Pastori Lucas
Luyckx Matthieu
Verschraegen Gauthier
Ozorai Tom
Debongnie Nathan

Traitement de Signal
Groupe 1
3TI
décembre 2020

Projet Traitement de Signal
Groupe 1

Pastori Lucas
Luyckx Matthieu
Verschraegen Gauthier
Ozorai Tom
Debongnie Nathan

i. Présentation du projet

L'énoncé de notre projet était le suivant : "Comptage automatique du nombre de personnes qui entrent ou sortent d'une zone (une pièce ou un portique ou qui simplement franchissent une ligne)".
Nous avons décidé de prendre la 2ème option, et donc de compter le nombre de gens entrant et sortant d'une zone par rapport à une ligne.

---

ii. Réalisation du projet

Nous avons dans un premier temps réfléchi aux différents éléments nécessaires au bon fonctionnement de notre application. Nous avons donc distingué 4 éléments majeurs :

- La GUI, donc l'interface qui va permettre à l'utilisateur d'utiliser et d'interagir avec l'application.
- La sélection de vidéo et l'importation de celle-ci dans l'application.
- La création de la ligne délimitant l'entrée et la sortie.
- La détection de mouvement, et ainsi, le comptage des personnes.

Nous avons donc scindé le groupe afin de remplir au maximum chacune des tâches individuellement.

Une fois chaque tâche correctement entamée, nous avons rassemblé le travail de chacun et avons ainsi progressé jusqu'à la fin du projet.

Pour réaliser ce projet, nous nous sommes orientés vers le langage *Python* avec le framework *OpenCV 2*. Notre autre alternative aurait été *MATLAB*, mais comme nous ne savions pas encore si nous allions avoir un accès continu au logiciel via l'école au vu de l'enseignement distanciel, nous avons privilégié l'autre option. Nous n'avions, à une exception près, aucune expérience dans ce langage.

---

iii. Explication du projet.

Pastori Lucas
Luyckx Matthieu
Verschraegen Gauthier
Ozorai Tom
Debongnie Nathan

1. GUI

Voici l'aperçu général de la GUI. Elle se compose d'une fenêtre principale avec 3 boutons. Un premier pour sélectionner la vidéo que l'on veut analyser. Deux autres pour le sens de comptage. Le reste se compose de 2 inputs qui contiennent les données de comptage final.

```
def ChangeInput(totalUp, totalDown) :
    InPass.delete(0, END)
    InPass.insert(0, totalUp)
    OutPass.delete(0, END)
    OutPass.insert(0, totalDown)
```

```
InPass = Entry(frameCount, font=('Arial', 20), bg="#4065A4", fg='white')
InPass.pack()

label_outPass = Label(frameCount, text="Nombre de personne(s) qui sorte(nt) : ",
font=('Arial', 15), bg='grey', fg="white")
label_outPass.pack(side=TOP)

OutPass = Entry(frameCount, font=('Arial', 20), bg="#4065A4", fg='white')
OutPass.pack()
```

On a une fonction pour faire passer les valeurs de comptage finales dans les inputs.

2. Choix de la vidéo

Avant de pouvoir tracer une ligne sur une image d'une vidéo, il faut extraire cette image de la vidéo. Pour ce faire, on crée d'abord une fonction permettant de sélectionner une vidéo se trouvant sur l'ordinateur.

```
def FrameCapture():
    file = filedialog.Askopenfilename(
        initialdir="C:",
        title="Choisissez un fichier vidéo",
        filetypes=(("all files", "*")));
```

Par la suite, on récupère la première frame/image de cette vidéo à l'aide de cv2 et on stocke celle-ci dans un dossier image.

```
    vidObj = cv2.VideoCapture(file)
    success, image = vidObj.read()
    cv2.imwrite("image\\frame_%d.jpg" % 0, image)
```

---

3. Sélection de la ligne

i. Mise en place

```
def mouse_drawing(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        # Coordonnées = (x,y)
```

