21/05/2021

Projet Informatique – 1ère année
PRO 3600

Curseur Oculaire

Livrable 1

Adam DAHAN
Alexandre CHAUSSARD
Simon CHEREL
Tom SALEMBIEN
Christophe TROALEN

Tuteur : Patrick HORAIN

Table des matières

1. Introduction

L'oculométrie est le terme généralement employé pour parler de suivi oculaire.
Il s'agit d'un ensemble de techniques permettant d'enregistrer et d'analyser les mouvements des yeux dans une vidéo. Pour des questions de précision, on emploie souvent des caméras infra-rouges lors de la phase de détection de la pupille car elle est particulièrement noire et délimitée sur l'image.

Dès lors, les techniques d'oculométrie permettent de calculer la direction du regard d'un individu face à une caméra. Une fois la détection de l'œil effectuée, l'objectif est d'étudier différentes caractéristiques du regard (direction, inquiétude, …) pour en tirer des informations sur le sujet d'étude.
On retrouve notamment l'analyse de ces données obtenues dans le cadre de la psychologie (autisme et communication involontaire), la médecine (pour les patients atteints de locked-in syndrom ou ALS), le marketing, les recherches sur le système visuel (communication entre individus) ou encore les interactions homme-machine comme c'est notre cas.

En effet, nous nous intéressons ici au contrôle du pointeur de la souris par le regard. Cette technologie alors relativement récente et novatrice a toutefois déjà connu bon nombre de logiciels et dispositif permettant sa bonne réalisation. On notera particulièrement le premier oculomètre par Edmund Huey en 1908 dans le cadre de la lecture, jusqu'au premiers contrôleurs de curseur de Tobii Technology, leader dans le domaine, autour des années 2008 (*source : Wikipédia*).
Par ailleurs, dans le cadre la crise sanitaire du COVID-19, la technologie de suivi oculaire pourrait connaitre un certain essor dans la mesure où elle permet un respect des gestes barrières en minimisant les contacts avec les machines publiques.

1

2. Cahier des charges

2.1 Description de la demande :

I. Les objectifs

Nous cherchons à développer un logiciel capable de reconnaître le visage d'un individu, reconnaitre les yeux du sujet, cibler la pupille, puis coordonner les mouvements de la pupille avec ceux du curseur en plaçant ce dernier à l'endroit où le sujet regarde l'écran.
On notera également le besoin d'une interface légère présentant 2 boutons permettant pour l'un d'échelonner le pointeur par rapport au regard, pour l'autre de quitter l'application.

En raison de la qualité des bibliothèques présentes dans le domaine de l'analyse d'image et la reconnaissance sémantique de forme, nous avons choisi de développer notre projet sous Python en utilisant la bibliothèque OpenCV (opencv.org).
Pour ce qui est des boutons on pourra utiliser la bibliothèque Tkinter implémentée sur Python.

Dès lors, la réalisation du projet passera par plusieurs étapes :

- Intégrer la caméra dans Python (potentiellement la caméra infra-rouge selon la disponibilité de notre Kinect 2)
- Détecter le visage du sujet face à la caméra en utilisant le FaceDetect d'OpenCV
- Donner la position des yeux dans l'image
- Cibler un œil sous forme d'imagette pour limiter les traitements d'image
- Détecter la pupille dans l'imagette par des méthodes algorithmiques (préférée dans le cadre du PRO3600) ou des méthodes d'apprentissages (déjà établies dans des bibliothèques comme OpenCV)
- Effectuer un étalonnage du regard à l'aide de points sur l'écran. Le nombre de points permet de définir une approximation polynomiale. On partira pour l'instant sur une approximation d'ordre 4.
- Estimer la position du curseur sur l'écran et appliquer la position au curseur à l'aide d'une bibliothèque adaptée (win32api & win32con ou encore PyAutoGUI)
- Afficher deux boutons d'interfaces effectuant pour l'un « quitter l'application » et l'autre lançant l'échelonnage.

II. Produits du projet

Dans ce projet, nous nous appuierons sur différentes bibliothèques :

- OpenCV : bibliothèque de détection du visage et des yeux et de traitement d'image
- Numpy : bibliothèque de calcul scientifique de Python utilisée par openCV
- PyAutoGUI : bibliothèque assurant le contrôle du curseur (pyautogui.readthedocs.io)
- Tkinter : bibliothèque graphique de Python (docs.python.org)
- Keyboard : bibliothèque clavier (https://pypi.org/project/keyboard/)
- Threading : bibliothèque de threads (https://docs.python.org/3/library/threading.html)

Nous aurons également besoin de matériel physique :

- Webcam ou Kinect 2 (pour l'infra-rouge, présente sur le campus et possédée par l'équipe)
- Si Kinect 2, un adaptateur USB pour l'utilisation sur ordinateur

III. Les fonctions du produit

Les fonctionnalités qui seront délivrées par le programme Python sont :

En ce qui concerne l'image et son traitement :

- Récupérer une image depuis la caméra lisible sous OpenCV (infra-rouge ou webcam)
- Traiter une image pour détecter le visage et les yeux et renvoyer ces deux informations sous format traitable par OpenCV
- Récupérer une imagette de l'œil droit présent sur une image d'un visage
- Traiter l'image par des techniques algorithmiques (méthode du gradient, seuils colorimétriques, méthode de Canny, méthode de Hough)
- Appliquer des effets visuels sur une image pour montrer où la détection d'un objet reconnu (œil, visage) a été faite
- Afficher une image
- Supprimer les fenêtres ouvertes par OpenCV

En ce qui concerne le curseur :

- Donner la priorité de mouvement à la souris
- Afficher l'étalonnage du regard par des points sur lesquels cliquer pour commencer à utiliser le mode oculaire
- Trouver la position du curseur sur l'écran par rapport à la position du regard
- Déplacer le curseur à une position
- Terminer l'utilisation du mode contrôle oculaire

En ce qui concerne l'interface :

- Afficher 2 boutons distincts
- Quitter l'application au clic sur le bouton « Quitter l'application »
- Lancer l'échelonnage au clic sur le bouton « Echelonner »
- Lancer un étalonnage du seuil au démarrage du programme avec un bouton « Valider » et une échelle d'étalonnage de 0 à 100
- Appuyer sur Q pour quitter le programme

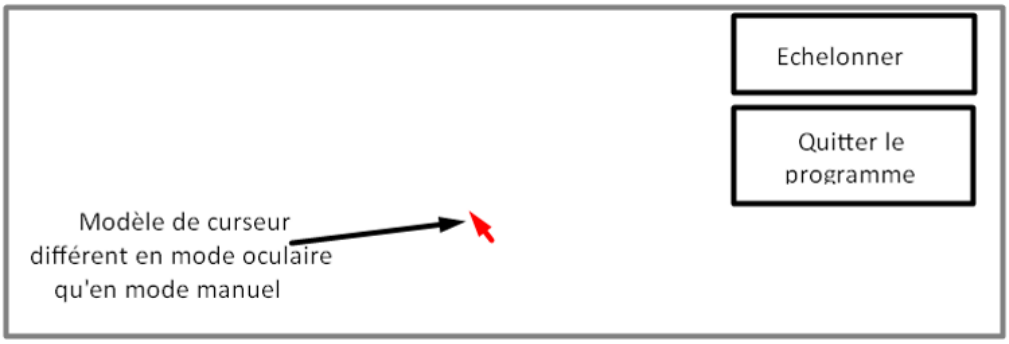
IV. Critères d'acceptabilité et de réception

Le bon fonctionnement de la technologie de suivi oculaire impose que le programme soit d'abord performant (utilisation en temps réel).

Il doit aussi être robuste : pouvoir fonctionner en cas de perte de contact visuel et proposer éventuellement un nouvel échelonnage si la précision devient faible.

Par ailleurs, le programme doit permettre la prise de contrôle manuel avec la souris (priorité à la souris pour le contrôle du curseur).

Le programme en fonctionnement devra s'appuyer sur la maquette suivante :



2.2 Contraintes

I. Contraintes de coûts

La caméra infra-rouge Kinect 2 a été achetée avant le projet par un membre de l'équipe.
Les bibliothèques étant toutes libres, il n'y a aucun frais pour ce projet.

II. Contraintes de délais

Le livrable final du projet est attendu pour le 21 mai, ce qui en fait notre date butoir.
Toutefois, le projet est attendu avant sur 2 livrables le 12 Février (structure) ainsi que le 16 mars (prototype) sur des objectifs précis.

III. Contraintes techniques

Comme on peut l'imaginer, la détection de la pupille dans une image est un challenge technique.

Aussi, il est important de bien faire attention à la précision de notre caméra qualifiée ici par sa résolution. En effet, une image de résolution trop faible peut entrainer une forte incertitude sur la détection de la pupille. De fait il y a un fort risque de faire des déplacements de curseurs erronés à cause d'une image de qualité médiocre.
Pour répondre à ce premier problème, nous proposons d'utiliser des caméras de bonne résolution (720p, 1080p) ou encore la Kinect 2 à haute résolution et sous infra-rouge, permettant ainsi une détection bien meilleure de la pupille en raison de son absorption importante dans l'infrarouge.
Toutefois cela pose une autre contrainte technique à résoudre : interfacer la Kinect 2 sur Python.
N'ayant pas encore la Kinect 2 en main, l'idée est pour l'instant en attente.
Sinon, les méthodes algorithmiques peuvent permettre d'améliorer de façon signifiante la qualité de traitement par analyse des formes (notamment les cercles dans notre cas), ce qui peut limiter le problème de résolution de la caméra.

