée/sortie à usage général, utilisé pour la ligne de réinitialisation (RES) pour réinitialiser l'affichage.
  - CS = GPIO8 (CE0) : Activation de la puce 0, utilisée pour sélectionner le périphérique pour la communication sur le bus SPI.



Testez

Maintenant que tout est prêt, testez votre système. Les fichiers rotary_encoder.py et button.py contiennent tous des codes pour vous aider à vérifier que vos composants fonctionnent correctement. Vous pouvez les exécuter et les tester. Pour tester le système complet, lancez main.py.

(Assurez-vous que votre environnement virtuel est activé.)

```
python3 rotary_encoder.py
```

```
python3 button.py
```

```
python3 main.py
```

Documentation des Classes - Programmation

Classe : Bouton

Constructeur :

__init__(self, pin) : Initialise l'objet Bouton avec la broche GPIO spécifiée.

pin : Le numéro de la broche GPIO connectée au bouton.

Méthodes :

- **update(self) :** Met à jour l'état du bouton en lisant la broche GPIO et en vérifiant les pressions simples et longues.
- **get_State(self) :** Renvoie l'état actuel du bouton, indiquant s'il a été pressé ou longuement pressé.
- **cleanup(self) :** Nettoie les paramètres GPIO lorsque le programme se termine.

Référence:

RPi-GPIO documentation : https://pypi.org/project/RPi.GPIO/

Classe : RotaryEncoder

Constructeur :

__init__(self, clk_pin, dt_pin) : Initialise l'objet RotaryEncoder avec les broches d'horloge et de données spécifiées.

clk_pin : La broche GPIO utilisée pour le signal d'horloge.
dt_pin : La broche GPIO utilisée pour le signal de données.

Méthodes :

- **update(self) :** Met à jour la position du codeur rotatif en fonction de l'état des broches d'horloge et de données.

- **set_pos(self, pos) :** Définit le compteur de position sur la valeur spécifiée.

  où pos est a nouvelle valeur de position à définir.

Références:

Digitalio docuemntation :https://docs.circuitpython.org/en/latest/shared-bindings/digitalio/index.html

l-gpio documentation : https://rpi-lgpio.readthedocs.io/en/latest/

Getting started with rotary encoders : https://core-electronics.com.au/guides/getting-started-with-rotary-encoders-examples-with-raspberry-pi-pico/

Classe : Screen

Constructeur :

__init__(self) : Initialise l'objet Screen avec la communication SPI et la configuration d'affichage.

Méthodes :

disp_img(self, image) : Affiche l'image fournie sur l'écran.
image : L'objet image à afficher.

Références:

adafruit_rgb_display documentation :
https://docs.circuitpython.org/projects/rgb_display/en/latest/api.html

busio documentation : https://docs.circuitpython.org/en/latest/shared-bindings/busio/

Classe : Camera

Constructeur :

__init__(self) : Initialise l'objet Camera avec des attributs pour la capture vidéo, l'enregistrement vidéo et le nom de fichier.

Méthodes :

- **init_capture(self) :** Initialise la capture vidéo depuis la caméra par défaut.
- **rtrn_frames(self) :** Capture et renvoie une image de la vidéo en direct.
- **save_frames(self, frame) :** Enregistre une image avec un superposition de la date et l'heure actuelles dans le fichier vidéo.
  frame : L'image vidéo à enregistrer.
- **create_file(self) :** Génère un nom de fichier basé sur la date et l'heure actuelles.
- **set_rec_settings(self, wfps, dimensions) :** Définit les paramètres d'enregistrement, y compris la résolution et le taux d'images par seconde, et initialise l'enregistreur vidéo.
  wfps : Le nombre d'images par seconde souhaité pour la vidéo.
  dimensions : Un tuple spécifiant la largeur et la hauteur de la vidéo.
- **turn_off(self) :** Libère les ressources de capture vidéo et d'enregistrement vidéo.
  Arguments : Aucun

Références:

open cv VideoCapture documentation :
https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html#a57c0e81e83e60f36c83027dc2a188e80

open cv Video Flags :
https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html#gaeb8dd9c89c10a5c63c139bf7c4f5704d

open cv VideoWrite documentation :
https://docs.opencv.org/4.x/dd/d9e/classcv_1_1VideoWriter.html

Classe : Draw

Constructeur :

__init__(self, height, width) : Initialise l'objet Draw avec une hauteur et une largeur spécifiées, créant un canevas d'image pour dessiner.
height : Hauteur de la zone de dessin.
width : Largeur de la zone de dessin.

Méthodes :

- **draw_msg(self, message, position) :** Dessine un message à la position spécifiée sur l'image.
  message : Le message texte à dessiner.
  position : Un tuple (x, y) indiquant la position où dessiner le message.