Pour commencer, il a fallu créer une fonction qui puisse traiter les informations lors d'un clic sur l'image, afin de pouvoir tracer une ligne par après qui passe par ce point. Pour cela, 3 lignes suffisent. La première, définissant la fonction n'a rien de très sorcier. Le paramètre "Event" sert à récupérer le type de clic effectué, les paramètres "x" et "y" sont les coordonnées du pointeur de la souris, et les paramètres flags et params ne sont pas utilisés dans ce projet. Ensuite, à la ligne suivante, on vérifie juste

que l'action de la souris est un clic gauche, et lorsque c'est le cas, on récupère les coordonnées x et y, qui sont stockées dans une variable que nous utiliserons plus tard.

```
originalImage = cv2.imread("<firstFrame>")
#cv2.namedWindow("Frame")
cv2.setMouseCallback("Frame", mouse_drawing)
cv2.imshow("Frame", originalImage)
```

Une fois que la fonction "mouse_drawing" est déclarée, nous pouvons passer à l'ouverture de la fenêtre. Pour cela, on utilise la fonction "imread" de cv2 afin de stocker la première image de la vidéo dans une variable. Ensuite, on lie notre fonction de traitement de clic avec notre image à l'aide de la fonction de cv2 appelée "setMouseCallback", qui va envoyer à notre fonction "mouse_drawing" toutes les informations nécessaires au bon traitement, tel que le type de clic, les coordonnées et d'autres paramètres inutiles dans ce cas. Enfin, nous affichons notre première image de la vidéo afin que l'utilisateur puisse en choisir les coordonnées.

Lorsque le programme est lancé, l'utilisateur peut choisir le sens de la ligne et la disposition de l'entrée et de la sortie en fonction du sens. Lorsque la ligne est horizontale, l'entrée peut être soit au-dessus soit en dessous de la ligne, et lorsque celle-ci est verticale, l'entrée est soit à gauche soit à droite. Une fois que le sens de la ligne a été choisi, un simple clic sur l'image définira l'emplacement de la ligne sur l'image. A partir de ce moment-là, le programme se lance et le comptage débute sur base de la vidéo et de la ligne tracée.

```
if sens == “Horizontal”
   cv2.line(frame1, (0,point), (H,point), (0,255,255), 2)
elif sens == “Vertical”
   cv2.line(frame1, (point,0), (point,W), (0,255,255), 2)
```

Pour ce qui est du traçage de la ligne sur la vidéo, nous utilisons la fonction “line” implémentée par cv2. Celle-ci prend en paramètres ces différents points ; En premier, on définit sur quelle image la ligne sera tracée. Ici, ‘frame1’ correspond à l’image actuelle montrée sur la vidéo. Ensuite, les 2 paramètres suivants correspondent aux points de début et fin de la ligne, délimitant le segment tracé. Le paramètre suivant est la couleur sous forme BGR, soit du RGB inversé, ici correspondant au jaune. Enfin, le dernier paramètre correspond à l'épaisseur du trait en pixels, ici fixée à 2, soit assez fin, pour que le trait soit présent mais pas trop épais sur la vidéo.

4. Détection de mouvement

Nous aborderons donc ici tout le développement de la détection de mouvement, nécessaire au comptage de personne.

i. Initialisation

```
cap = cv2.VideoCapture(videofile)
ret, frame1 = cap.read()
ret, frame2 = cap.read()
# ...
while cap.isOpened():
```

A la première ligne, la commande `VideoCapture` nous permet d’initier une capture vidéo d’un fichier vidéo `videofile` donné en paramètre . Nous allons récupérer cette capture dans la variable `cap`. La variable `ret` nous indique par une valeur booléenne si l’opération a bien renvoyé une image. Aux 2 lignes suivantes, nous enregistrons 2 images successives issues de la vidéo, avec la commande `read`, utilisable sur une capture vidéo.