Par ailleurs, il est faux de considérer qu'un visage plein face à la caméra est équivalent à un visage de biais sur la caméra. Ainsi pour limiter les erreurs de calculs liés à la profondeur de champ, nous nous plaçons dans des conditions idéales : il faut être bien en face de la caméra, avec un bon éclairage pour que le visage soit visible. Sinon il faudrait effectuer un étalonnage 3D, mais cela nous a été déconseillé par notre client M. Horain dans le cadre du PRO3600.

V. Clauses juridiques, etc.

Les bibliothèques étant en licence libre d'utilisation, le cadre juridique est exclu de notre projet.
On notera toutefois l'importance du droit d'image, aussi nous n'enregistrons aucune image captée par la caméra en dehors du traitement.

2.3 Déroulement du projet – Planification des Phases et Ressources

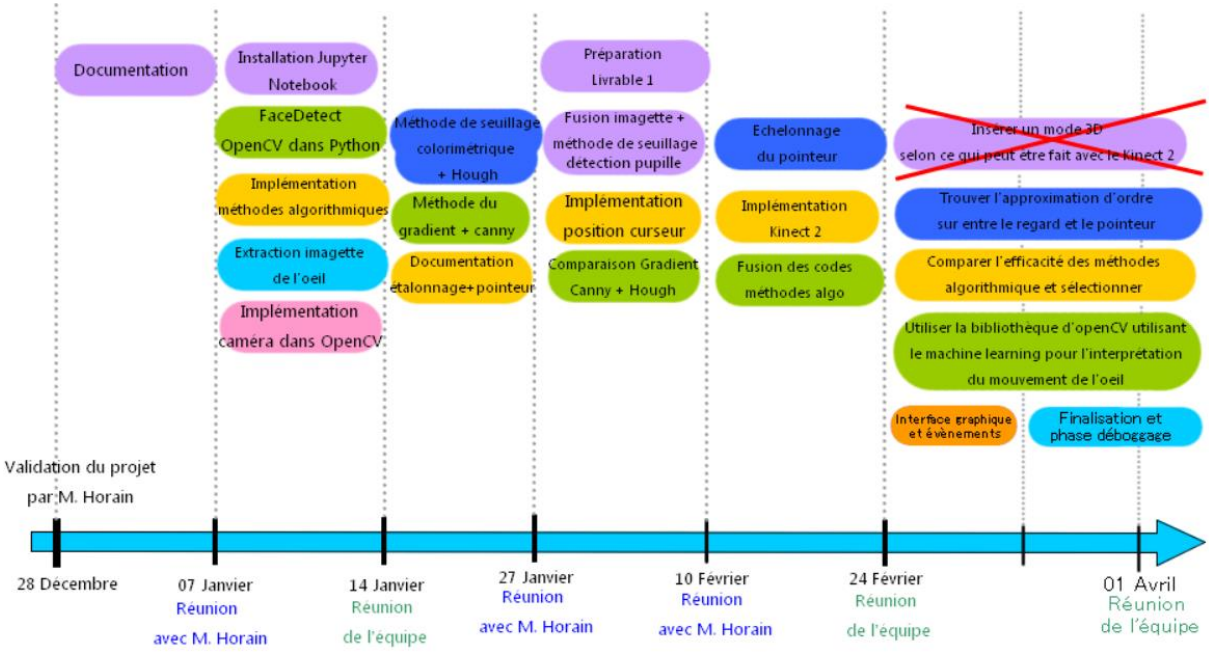


Diagramme Gantt du projet

Ce diagramme présente les tâches effectuées jusqu'à présent chronologiquement par l'équipe.
On notera que **les tâches violettes sont des tâches de groupe** et les tâches des autres couleurs sont des tâches individuelles ou en binôme.

2.4 Authentification – Date & signature du chef de projet et du maître d'ouvrage

Simon Chérel

Adam Dahan

Alexandre Chaussard

Tom Salembien

Christophe Troalen

2.5 Annexes – Lister et joindre au cahier des charges les éventuels documents que le client peut mettre à disposition

Fichier à disposition : « cours transformation de Hough »

3. Développement

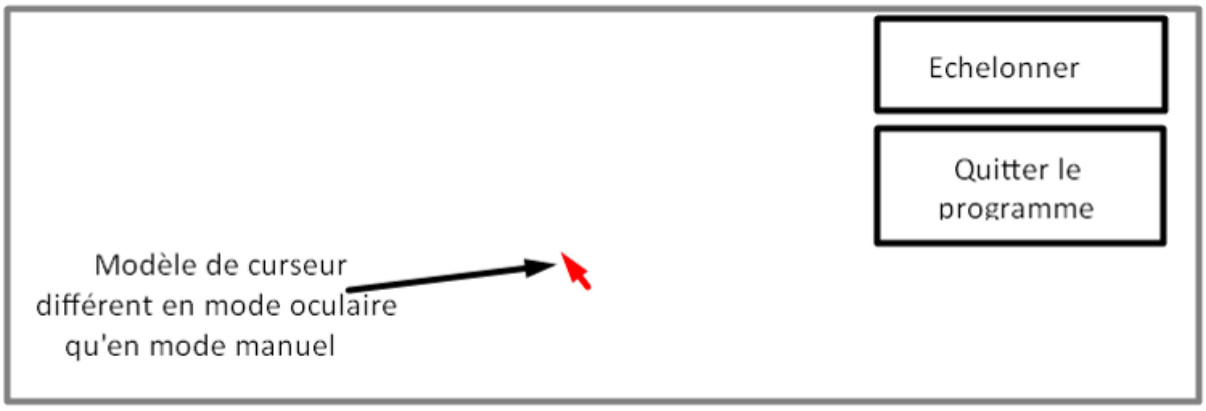
3.1 Analyse du problème et spécification fonctionnelle

Après lancement du programme, il est proposé à l'utilisateur une phase de calibrage du seuil de détection de la pupille, puis d'étalonnage.
Cette deuxième étape va permettre d'établir la relation mathématique entre la translation de la pupille et la position du curseur sur l'écran.
Pour ce faire, il sera imposé à l'utilisateur de regarder des points précis de l'écran comme par exemple les coins, sur lesquels il devra cliquer afin d'enregistrer le vecteur déplacement de sa pupille dans le programme.
Ainsi, le logiciel disposera des données nécessaires pour permettre le déplacement du curseur sur l'écran selon le regard définit par la pupille.

Cependant, si cette étape rencontre une quelconque difficulté, c'est tout le calibrage qui peut être faussé. Par exemple, un mauvais dimensionnement de l'écran peut entraîner l'existence de zones inaccessibles sur l'écran par le curseur oculaire ou encore un "dépassement" des limites de l'écran par celui-ci. De ce fait, les courbes de régression sont présentées à l'utilisateur et celui-ci peut choisir de recommencer l'opération ou valider la régression. Si toutefois il lance le mode contrôle oculaire, la touche Q lui permettra de quitter le programme instantanément. La présence de deux boutons de contrôle : "Échelonner" et "Quitter le programme" accessibles par la souris, assureront aussi la possibilité de rééchelonner le programme ou de pouvoir quitter l'application.

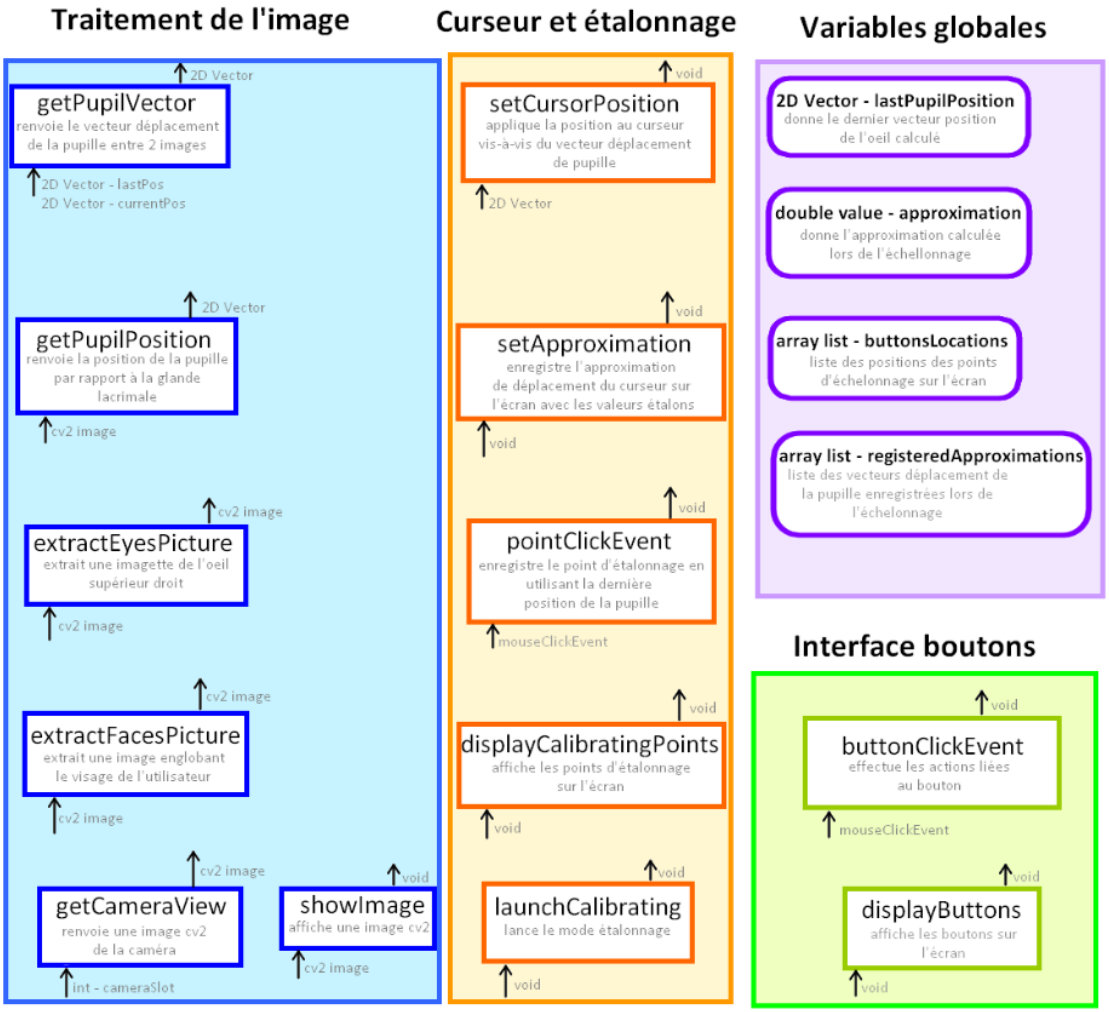
Afin d'effectuer le test de validation de notre programme, nous verrons si la phase d'étalonnage assurera bien un déplacement du curseur sur l'entièreté de l'écran avec précision. En effet en matière d'exigence du client, il n'est à tester que si le curseur est précis par rapport au point regardé et s'il peut atteindre tout point de l'écran. Le cas échéant, il faudra revoir la technique de fond d'étalonnage ou penser à utiliser une caméra à résolution plus élevée pour diminuer les incertitudes de calcul.