- **clear_img(self) :** Efface l'image en la remplissant avec la couleur de fond.

- **rtrn_img(self) :** Renvoie l'image actuelle.

- **convert(self, image) :** Convertit une image OpenCV en une image PIL et la redimensionne.
  image : L'image OpenCV à convertir.

- **bound_mssg(self, message, position, color)** : Dessine un message avec une boîte de délimitation à la position spécifiée avec la couleur spécifiée.
  message : Le message texte à dessiner.
  position : Un tuple (x, y) indiquant la position où dessiner le message.
  color : La couleur de la boîte de délimitation.

- **draw_arc_with_text(self, xy, start, end, fill='white', width=2, text_list=[], font=None) :** Dessine un arc avec du texte placé le long de celui-ci.
  xy : Un tuple (x0, y0, x1, y1) définissant la boîte englobante de l'arc.
  start : L'angle de départ de l'arc.
  end : L'angle de fin de l'arc.
  fill : La couleur de l'arc et du texte.
  width : La largeur de la ligne de l'arc.
  text_list : Une liste d'éléments texte à placer le long de l'arc.
  font : La police à utiliser pour le texte (optionnel).

Référence:

PIL documentation : https://pillow.readthedocs.io/en/stable/

Classe : Menu

Constructeur :

__init__(self, screen, draw0, draw1, draw2, molette, button, camera) : Initialise un objet Menu avec des composants nécessaires pour l'affichage, le dessin, la navigation, et la capture vidéo.

screen : Instance de la classe Screen représentant l'écran d'affichage.

draw0, draw1, draw2 : Instances de la classe Draw pour dessiner différents écrans de menu (draw0 pour le menu principal, draw1 pour le menu d'arrêt, draw2 pour le menu d'enregistrement).

molette : Instance de la classe RotaryEncoder pour contrôler la navigation dans le menu.

button : Instance de la classe Button pour gérer les pressions de bouton.

camera : Instance de la classe Camera pour la capture et l'enregistrement vidéo.

Méthodes :

- **add_item(self, name, wfps, dimensions, char, mode) :** Ajoute un élément au menu avec ses réglages associés.
  name : Nom ou libellé de l'élément de menu.
  wfps : Taux d'images par seconde pour l'enregistrement vidéo.
  dimensions : Dimensions de résolution pour l'enregistrement vidéo.
  char : Caractéristique supplémentaire ou information sur l'élément de menu.
  mode : Mode opérationnel de l'élément de menu.
- **menu_0(self) :** Affiche le menu principal (draw0) avec des options sélectionnables.
  Description : Efface le canevas draw0, dessine les éléments du menu en forme d'arc, et affiche le menu sur l'écran.
- **control_menu_0(self)** : Contrôle la navigation et l'interaction au sein de menu_0.
  Description : Met à jour la sélection actuelle à l'aide de l'encodeur rotatif (molette). Met en surbrillance l'élément de menu sélectionné à l'écran. Gère les pressions de bouton pour lancer des actions.
- **menu_1(self) :** Affiche le menu de confirmation d'arrêt (draw1).
  Description : Efface le canevas draw1 et affiche un message de confirmation d'arrêt à l'écran.
- **control_menu_1(self) :** Contrôle l'interaction au sein de menu_1.
  Description : Lance l'arrêt du système lors d'une pression sur le bouton. Efface le message de confirmation d'arrêt après l'exécution de la commande d'arrêt.
- **menu_2(self) :** Méthode de placeholder pour une future option de menu.
  Description : Actuellement ne fait rien mais prévue pour une fonctionnalité future.
- **control_menu_2(self) :** Méthode de placeholder pour contrôler l'interaction au sein de menu_2.
  Description : Actuellement ne fait rien mais prévue pour une fonctionnalité future.
- **menu_3(self) :** Affiche le menu d'enregistrement (draw2).
  Description : Efface le canevas draw2 et affiche un message d'indication d'enregistrement à l'écran.
- **control_menu_3(self) :** Contrôle l'interaction au sein de menu_3.
  Description : Capture des images depuis la caméra et les enregistre en continu jusqu'à ce qu'une pression sur le bouton indique d'arrêter. Affiche le statut d'enregistrement à l'écran.
- **menu_control(self) :** Méthode principale pour contrôler le système de menu.
  Description : Met à jour et affiche continuellement le menu actuel en fonction de

l'entrée de l'utilisateur (encodeur rotatif et pressions de bouton). Exécute les actions de menu correspondantes telles que la navigation dans les menus, l'arrêt du système, le démarrage de l'enregistrement vidéo et l'affichage de la vue de la caméra.

FIN

Revision #25
Created 30 June 2024 19:40:02 by El Ghoul Ali
Updated 5 July 2024 19:01:25 by El Ghoul Ali