Ensuite, nous rentrons dans une boucle qui perdurera tant que la vidéo n'a pas été entièrement lue. En effet, il s'agit de l'utilité de la commande `isOpened`, applicable sur une capture vidéo.

ii. Différence absolue des 2 images.

```
diff = cv2.absdiff(frame1, frame2)
```

Pour commencer la détection de mouvement, nous allons d'abord chercher la différence absolue entre 2 images.
La différence absolue de 2 images consiste, comme son nom l'indique, à créer une image faite à partir de la différence absolue d'intensité entre chaque pixel de 2 images différentes.
L'intérêt ici est donc de définir quels pixels ont changé d'intensité, et donc, qu'un mouvement ait été effectué.
La commande `absdiff` nous permet de créer cette image en effectuant la différence absolue entre les 2 images fournies en paramètre. L'image résultante de cette opération est stockée dans la variable `diff`.

iii. Conversion en nuance de gris

```
gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
```

Comme il est généralement plus facile de faire du traitement d'image sur une image en nuances de gris plutôt qu'en couleur, nous effectuons ici l'opération de conversion entre ces 2 formats.
La commande `cvtColor` nous permet de transposer notre image source donnée en paramètre (`diff`, la différence absolue entre les 2 images sources) dans une autre couleur. Ici, nous convertissons une image couleur (avec les calques de couleur dans l'ordre respectif bleu, rouge et vert) en une image en nuances de gris, via la méthode de conversion `COLOR_BGR2GRAY`. Cette opération est effectuée avec la formule de la luminance disant :

$$Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Soit, que l'intensité lumineuse totale d'un pixel d'une image est égale à la somme du produit de l'intensité du pixel dans chaque calque de couleur par un coefficient propre à chaque calque.

Nous stockons l'image résultante de cette opération dans la variable `gray`.

iv. Flou Gaussien

```
blur = cv2.GaussianBlur(gray, (5, 5), 0)
```

Afin de faciliter encore la détection de contour, et donc de mouvement, nous allons appliquer un flou Gaussien à notre image.
Le flou Gaussien agit ici comme un filtre passe-bas. Il nous permet de réduire le bruit de notre image, ou en d'autres termes, de réduire les différences élevées d'intensité entre 2 pixels voisins.
Le flou gaussien est en fait une convolution entre une image, donc une matrice de pixel, avec un kernel de convolution approximant une fonction de Gauss. L'objectif de cette fonction est de remplacer la valeur d'un pixel par la moyenne pondérée de son entourage (les plus proches influencent le plus le résultat, et les plus éloignés le moins).
La commande `GaussianBlur` nous permet d'appliquer un flou Gaussien sur une image source (ici `gray`, notre image grisée). Un kernel de convolution gaussien est défini par 2 éléments : ses dimensions, comme tout kernel de convolution, ainsi que sa déviation dans chacun des axes, notée sigma. Cette dernière va définir l'intensité du flou appliqué. Ces 2 paramètres sont définis dans cette commande dans leur ordre respectif, avec une dimension de `5` sur `5`, et une déviation de `0` (la déviation dans chaque dimension sera alors calculée automatiquement avec la formule suivante : **0.3*((ksize-1)*0.5 - 1) + 0.8**).
Nous stockons l'image résultante de cette opération dans la variable `blur`.

v. Binarisation de l'image

```
_, thresh = cv2.threshold(blur, 20, 255, cv2.THRESH_BINARY)
```

Nous allons maintenant chercher à distinguer clairement les contours des mouvements de notre image. Et nous allons accomplir cela en binarisant notre image, donc en convertissant notre image grisée et floutée en une image en noir et blanc.