Enfin nous replaçons ici la maquette du produit imaginé :



3.2 Conception préliminaire

La 1ère maquette du programme s'appuyait sur la description haut-niveau suivante :



Nous prévoyons les tests d'intégration suivants :

- getCameraView : vérifier que l'entier entrant correspond à un slot connecté (par défaut 0)
- showImage, extractFacesPicture, extractEyesPicture, getPupilPosition : vérifié que l'image d'entrée est non « None » et que l'affichage correspond à l'image souhaitée (on pourra s'appuyer sur le showImage après avoir vérifié son fonctionnement)
- getPupilVector : le vecteur sortant ne doit pas être nul et les vecteurs entrants initialisés. Pour s'assurer du bon fonctionnement, on pourra s'appuyer sur le rendu visuel image
- setCursorPosition : le déplacement du curseur peut être visualisé sur l'écran. Le vecteur entrant étant getPupilVector, nous n'avons pas besoin de vérifier son intégrité.
- Les autres fonctions sont des rendus visuels, l'affichage suffira donc à vérifier le fonctionnement.

Toutefois, l'évolution du code a fortement amélioré ce modèle (voire conception détaillée).

3.3 Conception détaillée

```
class Main:
 procédure  start (): CodeRetour
      si (pas de seuillage en cours):
          # Lancement de l'affichage seuillée de l'œil

      si (pas étalonnage en cours):
         #Récupère la position de l'œil
         [eyeX, eyeY] ← ImageProcessing.getPupilPosition(scaler.getCameraView())
         #Position estimée du curseur
          pos←[coordonnées estimées de x,coordonnées estimées de y]
        # Déplace le curseur à cette position
         scaler.setCursorPosition(pos)
        #puis enregistre les positions
         scaler.lastPupilPosition[0], scaler.lastPupilPosition[1] ← pos[0], pos[1]
   fproc

# Class définissant des boutons ronds
class RoundButton(tk.Canvas):

 fonction __init__ (self, parent, largeur, hauteur, cornerradius , padding, couleur, bg,
command=None):

#cornerradius : degré auquel les angles sont arrondis

      tk.Canvas.__init__(self, parent, borderwidth=0,
                  relief="flat", highlightthickness=0, bg=bg)

      si cornerradius est plus grand que 0.5 * largueur:
         sortir "Error: cornerradius is greater than width."

      si cornerradius est plus grand que 0.5 * hauteur:
         sortir "Error: cornerradius is greater than height.")

      rad← 2 * cornerradius
ffct

procédure shape():
       #création des différents cercles à afficher lors de l'étalonnage
       #on créé d'abord un polygone puis différents arc de cercle pour pouvoir effectuer          .
#régression par la suite avec ces points de référence pour donnée d'entrée
   fproc

      id = shape()

      #self bbox
      (x0, y0, x1, y1) ← coordonnées d'un rectangle regroupant tous les items
      largeur ←(x1 - x0)
      hauteur←(y1 – y0)
```

```
        #configuration de la fenetre et des boutons
        self.configure(width=largeur, height=hauteur)
        self.bind("<ButtonPress-1>", self._on_press)
        self.bind("<ButtonRelease-1>", self._on_release)

ffct

    procédure _on_press(self, event):
      #lorsque l'on clique sur le cercle rouges(bouton)
        self.configure(relief←"sunken")
        si scaler n'est pas nul (lorsque l'on n'a pas encore cliqué)
            alors invoquer la fonction pointClickEvent() de CcalingursorS

    fproc

    procédure _on_release(self, event):
      #lorsque l'on relanche un bouton
        self.configure(relief←"raised")
    fproc

class CursorScaling:

    # Constructeur de CursorScaling
    procédure __init__(self):
        # Donne le dernier vecteur position de l'oeil calculé
        self.lastPupilPosition ← initialise avec liste de 0

        # Donne l'approximation calculée lors de l'échellonnage
        self.approximation ← liste vide

        # Liste des positions des points d'échelonnage sur l'écran
        self.buttonsLocations ← [[0, 0], [0, 30], [0, 60]]

        # Liste des position de la pupille
        # enregistrés lors de l'échelonnage avec les positions des points cibles
        self.registeredApproximations ←liste vide
self.
        # Booléan indiquant si l'on est en mode étalonnage ou non
        self.isScaling ←False

       # Valeur du seuillage en luminosité pour définir le seuil de détection de l'oeil
        self.eyeThreshold ← 10

      # Booléan indiquant si l'on est en mode étalonnage ou non
      self.isScaling ← True

# Instance de la caméra
self.cap← None

# Instance de la fenêtre d'étalonnage
self.fenetre ←None
```

```
 fproc

#Fonction qui récupère l'image capturée par la caméra
fonction getCameraView(self, cameraSlot=0):

   si pas de capture du flux vidéo en cours :
      self.cap = cv2.VideoCapture(cameraSlot)
   _, frame = self.cap.read()

   renvoyer frame

ffct


 # Applique la position au curseur vis-à-vis du vecteur
   # déplacement de la pupille
   procédure setCursorPosition(self, vector):
      pyautogui.moveTo(vector[0], vector[1])

   # Constitue l'hypothèse de régression
   # sur l'écran étalonné les valeurs étalons
   # et optimisée
   # theta défini le vecteur de paramétrisation
   # x défini le vecteur d'entrer nouvelle
 fproc

  fonction hypothesis(self, theta, x):
      # On définie le vecteur sortie supposé
      y = 0
      # On construit y = sum(theta_i*x_i)
      Pour i allant de 0 à taille du vecteur x :
         y ← theta[i] * x[i]
      Affiche y
   ffct


   # Enregistre le point d'étalonnage en utilisation
   # la dernière position de la pupille
   fonction pointClickEvent(self):

# On récupère les coordonnées du curseur
      mouseX, mouseY ← coordonnées du curseur sur l'écran

      # On récupère les coordonnées de la pupille
      [eyeX, eyeY] ← coordonnées de la pupille
      Affiche(position de l'oeil)

      # Stockage des données pour la regression
      self.registeredApproximations.append([mouseX, mouseY, eyeX, eyeY])
```

```
        # On supprime le bouton cliqué si tout a bien marché
        button.destroy()


        # Condition définissant que tous les boutons ont été cliqués
        si la taille de la matrice stockant les données brutes=taille de la matrice stockant la position de
boutons affichés à l'écran :

            # Construction des matrice échantillon :
            # échantillons d'entrée : eyeX, eyeY
            X ← liste vide
            # échantillons de sortie : mouseX, mouseY
            YX ← liste vide
            YY ← liste vide
            Pour i allant de 0 à taille de la matrice stockant les données brutes :
                X ← concatène avec 1 suivi des coordonnées X,Y des yeux
                YX← concatène avec les coordonnées X de la souris
                YY ← concatène avec 1 suivi des corrdonnées Y de la souris

            # Transformation matricielle puis transposition pour coller à la formule de l'équation normale
            X ←  np.array(X)
            X.transpose()
            YX ← np.array(YX)
            YX.transpose()
            YY ← np.array(YY)
            YY.transpose()

            # Détermination des paramètres optimales pour les régressions

            #Opérations matricielles pour optimiser theta=(XX^T)^-1X^TY

            # Enregistrement des vecteurs optimisés
            self.approximation ← [thetaX, thetaY]
    # On supprime la fenêtre étalonnage
    si la fenetre n'est pas vide :
       détruire la fenetre avec fenetre.destroy()

    #tracés des représentations graphiques après l'étalonnage et la régression avec la bibliothèque
    matplotlib
    #tracés les axes
    #construit les boutons pour valider valider,recommencer,quitter la régression
    validationRegression ← mButton(ax1, "Valider")
    refaireRegression ← mButton(ax2, "Recommencer")
    quitReg ← mButton(ax3, "Quitter")

     ffct
       # Liste les positions des boutons étalons et initialise buttonsLocations
       procédure setButtonsPosition(self, xmax, ymax):
          side_width ← 3
          diam ← 100
          ray ← diam // 2
          xmilieu ← xmax // 2
```

```
        ymilieu ← ymax // 2
        self.buttonsLocations ← [coordonnées des buttons]
    fproc

    # Affiche les points d'étalonnage à l'écran
    fonction displayCalibratingPoints(self):
        # Appel d'une fenêtre tk
        fenetre ← tk.Tk()
        # Mode fullscreen
        fenetre.attributes('-fullscreen', True)
        # Echap permet de quitter l'interface
        fenetre.bind('<Escape>', lambda e: fenetre.destroy())
        # Récupération des informations de l'écran
        largeur ← fenetre.winfo_screenwidth()
        hauteur ← fenetre.winfo_screenheight()

            # création de la fenêtre avec tk.Canvas
            # initialisation des positions des boutons étalons
        self.setPosition(largeur, hauteur)
        # Placement des boutons
        Pour i allant de 0 à la taille de buttonsLocations :
            #Création de chacun des 9 voutons
            # Ajout du bouton à la boucle tk

        # Lancement du scheduler tk
        fenetre.mainloop()
        # Destruction de la fenêtre
        fenetre.destroy()
ffct
    # Affiche la vue seuillée de la caméra
fonction runThresholdView(self):

    Tant que le seuillage n'est pas terminé :

        # Récupération de l'image seuillée avec getThresholdedEye(imagette de l'oeil niveau de gris)
        si l'image seuillé est bien détecté:

            # Affichage de l'image seuillée avec showImage
            ImageProcessing.showImage(thresholdView, "Etape 1 - Etalonnage du seuil")
            cv2.waitKey(20)
ffct

# Affiche l'interface d'étalonnage du seuillage
fonction displayThresholdCalibration(self):

    # Appel d'une fenêtre tk.Tk() et définition de la géométrie
    # Echap permet de quitter l'interface
    # Récupération des informations de l'écran largeur et hauteur avec la bibliothèque pyautogui
    # création de la fenêtre
ffct
```

```
# Permet la validation de l'étape de seuillage

fonction validateThreshold():
    # On valide l'étape de seuillage
    self.isThresholded ← True
    # On supprime toutes les fenêtres
    # On lance la calibration curseur en affichant les points de calibration
    self.displayCalibratingPoints()
    #mise en forme du bouton de validation
ffct


# Lance le mode étalonnage
fonction launchCalibrating(self):

  # On coupe les fenêtres tkInter dans le cas d'un réétalonnage

  # On montre au programme qu'il est en mode étalonnage
  self.isScaling ← True
  # On reset l'état du thresholding
  self.isThresholded ← False
  # On reset les anciennes données de régression avec registeredApproximations.clear()
  # On affiche l'étape de seuillage avec displayThresholdCalibration()

ffct

class ImageProcessing:

# Renvoie le vecteur déplacement de la pupille entre 2 images
fonction getPupilVector(lastPos, currentPos):
  return [lastPos[0] - currentPos[0], lastPos[1] - currentPos[1]]
ffct

# Renvoie la position de la pupille par rapport à la glande lacrimale
fonction getPupilPosition(image):
  # Extraction de l'imagette de l'oeil supérieur droit détecté
  eye_color ← extractEyesPicture(image)
  # Condition de détection
  si aucun œil n'est détecté:
    affiche("Eye_color is None")
    rencoyer [0, 0]

  # Passage en noir et blanc pour la réduction d'information et une meilleure détection
  gray ← cv2.cvtColor(eye_color, cv2.COLOR_BGR2GRAY)

  # Récupération de la taille de l'image
  rows, cols ← gray.shape
  # Application d'un flou gaussien
  gray_blurred ← cv2.GaussianBlur(gray, (7, 7), 0)
  # Seuillage pour passer l'image en binaire inversé pixel noir si supérieur au seuil et blanc sinon
  _, threshold ← cv2.threshold(gray_blurred, 80, 255, cv2.THRESH_BINARY_INV)
  # Recherche des contours de l'oeil
```

```
    contours ← cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    # Conditions de détection de l'oeil
    si aucun contours n'est détecté :
        afficher("No eye contour detected")
        renvoyer [0, 0]
    if la taille du vecteur contours est nulle
        afficher("No eye contour detected")
        renvoyer [0, 0]

    # Récupération des coordonnées du rectangle minimisé englobant la pupille
    (x, y, w, h) ← cv2.boundingRect(contours[0])

    Renvoyer [coordonnées de la pupille]
ffct


# Extrait une imagette de l'oeil supérieur droit
fonction extractEyesPicture(image):
    # Méthode des ondelettes de Haar pour la détection des yeux
    eye_cascade ←  'haarcascade_eye.xml'

    # Définiton de l'image du visage en couleur
    roi_color, w, h ← extractFacesPicture(image);
    # Condition de détection du visage
    si aucun visage n'est détecté
        Affiche("roi_color is None")
        Renvoyer None
    # Passage en noir et blanc pour la réduction d'information et une meilleure détection
    gray ← cv2.cvtColor(roi_color, cv2.COLOR_BGR2GRAY)
    # Détection des yeux
    eyes ← eye_cascade.detectMultiScale(gray)

    # Définition de l'image extraite
    extracted = None
    # Bouclage sur les yeux détectés
    for (ex, ey, ew, eh) in eyes:
        # Condition de détection de l'oeil supérieur droit
       # Récupère imagette de l'oeil gauche en niveau de gris puis en couleur
            extracted ← roi_color[ey:ey + eh, ex:ex + ew]

    return extracted
ffct

# Extrait une image englobant le visage de l'utilisateur
fonction extractFacesPicture(image):
    # Méthode des ondelettes de Haar pour la détection de visage
    face_cascade ←  'haarcascade_frontalface_default.xml')
    # Passage en noir et blanc pour la réduction d'information et une meilleure détection
    gray ← cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Détection des visages
    faces ← face_cascade.detectMultiScale(gray, 1.3, 5)
```

```
    # Conditions de détection du visage

    si aucun visage n'est détecté:
        afficher("No face detected")
        Renvoyer None, None, None

    # Extraction des données sur l'image du premier visage détecté
    (x, y, w, h) ← faces[0]
    # Extraction du 1er visage détecté sur l'image en couleur
    extracted ← image[y:y + h, x:x + w]
    Renvoyer extracted, w, h
ffct

# Renvoie une image cv2 de la caméra
fonction getCameraView(cameraSlot=0):
    cap ← cv2.VideoCapture(cameraSlot, cv2.CAP_DSHOW)
    _, frame ← cap.read()
    Afficher frame
ffct

# Permet d'afficher à l'écran une image (utile pour le debug)
fonction showImage(image, imageName="noName"):
    cv2.imshow(imageName, image)
ffct

class Interface:

# Affiche les boutons utilisateurs (exit & étalonnage)
fonction displayButtons(cs):

    # Appel d'une fenêtre tk

    fenetre ← tk.Tk()
    # Mode fullscreen
    # Echap permet de quitter l'interface

    fenetre.bind('<Escape>', lambda e: fenetre.destroy())
    # Récupération des informations de l'écran

    largeur ← fenetre.winfo_screenwidth()
    hauteur ← fenetre.winfo_screenheight()

    # création de la fenêtre
    canvas ← tk.Canvas(fenetre, width=width, height=height, bg='black')
    canvas.pack()

    # initialisation des positions des boutons étalons
    # destruction des fenetres
    fenetre.mainloop()
    fenetre.destroy()
    return
ffct
```

3.4 Codage

Le code respecte le format doxygen, ce qui permet sa compréhension lors de l'implémentation des fonctions ou d'appels depuis un autre code.

3.5 Tests unitaires

Tests unitaires :

Dans le but de vérifier la fiabilité de notre programme. Nous avons effectué quand cela était possible des tests unitaires sur chaque fonctions viables. Ainsi nous avons pu vérifier de la véracité des fonctionnalités de notre code dans les conditions d'utilisations.

Voici l'exemple des tests unitaires effectués à l'aide de la librairie pytest :

```python
import Interface
import ImageProcessing
import RoundButton
from CursorScaling import CursorScaling
from RoundButton import RoundButton
import pytest
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
cs=CursorScaling()
img_oeil = mpimg.imread('pupille.jpg')
img_noir = mpimg.imread('noir.png')
img_tete = mpimg.imread('tete.png')
img_unique_tete = mpimg.imread('tete_unique.png')

def test_getPupilVector():
    assert ImageProcessing.getPupilVector([1,1],[0,0])==[1,1]

def  test_getThresholdedEye():
    assert ImageProcessing.getThresholdedEye(img_oeil)==[292,300]

def test_extractEyesPicture():
    assert ImageProcessing.extractEyesPicture(img_noir)==[40]

def test_extractFacesPicture():
    assert ImageProcessing.extractFacesPicture(img_tete)==img_unique_tete

def test_hypothesis():
    assert cs.hypothesis([13,0.3],500)==133
```

Cette même librairie à renvoyé le résultat suivant :

« simon.cherel@macbook-pro:~/Documents/Études/Projet_info ➜ pytest TestsUnitaires.py 1 ↲ 22:43:16

```
============================= test session starts ==============================
platform darwin -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /Users/simon.cherel/Documents/Études/Projet_info
collected 1 item

TestsUnitaires.py .                                                      [100%]

============================= warnings summary ==============================
```

==========
====

../../../../../usr/local/lib/python3.9/site-packages/rubicon/objc/ctypes_patch.py:21

/usr/local/lib/python3.9/site-packages/rubicon/objc/ctypes_patch.py:21: UserWarning: rubicon.objc.ctypes_patch has only been tested with Python 3.4 through 3.8. You are using Python 3.9.5. Most likely things will work properly, but you may experience crashes if Python's internals have changed significantly.

warnings.warn(

-- Docs: https://docs.pytest.org/en/stable/warnings.html

========== 1 passed, 1 warning in 0.99s ==========”

Attestant du bon fonctionnement du programme.

3.6 Tests d’intégration

3.7 Tests de validation

La fonction start() de la classe Main étant appelante de l’ensemble des fonctions du programme par embriquement, un test d’intégration sur celle-ci permet d’assurer la viabilité de l’ensemble du programme.

4. Manuel utilisateur

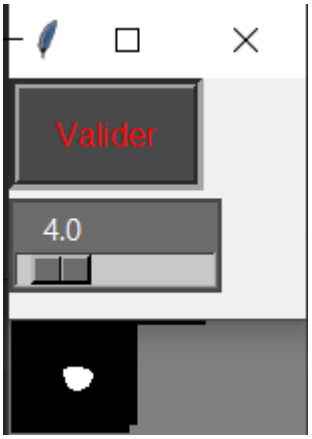
4.1 Lancement par l’utilisateur

Le programme peut être lancé par le terminal, sous réserve que python et l’ensemble des bibliothèques soient bien installée, ainsi qu’une caméra connectée.

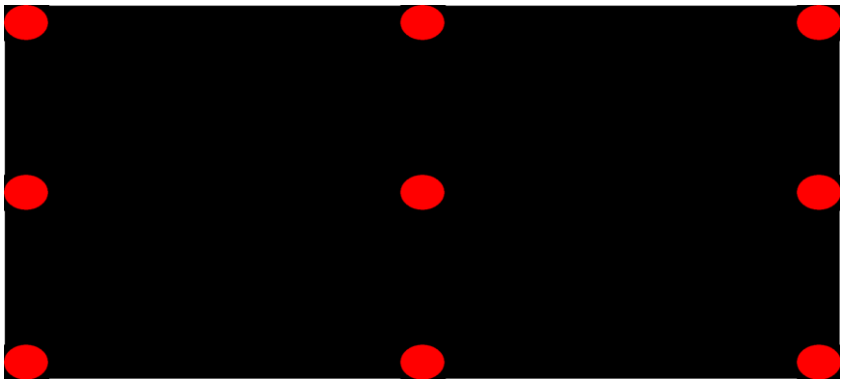
4.2 Réalisation des tests

4.3 Utilisation

Lorsque le programme se lance, la fenêtre d’étalonnage du seuil s’ouvre. Celle-ci permet de régler le seuil associé à l’image en noir et blanc de l’oeil droit de l’utilisateur capturée par sa webcam.
À l’aide du curseur horizontal, l’utilisateur règle le seuil jusqu’à ce que l’image seuillée de son oeil représente uniquement sa pupille en blanc et le reste en noir.

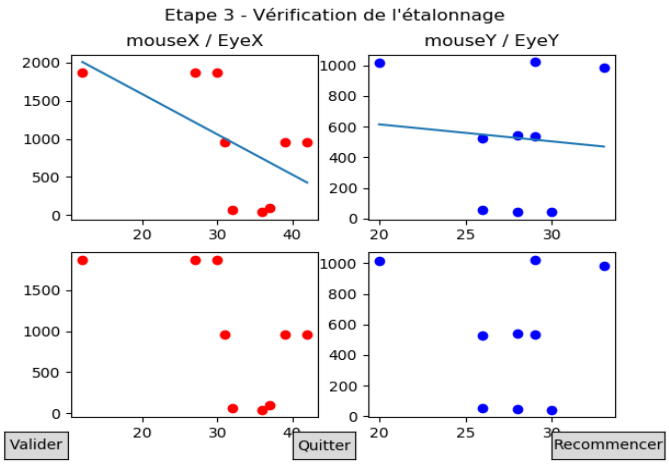


Après avoir cliqué sur le bouton « Valider », cette fenêtre se ferme pour ouvrir la fenêtre d'étalonnage, qui a pour but d'assurer d'enregistrer des échantillons de corrélation entre la position de l'œil et celle du curseur à des points précis.

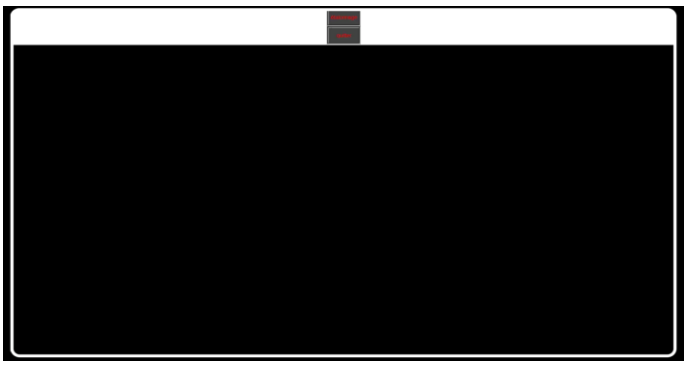


L'utilisateur clique sur les points rouges en fixant le curseur. En arrière-plan, la caméra enregistre la position de pupille à chaque clic sur les points étalons. Cela forme alors l'échantillon de corrélation entre l'œil et le curseur. Lorsque le point étalon est validé, il disparait.

Lorsque cette étape est terminée, la fenêtre d'étalonnage se ferme. Des graphes apparaissent ensuite à l'écran représentant les régressions linéaires associées à l'étalonnage. Plus les points des graphes s'alignent avec les régressions tracées, et plus le calibrage est réussi. 3 boutons sont présents et permettent respectivement de passer à la prochaine étape, arrêter le programme et recommencer l'étalonnage.



Si l'utilisateur clique sur « Valider », il est redirigé vers une ultime fenêtre présente ci-dessous qui clôt l'étalonnage. L'utilisateur contrôle le curseur avec son œil gauche dans cette fenêtre. Il peut également quitter le programme depuis le bouton disponible (la touche Q a la même fonction que le bouton Quitter) ou bien recommencer l'étalonnage si celui-ci n'est pas satisfaisant avec le bouton « Etalonnage ».



5. Conclusion

Ce rapport décrit les principales étapes du développement de la version « terminal » d'un détecteur oculaire.

Le projet réalisé s'est articulé autour de deux problématiques : la détection de l'œil et la retranscription du regard sur l'écran. La bibliothèque OpenCV sous Python a été utilisée pour toutes les problématiques de détection du visage, des yeux et enfin de la pupille de l'utilisateur sur l'image.

Une approche consistant à estimer les coordonnées de l'œil à travers un apprentissage machine a été proposée, à travers une régression linéaire via l'estimateur des moindres carrés. Celle-ci a permis d'observer la corrélation entre les variables de position (X,Y) œil-écran, et d'obtenir le lien entre la direction du regard à partir de la photo de l'utilisateur, et la zone effectivement regardée sur l'écran.

Un étalonnage où la capture des yeux était continue a permis d'obtenir une large base de données. Cette méthode, bien que fonctionnelle, présentait quelques points aberrants dus à un

mauvais éclairage du visage, des mouvements de tête ou encore des imprécisions de la part de l'utilisateur. Quelques problèmes pour la corrélation des coordonnées en Y ont également été constatés.

Ainsi, une capture discrète (photo par photo) plus contrôlable a permis de supprimer la présence de ces points aberrants, au moyen d'interfaces visuelles et interactives. Toutefois, cette méthode n'a pas permis de contourner les problèmes de corrélation en Y.

En inversant les données de la régression, un curseur indiquant la zone où se porte le regard de l'utilisateur peut être affiché à l'écran. Les résultats obtenus sont très satisfaisants pour ce qui est de la composante horizontale. Ils le sont un peu moins pour ce qui est du déplacement vertical de l'œil. Cela est en grande partie dû à un problème de précision de la caméra, étant donné que le déplacement haut-bas de l'œil est bien plus restreint que le déplacement droite-gauche.

Afin d'obtenir la version finale du curseur oculaire, il reste à implémenter une automatisation qui enclenche un clic dès lors que l'œil se stabilise sur une zone de l'écran