La commande “threshold” nous permet d’appliquer un seuil à une image et via celui-ci, d’appliquer une règle en fonction de l’intensité de chaque pixel par rapport au seuil. La commande prend plusieurs paramètres :

- `blur` : notre image grisée et floutée
- `20` : la valeur du seuil
- `255` : la valeur à appliquer si l’intensité du pixel respecte une certaine condition vis-à-vis du mode utilisé.
- `THRESH_BINARY` : mode de modification du pixel. Ce mode applique les règle suivantes :
  - Si l’intensité du pixel est plus grande que le seuil, le nouveau pixel prendra la valeur donnée en paramètre, à savoir 255 (et donc, la couleur blanche)
  - Si l'intensité du pixel est plus petite ou égale au seuil, le nouveau pixel prendra la valeur 0 (et donc, la couleur noire).

Nous stockons l’image résultante de cette opération dans la variable `thresh`.

vi. Dilatation de l’image

```
dilated = cv2.dilate(thresh, None, iterations=3)
```

Afin de limiter les trous dans notre image, nous allons ici dilater notre image. L’objectif est donc d’élargir le blanc et de réduire le noir.
La commande `dilate` nous permet d’effectuer cette opération. Ses paramètres sont les suivants :

- `thresh` : l’image source (l’image en noir et blanc).
- `None` : l’élément structurant utilisé (avec cette valeur, l’élément par défaut est choisi. Il s’agit d’un rectangle 3x3).
- `iterations=3` : Nombre de fois que l’opération est effectuée.

Nous stockons l’image résultante de cette opération dans la variable `dilated`.

vii. Recherche des contours

```
contours, _ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Nous allons ici chercher à détecter et stocker tous les contours existants sur l'image. Un contour est défini comme une série de points délimitant un objet et étant de même intensité.
La commande `findContours` va nous permettre de stocker tous ces contours. Les contours sont retrouvés à partir de l'algorithme de Satoshi Suzuki, de l'université de Shizuoka au Japon. Les paramètres de la fonction sont les suivants :

- `contours` : l'image source (l'image en noir et blanc dilatée)
- `RETR_TREE` : mode de récupération des contours. Ce mode nous donnera tous les contours existants, et organisés de manière hiérarchique vis-à-vis des contours inclus dans d'autres.
- `CHAIN_APPROX_SIMPLE` : mode d'approximation du stockage des contours. Ce mode va simplement nous permettre d'économiser énormément de mémoire.
  En effet, grâce à lui, nous ne mémoriserons pas chaque point du contour, mais seulement les extrémités des segments issus de la compression des segments diagonaux/horizontaux et verticaux (exemple : un rectangle serait simplifié par ses 4 coins).

Tous les contours seront stockés dans une variable de type tableau, ici `contours`.

viii. Parcourir les contours

1. *Boucle et détection rectangle*

```
for contour in contours:
   (x, y, w, h) = cv2.boundingRect(contour)
```

Lorsque nous avons récupéré tous les contours de l'image, il faut les parcourir afin de vérifier si la ligne dessinée précédemment a été franchie par une personne. Pour cela nous devons les identifier et les marquer. La première chose est donc de créer une boucle qui parcourt tous les contours de l'image. Ensuite, avec la fonction boundingRect, nous enregistrons les coordonnées d'un rectangle qui est juste assez grand pour contenir la totalité du contour passé en paramètre. La fonction nous renvoie la position (X et Y) du coin en haut à droite du contour, ainsi que la hauteur et la largeur du rectangle.

2. *Taille du contour*

```
if cv2.contourArea(contour) < 900:
        continue
```

Ensuite intervient la fonction contourArea. Cette dernière prend en paramètres le contour et calcule via le théorème de Green la surface du contour. Le résultat est ensuite comparé à une taille seuil (ici 900, mais elle dépend de la vidéo et de la taille moyenne des personnes qui passent). Si le contour est plus petit que le seuil, le programme passe immédiatement au contour suivant.

3. *Affichage du rectangle*

```
cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

Si la condition ci-dessus n'est pas remplie, le programme dessine un rectangle suivant les coordonnées récupérées avec le "boundingRect".
La fonction rectangle de OpenCV reçoit en paramètres l'image sur laquelle le rectangle doit être dessiné, les coordonnées d'un des coins du rectangle à dessiner, les coordonnées du coin opposé, le code pour la couleur du rectangle (R,G,B) ainsi que l'épaisseur du rectangle.