A. Code source

A.1 Code source commenté « à la doxygen »

```
import pyautogui
from CursorScaling import CursorScaling
import ImageProcessing
import threading
import keyboard

"""
Main correspond au programme principal à lancer pour lancer le programme
"""

# Définition du curseur étalon
scaler = CursorScaling()
# Dépacement de la sécurité grand mouvement
pyautogui.FAILSAFE = False


def start():
    """ Fonction principale du programme

    :return: None
    """
    # Définition de la fonction de gestion du curseur
    while True:
        if keyboard.is_pressed('q'):
            scaler.quitProcess()
        # Si l'on est en attente du seuillage
        if not scaler.isThresholded:
            # Lancement de l'affichage seuillée de l'oeil
            scaler.runThresholdView()
        # Condition de mise à jour de la position du curseur
        if not scaler.isScaling:
            # On récupère la position de l'oeil
            [eyeX, eyeY] = ImageProcessing.getPupilPosition(scaler.getCameraView())
            # On en déduit la position prévue du curseur Les index 0,1 références thetaX et thetaY optimisant
```

```
            # l'hypothèse de régression et ont été calculés lors de l'étalonnage
            pos = [scaler.hypothesis(scaler.approximation[0], eyeX), scaler.hypothesis(scaler.approximation[1],
eyeY)]
            # On déplace le curseur à cette position
            scaler.setCursorPosition(pos)
            mouseX, mouseY = pyautogui.position()
            print("\neyeX : ", eyeX, " eyeY : ", eyeY)
            print("predictedMouseX : ", pos[0], " predictedMouseY : ", pos[1])
            print("mouseX : ", mouseX, "mouseY : ", mouseY)
            # On enregistre le dernier vecteur position de la pupille
            scaler.lastPupilPosition[0], scaler.lastPupilPosition[1] = pos[0], pos[1]


threading.Thread(target=start).start()

# Lancement de l'étalonnage au démarrage du programme
scaler.launchCalibrating()

# Ferme le flux caméra
if scaler.cap is not None:
    scaler.cap.release()
```

```
import pyautogui
import ImageProcessing
import numpy as np
import Interface
from RoundButton import RoundButton
import tkinter as tk
import matplotlib
import cv2
import sys

matplotlib.use("TkAgg")
from matplotlib import pyplot as plt
from matplotlib.widgets import Button as mButton

"""
CursorScaling permet l'étalonnage du système
et le contrôle de cette phase
"""


class CursorScaling:

    # Constructeur de CursorScaling
    def __init__(self):
        """Constructeur de CursorScaling.

            lastPupilPosition : Donne le dernier vecteur position de l'oeil calculé
            approximation : Donne l'approximation calculée lors de l'échellonnage
            buttonsLocations : Liste des positions des points d'échelonnage sur l'écran
            registeredApproximations : Liste des position de la pupille enregistrés lors de l'échelonnage avec les
positions des points cibles
            isThresholded : Booléan indiquant la validation de l'étape de seuillage
            eyeThreshold : Valeur du seuillage en luminosité pour définir le seuil de détection de l'oeil
            cap : Instance de la caméra
```

```
        fenetre : Instance de la fenêtre d'étalonnage
        """
        pass
        # Donne le dernier vecteur position de l'oeil calculé
        self.lastPupilPosition = [0, 0]

        # Donne l'approximation calculée lors de l'échellonnage
        self.approximation = []

        # Liste des positions des points d'échelonnage sur l'écran
        self.buttonsLocations = [[0, 0], [0, 30], [0, 60]]

        # Liste des position de la pupille
        # enregistrés lors de l'échelonnage avec les positions des points cibles
        self.registeredApproximations = []

        # Booléan indiquant la validation de l'étape de seuillage
        self.isThresholded = False

        # Valeur du seuillage en luminosité pour définir le seuil de détection de l'oeil
        self.eyeThreshold = 10

        # Booléan indiquant si l'on est en mode étalonnage ou non
        self.isScaling = True

        # Instance de la caméra
        self.cap = None

        # Instance de la fenêtre d'étalonnage
        self.fenetre = None

    # Renvoie une image cv2 de la caméra
    def getCameraView(self, cameraSlot=0):
        """ Récupère l'image capturée par la caméra
        :param cameraSlot: slot de la caméra sur l'ordinateur (default = 0)
        :return: frame - image de la caméra
        """
        if self.cap is None:
            self.cap = cv2.VideoCapture(cameraSlot, cv2.CAP_DSHOW)
        _, frame = self.cap.read()
        return frame

    # Applique la position au curseur vis-à-vis du vecteur
    # déplacement de la pupille
    def setCursorPosition(self, vector):
        """ Applique la position au curseur au vecteur donnée sur l'écran

        :param vector: position où mettre le curseur sur l'écran
        :return: None
        """
        pyautogui.moveTo(vector[0], vector[1])

    # Constitue l'hypothèse de régression
    # sur l'écran étalonné les valeurs étalons
    # et optimisée
    # theta défini le vecteur de paramétrisation
    # x défini le vecteur d'entrer nouvelle
    def hypothesis(self, theta, x):
        """ Fonction hypothèse de la régression linéaire
```

```
    :param theta: vecteur paramètre de la régression
    :param x: position de l'oeil entrante (X ou Y)
    :return: y - Position estimée sur l'écran
    """
    # On définie le vecteur sortie supposé
    y = 0
    if True:
        return theta[0] + theta[1] * x
    # On construit y = sum(theta_i*x_i)
    for i in range(0, len(x)):
        y = theta[i] * x[i]
    return y

# Enregistre le point d'étalonnage en utilisation
# la dernière position de la pupille
def pointClickEvent(self, button):
    """ Définie les actions liées au clic sur les boutons d'étalonnage

    :param button: bouton sur lequel est effectué le clic
    :return: None
    """
    # On récupère les coordonnées du curseur
    mouseX, mouseY = pyautogui.position()

    # On récupère les coordonnées de la pupille
    eyeX, eyeY = ImageProcessing.getPupilPosition(self.getCameraView(), self.eyeThreshold)
    print("Position de l'oeil : " + str(eyeX) + ", " + str(eyeY))
    # Condition de détection de la pupille
    if eyeX == 0 and eyeY == 0:
        return
    # On enregistre la réalisation d'étalonnage en prévision de la régression
    self.registeredApproximations.append([mouseX, mouseY, eyeX, eyeY])
    # On supprime le bouton cliqué si tout a bien marché
    button.destroy()

    # Condition définissant que tous les boutons ont été cliqués
    if len(self.registeredApproximations) == len(self.buttonsLocations):

        # Construction des matrice échantillon :
        # échantillons d'entrée : eyeX, eyeY
        XX = []
        XY = []
        # échantillons de sortie : mouseX, mouseY
        YX = []
        YY = []
        simpleVecEyeX = []
        simpleVecEyeY = []
        for i in range(0, len(self.registeredApproximations)):
            simpleVecEyeX.append(self.registeredApproximations[i][2])
            simpleVecEyeY.append(self.registeredApproximations[i][3])
            XX.append([1, self.registeredApproximations[i][2]])
            XY.append([1, self.registeredApproximations[i][3]])
            YX.append(self.registeredApproximations[i][0])
            YY.append(self.registeredApproximations[i][1])

        # Transformation matricielle puis transposition pour coller à la formule de l'équation normale
        XX = np.array(XX)

        XY = np.array(XY)
```

```
        YX = np.array(YX)
        YY = np.array(YY)

        print("XX : \n", XX)
        print("\nYX : \n", YX)

        # Détermination des paramètres optimales pour les régressions
        invertMatrixX = np.linalg.inv((XX.transpose()).dot(XX))
        invertMatrixY = np.linalg.inv((XY.transpose()).dot(XY))

        thetaX_ = np.matmul(invertMatrixX, ((XX.transpose()).dot(YX)))
        thetaY_ = np.matmul(invertMatrixY, ((XY.transpose()).dot(YY)))

        # Enregistrement des vecteurs optimisés
        self.approximation = [thetaX_, thetaY_]

        # On supprime la fenêtre étalonnage
        if self.fenetre is not None:
            self.fenetre.destroy()

        fig, axs = plt.subplots(2, 2)
        fig.suptitle("Etape 3 - Vérification de l'étalonnage", fontsize=12)
        axs[0, 0].plot(simpleVecEyeX, YX, 'ro')
        axs[0, 0].set_title('mouseX / EyeX')
        axs[0, 1].plot(simpleVecEyeY, YY, 'bo')
        axs[0, 1].set_title('mouseY / EyeY')

        axs[1, 0].plot(simpleVecEyeX, YX, 'ro')
        axs[1, 1].plot(simpleVecEyeY, YY, 'bo')

        minX = simpleVecEyeX[0]
        maxX = simpleVecEyeX[0]
        for i in simpleVecEyeX:
            if minX > i:
                minX = i
            if maxX < i:
                maxX = i
        x1 = np.linspace(minX, maxX, 400)
        minX = simpleVecEyeY[0]
        maxX = simpleVecEyeY[0]
        for i in simpleVecEyeY:
            if minX > i:
                minX = i
            if maxX < i:
                maxX = i
        x2 = np.linspace(minX, maxX, 400)

        By1 = thetaX_[0] + thetaX_[1] * x1
        By2 = thetaY_[0] + thetaY_[1] * x2

        axs[0, 0].plot(x1, By1)
        axs[0, 1].plot(x2, By2)

        ax1 = plt.axes([0.03, 0.02, 0.09, 0.06])
        ax2 = plt.axes([0.81, 0.02, 0.16, 0.06])
        ax3 = plt.axes([0.44, 0.02, 0.09, 0.06])
        validationRegression = mButton(ax1, "Valider")
        refaireRegression = mButton(ax2, "Recommencer")
        quitReg = mButton(ax3, "Quitter")
```

```
        # Sous fonctions évenementielles

        def closePlot(event=None):
            # On ferme le plot
            plt.close(fig)
            # On clear matplotlib sinon il fatigue
            plt.cla()
            plt.clf()
            # On désactive le mode étalonnage
            self.isScaling = False
            # Une fois les opérations effectuées, on affiche les boutons de contrôle
            Interface.displayButtons(self)

        def relaunchRegression(event=None):
            # On ferme le plot
            plt.close(fig)
            # On clear matplotlib sinon il fatigue
            plt.cla()
            plt.clf()
            # On relance l'étalonnage
            self.launchCalibrating()

        def quit(event=None):
            # On ferme le plot
            plt.close('all')
            # On quitte le programme
            sys.exit(0)

        # Initialisation des events liés aux boutons
        validationRegression.on_clicked(closePlot)
        refaireRegression.on_clicked(relaunchRegression)
        quitReg.on_clicked(quit)

        # Création de références type "Dummy" pour que les boutons fonctionnent avec le fig.show()