4. *Récupération du centre du contour*

```
M = cv2.moments(contour)
cX = int(M["m10"] / M["m00"])
cY = int(M["m01"] / M["m00"])
```

La première ligne ci-dessus permet de récupérer la moyenne pondérée des positions des pixels du contour qu'il est en train de lire. Grâce à cela, il est possible de retrouver le point central du contour grâce à la formule $\bar{x} = \frac{m_{10}}{m_{00}}, \bar{y} = \frac{m_{01}}{m_{00}}$.

5. *Modification du compteur*

```
if sens == "V":
   if testX is not None and point + 2 > cX > point - 2:
       if cX < testX:
           totalUp += 1
           ChangeInput(totalUp, totalDown, entree)
```

```
        elif cX > testX:
            totalDown += 1
            ChangeInput(totalUp, totalDown, entree)
```

Afin de pouvoir vérifier si une personne traverse une ligne, nous avons besoin de la position et l'orientation de la ligne, ainsi que de la position actuelle et précédente de la personne. Le code ci-dessus permet de compter les personnes lorsque la ligne est verticale sur l'image. Le sens passé en paramètres correspond alors à "V" ("H" dans le cas où celle-ci est à l'horizontale).
Ensuite, si les coordonnées de la personne à l'image précédente (testX) ne sont pas inexistantes et que les coordonnées actuelles du centre de la personne (cX) sont sur la ligne (avec une marge de 2 pixels de part et d'autre de la ligne), il compare la position actuelle avec celle de l'image précédente. Le repère se trouvant en haut à gauche, si la position actuelle est plus grande que la précédente, la personne se dirige vers la droite. Sinon, elle se dirige vers la gauche. Il y a lors de chaque détection de personne également un appel à une fonction de la GUI. Cette fonction "ChangeInput" permet de mettre à jour les compteurs de la GUI.

6. *Affichage du centre et préparation pour la frame suivante*

```
cv2.circle(frame1, (cX, cY), 2, (255, 255, 255), -1)

testX = cX
testY = cY
```

Pour que la vidéo soit claire, nous ajoutons un cercle au centre de la personne. Cela se fait avec la fonction "circle" du module OpenCV. La fonction prend en paramètres l'image sur laquelle il faut ajouter le point, les coordonnées de celui-ci, le rayon du cercle à dessiner, la couleur de celui-ci ainsi que l'épaisseur du bord (-1 pour remplir l'entièreté du cercle). Ensuite nous récupérons les positions cX et cY pour les réinjecter dans la variable des coordonnées centrales du contour précédent testX et testY.

1. Affichage du compteur sur la vidéo

```
if entree == "up" or entree == "right":
   cv2.putText(frame1, "Entrees: {}".format(totalUp), (10, 20),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 3)
   cv2.putText(frame1, "Sorties: {}".format(totalDown), (10, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 3)
```

Nous ajoutons également un affichage du compteur sur la vidéo. Pour cela, nous devons vérifier dans quel sens les personnes doivent être comptées. Le compteur s'affiche en haut à droite, à l'aide de la fonction putText de OpenCV.

```
if sens == "H":
   cv2.line(frame1, (0, point), (H, point), (0, 255, 255), 2)
else:
   cv2.line(frame1, (point, 0), (point, W), (0, 255, 255), 2)
```

Il faut ensuite dessiner la ligne en fonction de l'endroit et l'orientation demandés par l'utilisateur.

```
image = cv2.resize(frame1, (1280, 720))
out.write(image)
cv2.imshow("Video", frame1)
frame1 = frame2
ret, frame2 = cap.read()
```