        ax1._button = validationRegression
        ax2._button = refaireRegression
        ax3._button = quitReg

        # On affiche la figure de régression
        fig.show()

    # Liste les positions des boutons étalons et initialise buttonsLocations
    def setButtonsPosition(self, xmax, ymax):
        """ Liste les positions des boutons étalons et initialise buttonsLocations

        :param xmax: taille de la fenêtre en X
        :param ymax: taille de la fenêtre en Y
        :return:
        """
        side_width = 3
        diam = 100
        ray = diam // 2
        xmilieu = xmax // 2
        ymilieu = ymax // 2
        self.buttonsLocations = [[0, 0], [0, ymilieu - ray], [0, ymax - diam], [xmilieu - ray, 0],
                                 [xmilieu - ray, ymilieu - ray],
                                 [xmilieu - ray, ymax - diam], [xmax - diam, 0], [xmax - diam, ymilieu - ray],
                                 [xmax - diam, ymax - diam]]

    # Affiche les points d'étalonnage à l'écran
```

```
def displayCalibratingPoints(self):
    """ Lance le mode étalonnage et affiche les points

    :return: None
    """
    # Appel d'une fenêtre tk
    self.fenetre = tk.Tk()
    # Mode fullscreen
    self.fenetre.attributes('-fullscreen', True)
    # Echap permet de quitter l'interface
    self.fenetre.bind('<Escape>', lambda e: self.fenetre.destroy())
    # Récupération des informations de l'écran
    width = self.fenetre.winfo_screenwidth()
    height = self.fenetre.winfo_screenheight()
    # création de la fenêtre
    canvas = tk.Canvas(self.fenetre, width=width, height=height, bg='black')
    canvas.pack()
    # initialisation des positions des boutons étalons
    self.setButtonsPosition(width, height)
    # Placement des boutons
    for i in range(len(self.buttonsLocations)):
        button_i = RoundButton(self.fenetre, 100, 100, 50, 0, 'red', "black", self)
        button_i.place(x=self.buttonsLocations[i][0], y=self.buttonsLocations[i][1])
        # Ajout du bouton à la boucle tk
        button_i.pack

    # Lancement du scheduler tk
    self.fenetre.mainloop()

# Affiche la vue seuillée de la caméra
def runThresholdView(self):
    """ Affiche la vue seuillée de la caméra

    :return: None
    """
    # Tant que le seuillage n'est pas terminé
    while not self.isThresholded:
        # Récupération de l'image seuillée
        thresholdView = ImageProcessing.getThresholdedEye(self.getCameraView(), self.eyeThreshold)
        if thresholdView is not None:
            # Affichage de l'image seuillée

            ImageProcessing.showImage(thresholdView, "Etape 1 - Etalonnage du seuil")
            cv2.waitKey(20)

# Affiche l'interface d'étalonnage du seuillage
def displayThresholdCalibration(self):
    """ Affiche l'interface d'étalonnage du seuillage

    :return: None
    """
    # Appel d'une fenêtre tk
    self.fenetre = tk.Tk()
    self.fenetre.geometry("80x120+900+300")
    # Echap permet de quitter l'interface
    self.fenetre.bind('<Escape>', lambda e: self.fenetre.destroy())
    # Récupération des informations de l'écran
    width = self.fenetre.winfo_screenwidth()
    height = self.fenetre.winfo_screenheight()
```

```
        # création de la fenêtre

        # Permet la validation de l'étape de seuillage
        def validateThreshold():
            # On valide l'étape de seuillage
            self.isThresholded = True
            # On supprime toutes les fenêtres
            self.fenetre.destroy()
            cv2.destroyAllWindows()
            # On lance la calibration curseur en affichant les points de calibration
            self.displayCalibratingPoints()

        validate = tk.Button(self.fenetre, text='Valider', command=validateThreshold,
                             background="#494949", foreground="#F90808",
                             relief=tk.GROOVE, height=2, width=9, bd=4.55, font="serif",
activebackground="#6B6B6B")
        validate.place(x=100, y=100)
        validate.grid()

        def updateThreshold(Event):
            self.eyeThreshold = scale.get()

        scale = tk.Scale(self.fenetre, from_=0, to=100, resolution=0.5, orient=tk.HORIZONTAL,
background="#6B6B6B",
                         activebackground="#6B6B6B", foreground="#FFFFFF", highlightbackground="#494949",
                         command=updateThreshold)
        scale.bind("<Leave>", updateThreshold)
        scale.set(self.eyeThreshold)
        scale.place(x=0, y=60)

        # Lancement du scheduler tk
        self.fenetre.mainloop()

    def quitProcess(self):
        """ Permet de quitter le programme proprement

        :return: None
        """
        # On ferme toutes les fenêtre tkinter
        self.fenetre.destroy()
        # Ferme le flux caméra
        self.cap.release()
        # On coupe le programme
        sys.exit(0)

    # Lance le mode étalonnage
    def launchCalibrating(self):
        """ Lance le mode étalonnage

        :return: None
        """
        # On coupe les fenêtres tkInter dans le cas d'un réétalonnage
        if self.fenetre is not None:
            try:
                self.fenetre.destroy()
            except tk.TclError:
                None
        # On montre au programme qu'il est en mode étalonnage
        self.isScaling = True
        # On reset l'état du thresholding
```

```
        self.isThresholded = False
        # On reset les anciennes données de régression
        self.registeredApproximations.clear()

        # On affiche l'étape de seuillage
        self.displayThresholdCalibration()
```

```
import cv2
import numpy as np

"""
ImageProcessing regroupe l'ensemble des fonctions de traitement
de l'image dans la récupération de l'oeil et de sa position
"""


def getPupilVector(lastPos, currentPos):
    """ Renvoie le vecteur déplacement de la pupille entre 2 images

    :param lastPos: dernière position de la pupille
    :param currentPos: position actuelle de la pupille
    :return: None
    """
    return [lastPos[0] - currentPos[0], lastPos[1] - currentPos[1]]


def getPupilPosition(image, thresholdValue=30):
    """ Renvoie la position de la pupille dans le cadre de détection

    :param image: image issue de la caméra
    :param thresholdValue: valeur de seuil de détection de la pupille
    :return: x, y - Position de la pupille | 0,0 si cas d'erreur
    """
    # Récupération de l'image de l'oeil seuillée adaptée
    threshold = getThresholdedEye(image, thresholdValue)

    # attente d'une erreur de format CV2 due à une non détection de l'oeil (mauvais seuillage généralement)
    try:
        # Recherche des contours de l'oeil
        contours = cv2.findContours(np.array(threshold), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
        # Conditions de détection de l'oeil
        if contours is None:
            print("No eye contour detected")
            return [0, 0]
        if len(contours) == 0:
            print("No eye contour detected")
            return [0, 0]

        # Récupération des coordonnées de la pupille
        (x, y, w, h) = cv2.boundingRect(contours[0])

        return x + int(w / 2), y + int(h / 2)

    except cv2.error:
        # Retourne le cas d'erreur
        return 0, 0


def getThresholdedEye(image, thresholdValue):
    """ Permet de seuiller en noir/blanc l'imagette de l'oeil extraite d'une image caméra sur un critère de luminosité

    :param image: image issue de la caméra
    :param thresholdValue: valeur de seuil de détection de la pupille
    :return: threshold - Image de l'oeil sur laquelle est appliquée le seuil | [0, 0] cas d'erreur
    """
    # Extraction de l'imagette de l'oeil supérieur droit détecté
    image = extractEyesPicture(image)
    # Condition de détection
    if image is None:
        print("Eye_color is None")
        return [0, 0]

    # Passage en noir et blanc pour la réduction d'information et une meilleure détection
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Récupération de la taille de l'image
    rows, cols = gray.shape
    # Application d'un flou gaussien
    gray_blurred = cv2.GaussianBlur(gray, (7, 7), 0)
    # Seuillage
    _, threshold = cv2.threshold(gray_blurred, thresholdValue, 255, cv2.THRESH_BINARY_INV)

    return threshold


def extractEyesPicture(image):
    """ Extrait une imagette de l'oeil supérieur droit

    :param image: image type openCV2 issue de la caméra
    :return: extracted - Imagette de l'oeil supérieur droit extrait
    """
    # Méthode des ondelettes de Haar pour la détection des yeux
    eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')

    # Définiton de l'image du visage en couleur
    roi_color, w, h = extractFacesPicture(image);
    # Condition de détection du visage
    if roi_color is None:
        print("roi_color is None")
```

```
        return None
    # Passage en noir et blanc pour la réduction d'information et une meilleure détection
    gray = cv2.cvtColor(roi_color, cv2.COLOR_BGR2GRAY)
    # Détection des yeux
    eyes = eye_cascade.detectMultiScale(gray)

    # Définition de l'image extraite
    extracted = None
    # Bouclage sur les yeux détectés
    for (ex, ey, ew, eh) in eyes:
        # Condition de détection de l'oeil supérieur droit
        if ex < w / 2 - 50 and ey < h / 2 + 50:
            extracted = roi_color[ey:ey + eh, ex:ex + ew]

    return extracted


def extractFacesPicture(image):
    """ Extrait une image englobant le visage de l'utilisateur

    :param image: image type openCV2 issue de la caméra
    :return: extracted - Image englobant le visage de l'utilisateur
    """
    # Méthode des ondelettes de Haar pour la détection de visage
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    # Passage en noir et blanc pour la réduction d'information et une meilleure détection
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Détection des visages
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    # Conditions de détection du visage
    if faces is None:
        print("No face detected")
        return None, None, None
    if len(faces) == 0:
        print("No face detected")
        return None, None, None
    # Extraction des données sur l'image du premier visage détecté
    (x, y, w, h) = faces[0]
    # Extraction du 1er visage détecté sur l'image en couleur
    extracted = image[y:y + h, x:x + w]
    return extracted, w, h


def showImage(image, imageName="noName"):
    """ Permet d'afficher à l'écran une image openCV2

    :param image: image à afficher
    :param imageName: nom de la fenêtre d'affichage
    :return: None
    """
    cv2.imshow(imageName, np.array(image))
    cv2.moveWindow(imageName, 900, 420);
```