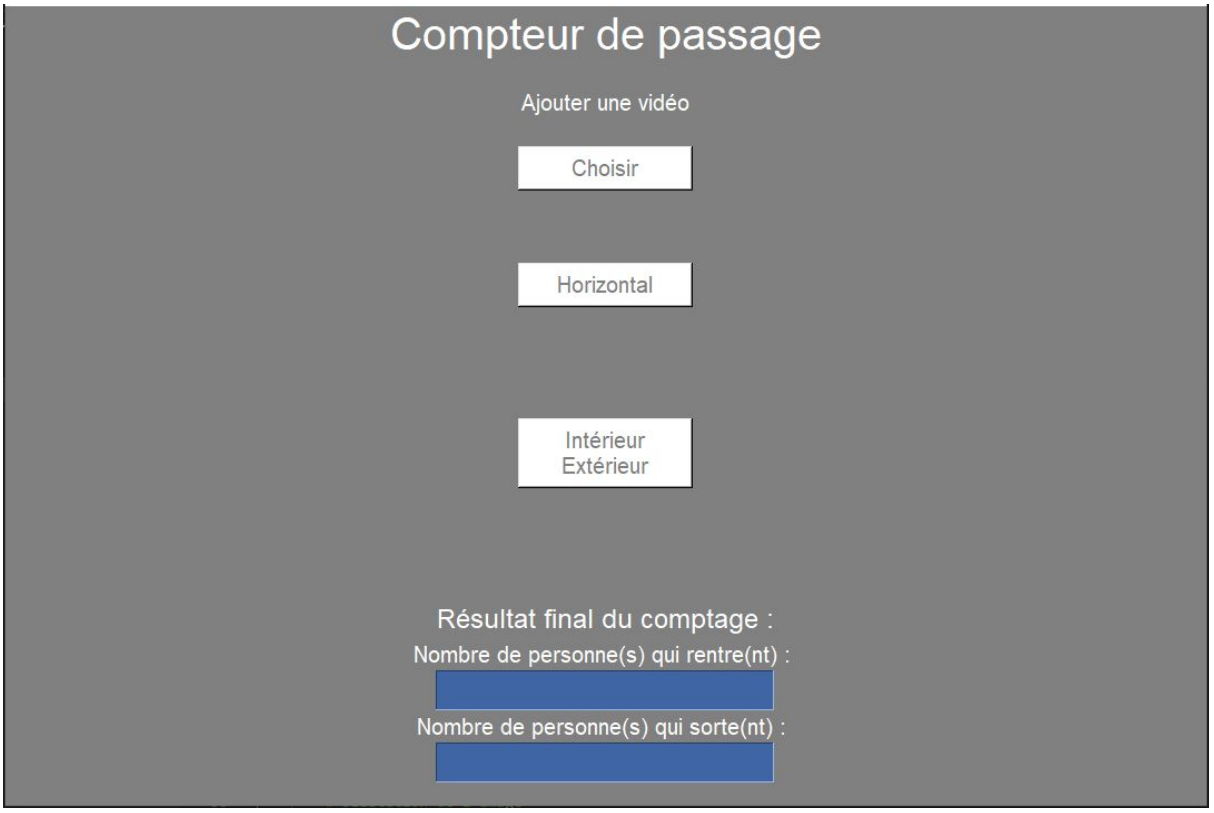
Avant de passer à la frame suivante, la frame actuelle est envoyée vers la vidéo de sortie. Elle est également affichée dans la fenêtre avant d'être modifiée par la valeur de la frame suivante. La valeur de la frame suivante est ensuite changée par la lecture de la frame qui suit dans la vidéo.

5. Exécution du programme

Au préalable, il faudra d'abord que vous ayez accès au package *opencv-python.* Nous l'avons obtenu via l'utilitaire *pip*, mais la librairie est

disponible, mais vous pouvez voir toutes les méthodes d'installation sur la documentation OpenCV.
Une fois le package à disposition, vous pourrez lancer la commande suivante : **python GUI.py**

Une fenêtre comme celle indiquée ci-dessous devrait alors apparaître.



Vous pourrez alors appuyez sur les boutons dans l'ordre suivant :

A. "Choisir" : Choisir une vidéo parmi les fichiers de votre ordinateur. Notre projet contient 2 vidéos de test, une pour une limite verticale et une pour une limite horizontale.
B. "Intérieur/Extérieur" : Défini où seront l'entrée et la sortie par rapport à la limite
C. "Horizontal/Vertical" : La limite tracée sur l'image sera obligatoirement une droite verticale ou horizontale. Vous pouvez ici choisir quel sens adopter.

Une fois le A. effectué , la première frame de votre vidéo vous sera affichée. Le prochain clic sur cette image tracera la ligne selon la règle choisie au C. Le traitement de la vidéo se lancera automatiquement après avec comme règle de comptage celle choisie au B.

Une fois le traitement terminé, la vidéo traitée se ferme, et vous verrez sur l'interface initiale le nombre de personnes compté dans chaque sens.

---

iv. Conclusions

Groupe

Globalement, nous avons tous bien aimé apprendre/approfondir le python. N'ayant dans l'ensemble que très peu touché à ce langage, son apprentissage a été plutôt chouette et passionnant à faire. De plus, le fait d'appliquer des éléments théoriques axés sur le traitement d'images vus au cours de traitement de signaux a été plaisant, car nous mettons ces connaissances dans un cas concret d'utilisation. Ce genre d'application est vraiment motivante dans le sens où toute la théorie apprise auparavant n'est plus "juste" de la simple connaissance à nos yeux, ça devient un outil potentiel pour de futurs projets.

Personnelles

Tom : Un projet dans l'ensemble très intéressant. C'est bien de vraiment pouvoir mettre en pratique ce que l'on a vu en cours. L'utilisation de Python apportait également une nouveauté et la prise en main de ce langage que je connaissais très peu a été plutôt rapide. J'ai eu l'occasion de faire l'interface graphique et également de participer à la conception de la reconnaissance visuelle grâce à OpenCV. C'était un projet très concret car il me donne une petite base pour mon stage et mon TFE qui se base là dessus, avec en plus du machine Learning.

Matthieu : De mon côté, ce projet m'a énormément plu. Réalisant un projet concret sur base d'un sujet de la vie courante, ça apporte beaucoup plus de motivation par rapport à un enchaînement de commandes lors des laboratoires uniquement dans le but d'apprendre celles-ci. Ici, appliquer tout ça dans le but de réaliser un projet réaliste est une plus value considérable à la motivation et au plaisir d'apprendre.
L'apprentissage du python a également été passionnant. N'ayant jamais touché à ce langage auparavant, ce projet m'y a initié et ça m'apporte un brin de culture informatique en plus, ce qui ne me déplait absolument pas.

Lucas : Projet très intéressant dans l'ensemble. Découverte et apprentissage de python plutôt plaisant même si ce n'était pas le but principal du projet. De plus, le fait de mettre en pratique ce qui a été vu en cours m'aide beaucoup à mieux comprendre le cours dans l'ensemble. J'ai principalement travaillé sur la récupération de la première image de la vidéo et sur la sélection de celle-ci. Cela ne m'a pas empêché de me pencher souvent sur la partie concernant la détection de personnes que ce soit pour aider ou pour mieux comprendre celle-ci.

Nathan : Grâce au cours de 2eme sur Python, j'avais déjà les bases de ce langage. OpenCV, le module utilisé pour le traitement d'images, m'était cependant inconnu. La compréhension n'a pas été trop compliquée étant donné que les fonctions du module Python ne diffèrent pas tant que cela de Matlab. De plus, comme tout projet, cela m'a permis de voir le cours en application, ce qui aide beaucoup pour la compréhension.