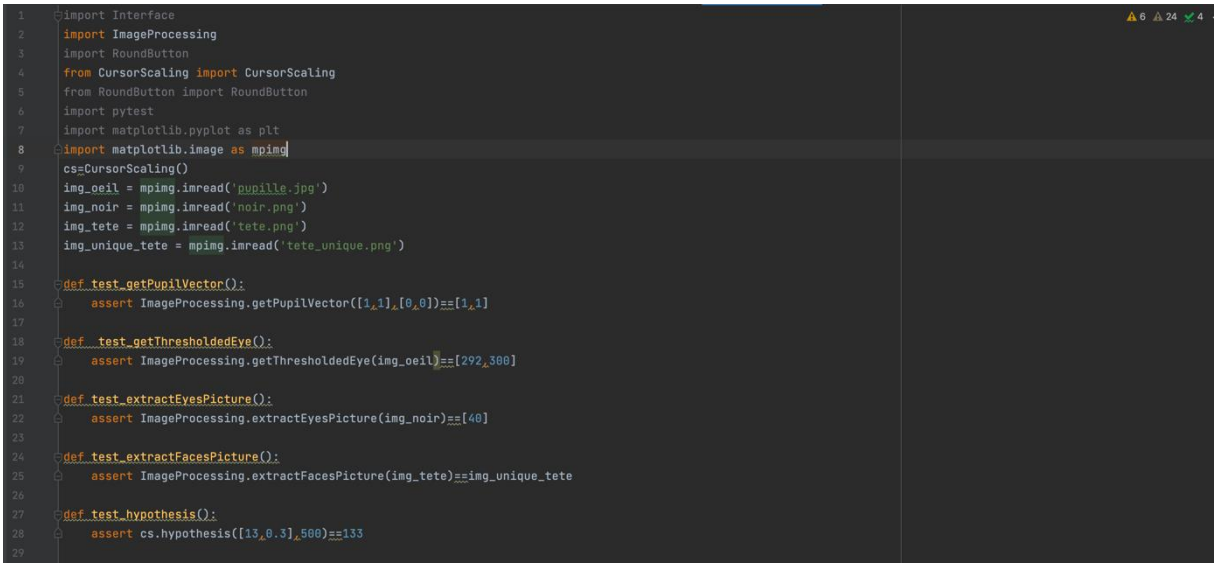
```
from tkinter import *
from RoundButton import *

"""
Interface permet l'affichage de la partie controle final du curseur par l'oeil
"""
```

```
# Affiche les boutons utilisateurs (exit & étalonnage)
def displayButtons(cs):
    """ Affiche les boutons utilisateurs (exit & étalonnage)

    :param cs: instance d'un objet CursorScaling
    :return: None
    """
    # Appel d'une fenêtre tk
    cs.fenetre = Tk()
    # Mode fullscreen
    cs.fenetre.attributes('-fullscreen', True)
    # Echap permet de quitter l'interface
    cs.fenetre.bind('<Escape>', lambda e: cs.fenetre.destroy())
    # Récupération des informations de l'écran
    width = cs.fenetre.winfo_screenwidth()
    height = cs.fenetre.winfo_screenheight()
    # initialisation des positions des boutons étalons
    etan = Button(cs.fenetre, text='étalonnage', command=cs.launchCalibrating,
                  background="#494949", foreground="#F90808",
                  relief=GROOVE, height=2, width=9, bd=4.55, font="serif", activebackground="#6B6B6B")
    etan.place(x=100, y=100)
    etan.grid()
    quitButton = tk.Button(cs.fenetre, text='quitter', command=cs.quitProcess,
                           background="#494949", foreground="#F90808",
                           relief=GROOVE, height=2, width=9, bd=4.55, font="serif", activebackground="#6B6B6B")
    quitButton.place(x=600, y=100)
    quitButton.grid()

    # création de la fenêtre
    canvas = tk.Canvas(cs.fenetre, width=width, height=height, bg='black')
    canvas.grid()
    cs.fenetre.mainloop()
    return
```

```
import tkinter as tk

"""
Class définissant des boutons ronds | Récupération retravaillée de la source :
https://stackoverflow.com/questions/42579927/rounded-button-tkinter-python/45536589
"""


class RoundButton(tk.Canvas):

    def __init__(self, parent, width, height, cornerradius, padding, color, bg, scaler):
        """ Constructeur de RoundButton

        :param parent: canvas parent tkinter
        :param width: largeur du bouton
        :param height: hauteur du bouton
        :param cornerradius: rayon du cercle
        :param padding: remplissage
        :param color: couleur
        :param bg: fond
        :param scaler: instance d'un objet CursorScaling
        """
        tk.Canvas.__init__(self, parent, borderwidth=0,
```

```
                  relief="flat", highlightthickness=0, bg=bg)

    # Le bouton doit pouvoir disparaître dans le scaler, on a donc besoin de l'instance
    self.scaler = scaler

    if cornerradius > 0.5 * width:
        print("Error: cornerradius is greater than width.")
        return None

    if cornerradius > 0.5 * height:
        print("Error: cornerradius is greater than height.")
        return None

    rad = 2 * cornerradius

    def shape():
        self.create_polygon((padding, height - cornerradius - padding, padding, cornerradius + padding,
                             padding + cornerradius, padding, width - padding - cornerradius, padding,
                             width - padding, cornerradius + padding, width - padding,
                             height - cornerradius - padding, width - padding - cornerradius, height - padding,
                             padding + cornerradius, height - padding), fill=color, outline=color)
        self.create_arc((padding, padding + rad, padding + rad, padding), start=90, extent=90, fill=color,
                        outline=color)
        self.create_arc((width - padding - rad, padding, width - padding, padding + rad), start=0, extent=90,
                        fill=color, outline=color)
        self.create_arc((width - padding, height - rad - padding, width - padding - rad, height - padding),
                        start=270, extent=90, fill=color, outline=color)
        self.create_arc((padding, height - padding - rad, padding + rad, height - padding), start=180, extent=90,
                        fill=color, outline=color)

    id = shape()
    (x0, y0, x1, y1) = self.bbox("all")
    width = (x1 - x0)
    height = (y1 - y0)
    self.configure(width=width, height=height)
    self.bind("<ButtonPress-1>", self._on_press)
    self.bind("<ButtonRelease-1>", self._on_release)

def _on_press(self, event):
    """ Action effectuée quand évènement de clic sur le bouton

    :param event: Evenement de clic
    :return: None
    """
    self.configure(relief="sunken")
    if self.scaler is not None:
        self.scaler.pointClickEvent(self)

def _on_release(self, event):
    """ Action effectuée quand relachement du bouton

    :param event: Evenement de relachement
    :return: None
    """
    self.configure(relief="raised")
```