Gauthier : La réalisation de ce projet et sa compréhension ont vraiment été intéressantes. Je me suis surtout penché sur cette dernière et cela m'a vraiment plu de décortiquer le code et de comprendre étape par étape les opérations effectuées. J'ai toutefois un petit ressenti d'inachevé, surement dû au fait d'avoir dû "apprendre à faire le projet" et pas mettre en pratique des savoirs pratiques vu au cours. J'ai aussi trouvé dommage de n'avoir eu le point de matière du cours sur notre sujet que vers la fin du trimestre, car je pense que ça nous aurait beaucoup facilité la tâche dans notre approche du projet.
J'ai globalement apprécié cette immersion dans ce domaine et je suis satisfait de l'expérience que cela m'a donné.

Alternatives

- Matlab vs Python : Dans le cadre de ce projet, deux solutions s'offraient à nous. Soit nous utilisons Matlab, comme pour les travaux pratiques, soit Python. Nous avons opté pour ce dernier pour une raison principale. Etant donné que Matlab est payant et que nous n'étions donc pas certains d'avoir accès au logiciel.

- Flou Gaussien vs Mean : La différence principale entre “mean” et le flou Gaussien est le fait que le flou gaussien fait une moyenne pondérée des pixels aux alentours. Il y a donc une notion de “poids” qui définit l’impacte du pixel sur le pixel traité. Mean quant à lui garde un”poids” équivalent peu importe la distance.
- Application filtre de Sobel : Ce filtre aurait pu nous être utile dans la détection de contour. Comme nous avions déjà de bons résultats sans celui-ci, nous avons décidé de ne pas l’utiliser.

Pistes d’améliorations

- Pouvoir compter le passage également sur base de lignes obliques.
- La ligne pourrait être tracée de telle manière à uniquement prendre les dimensions de la porte et non l’ensemble de l’image sur la vidéo.
- Réaliser un algorithme plus performant pour encadrer avec précision une personne.
- Privilégier des algorithmes de tracking de personnes (Machine/Deep Learning : algorithme YOLO ou deep SORT).
- Pouvoir ajuster la sensibilité à la vidéo à la taille des éléments à détecter sur la vidéo.

D. Bibliographie

- GUI : https://www.youtube.com/watch?v=N4M4W7JPOL4
- Gestion des “Mouse event” pour récupérer des coordonnées : https://www.youtube.com/watch?v=H_068uiMR9M
- Extraction de la première image d’une vidéo : https://www.geeksforgeeks.org/python-program-extract-frames-using-opencv/
- Traçage de la ligne sur la vidéo : https://www.geeksforgeeks.org/python-opencv-cv2-line-method/
- Sélection d’un document du pc : https://www.youtube.com/watch?v=Aim_7fC-inw
- Documentation OpenCV: https://docs.opencv.org/4.5.0/index.html

- Video comptage de personnes: https://www.youtube.com/watch?v=MkcUgPhOIP8&fbclid=IwAR1E7WWwiIZ6X7mK8GpEiqtQss6Tz6IsUFfPN0rjJMPCSGdbGPrfQcWRi6I
- Détection de mouvement en machine learning : https://medium.com/analytics-vidhya/yolo-v3-real-time-object-tracking-with-deep-sort-4cb1294c127f
- Différence filtre Gaussien et “mean” : https://stackoverflow.com/questions/31131672/difference-between-mean-and-gaussian-filter-in-result
- Fonction gaussienne : https://fr.wikipedia.org/wiki/Fonction_gaussienne
- Flou gaussien : http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm
- Flou gaussien : https://www.geo.fr/voyage/photographie-conseils-pratiques-flou-gaussien-169306
- Contours en Python : https://www.geeksforgeeks.org/find-and-draw-contours-using-opencv-python/?ref=lbp