A.2 Code source d’un test unitaire



```
import Interface
import ImageProcessing
import RoundButton
from CursorScaling import CursorScaling
from RoundButton import RoundButton
import pytest
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
cs=CursorScaling()
img_oeil = mpimg.imread('pupille.jpg')
img_noir = mpimg.imread('noir.png')
img_tete = mpimg.imread('tete.png')
img_unique_tete = mpimg.imread('tete_unique.png')

def test_getPupilVector():
    assert ImageProcessing.getPupilVector([1,1],[0,0])==[1,1]

def  test_getThresholdedEye():
    assert ImageProcessing.getThresholdedEye(img_oeil)==[292,300]

def test_extractEyesPicture():
    assert ImageProcessing.extractEyesPicture(img_noir)==[40]

def test_extractFacesPicture():
    assert ImageProcessing.extractFacesPicture(img_tete)==img_unique_tete

def test_hypothesis():
    assert cs.hypothesis([13,0.3],500)==133
```

A.3 Code source d’un test d’intégration

A.4 Code source d’un test de validation

««««@file validation
*
* Test– caméra disponible
*
* vérifier si accès camera
* lancer l’étalonnage avec la caméra
* stocker les coordonnées des yeux
* montrer les graph de la regression
* montrer l’interface pour les mouvements de la souris
*
*
* @version 2
* @author Alexandre CHAUSSARD,Tom SALEMBIEN,Simon CHEREL, Christophe TROALEN, Adam DAHAN
* @date Mai 2021»»»

```
#import <pyautogui>
#import <keyboard>
#import <cv2>
#import <tkinter>
#import <matplotlib>
#import "Cursor Scaling"
#import "ImageProcessing"
#import "RoundButton" #import
"Interface"

#creation d’une fenetre initialisée a partir des classes
RoundButton et Interface
#Pour tester solve si pas encore teste :
solve([0,3,5,1],[8,3,1,2],[13,0,7,1],[10,7,3,9]):


    if (eyeX == NULL) or (eyeY == NULL) {
```

```
      print(" Impossible de faire une regression \n");
      return EXIT_FAILURE;
    }

    # initialisation de la regression à un ordre supérieur :
    # si problème d'accès caméra
        print("Probleme d'accès caméra\n");
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```