Guide visibilité par SDoG

Les librairies nécessaires pour faire fonctionner ce programme sont :

- **Numpy** : pour effectuer les calculs sur les images
- **OpenCV** : pour lire les images et effectuer certaines opérations
- **Matplotlib** : pour afficher les résultats sous forme de grille d'images ou de graphique

Les différents fichiers du programme sont :

Parameters.py

Regroupe les différents paramètres du programme : la taille et la résolution de l'écran, les σ et les poids nécessaires pour la construction des SDoG

ConvolutionFilter.py

Permet de générer un filtre de convolution avec la classe **Filter**, ainsi que les dérivées horizontales et verticales du filtre

Paramètres d'initialisation :

distance_from_screen (**obligatoire**): permet de régler la taille du filtre. Les images sinusoïdales sur lequel le filtre est appliqué doivent se situer à la même distance afin que le filtre soit efficace

sigma_list : représente la liste des σ pour la construction des différentes DoG. Les sigmas doivent être en *cpd*

weight_list : donne le poids de chacune des DoG, dans le même ordre que les σ sont définies. Si cette liste n'est pas donnée, alors les DoG sont initialisées avec un poids de 1

frequency : si la liste des σ n'est pas donnée, construit une DoG permettant d'isoler la fréquence en question

filter_size : permet de définir la taille du filtre. Si ce paramètre n'est pas renseigné (**recommandé**), le filtre sera compris entre -3λσ et 3λσ et aura donc une efficacité de 99.73%

Attributs :

filter_size : représente la taille du filtre

filter_array : représente les valeurs du filtre 2D sous forme de *numpy array*

filter_dx : représente la dérivée horizontale du filtre 2D sous forme de *numpy array*

filter_dy : représente la dérivée verticale du filtre 2D sous forme de *numpy array*

ImageAnalyzer.py

Permet de créer un analyseur d'images lié à un opérateur SDoG. Cet opérateur pourra ensuite être appliqué sur des images ce qui permettra de récupérer plusieurs informations suite à l'application de ce filtre

Paramètres d'initialisation :

convolution_filter : filtre de convolution (SDoG) qui sera appliqué sur nos images

Attributs :

image_array : image brute sur laquelle notre opérateur a été appliqué

filtered_img : image filtrée par notre opérateur

gradient_filtered_img : gradient de notre image filtrée, déterminée grâce aux dérivées horizontale et verticale de notre filtre

edge_localisation : image représentant la localisation de nos contours, un pixel a une valeur de 1 si c'est un contour, sinon il vaut 0

visibility_map : carte de visibilité de notre image, en combinant la localisation de nos contours et le gradient de notre image filtrée

pixel_size_degree : taille d'un pixel en degré, afin que la distance entre les pixels de notre image varie selon la distance à laquelle on regarde l'image et soit cohérente avec notre filtre

mean_intensity : intensité moyenne de l'image

mean_visibility : visibilité moyenne de l'image (déterminées à partir de la carte de visibilité), en excluant les bords de celle-ci

Fonctions :

save_image(file_name) : Sauvegarde le gradient de l'image filtrée, la localisation des contours et la carte de visibilité dans des fichiers PNG avec le nom donné en paramètre

save_visibility_map(file_name) : Sauvegarde uniquement la carte de visibilité dans un fichier PNG

generate_visibility_map (img, method) : permet de déterminer la visibilité d'une image. La méthode permet de définir comment on détermine la localisation des contours :

- **ZERO_CROSSING** utilise notre image filtrée par notre SDoG pour trouver les passages à zéros, qui seront classifiées en tant que contours
- **GRADIENT** utilise le gradient de l'image pour trouver les extrema locaux et les répertorier en tant que contour

Cette fonction applique d'abord un pre-processing à l'image pour ensuite déterminer le gradient de l'image et la localisation des contours afin de déterminer la carte de visibilité de l'image

ImageGenerator.py

Permet de générer ou de charger depuis un fichier une image et la stocker dans la classe

Paramètres d'initialisation :

file_path : chemin du fichier à charger

Attributs :

image_array : représentation de notre image sous forme de *numpy array*

mean_intensity : intensité moyenne de notre image

Fonctions :

load_image(file_path) : permet de charger une image depuis un fichier image

load_txt(file_path) : permet de charger une image depuis un fichier *txt* (avec un formatage particulier)

convert_into_linear_space() : permet de convertir l'image de l'espace gamma vers l'espace linéaire

convert_into_gamma_space() : permet de convertir l'image de l'espace linéaire vers l'espace gamma

generate_[sine]/[square]_[wave]/[circle]_img(img_width_pxl, img_width_degree, frequency, period_pxl, img_ratio, distance_from_screen) : génère une image representant la fonction sinusoïdale ou la fonction carré, en forme de vague ou en forme de cercles. Les différents paramètres sont :

- La taille de l'image, que l'on peut régler en pixel avec *img_width_pixel* ou en degré avec *img_width_degree*
- La fréquence, déterminée avec *frequency* (en cpd) ou avec la période en pixel avec *period_pixel*
- Le ratio de l'image avec *img_ratio*
- La distance à laquelle on regarde l'image *distance_from_screen* afin d'avoir une conversion pour la fréquence de cpd vers pixels

tune_image_contrast_michelson(mean_luminance, contrast) : permet de modifier l'image en définissant son contraste de Michelson

generate_[aa_]target_img(img_width_pxl, angular_size, luminance_background, luminance_target, distance_from_screen) : génère une image de taille *img_width_pxl* avec en son centre une cible de de taille angulaire *angular_size* vu d'une distance *distance_form_screen* avec une luminance *luminance_target* sur un fond de luminance *luminance_background*. En rajoutant *aa* dans la fonction, cela rajoute un effet d'anti-aliasing sur la génération de la cible

save_image(file_name) : permet de sauvegarder l'image dans un fichier avec **OpenCV**

PltGraph.py

Permet de générer un graphique **matplotlib** et d'y ajouter différentes courbes pour ensuite les afficher

Paramètres d'initialisation :

title : titre du graphique

x_label : nom sur l'axe des abscisses

y_label : nom sur l'axe des ordonnées

Fonctions :

set_x_label(x_label) : définit le nom sur l'axe des abscisses

set_y_label(y_label) : définit le nom sur l'axe des ordonnées

set_log_scale(axis) : permet de définir l'axe des abscisses et/ou l'axe des ordonnées en échelle logarithmiques. *Axis* est un **string** et ses 3 valeurs possibles sont :

- *"x"* : si on souhaite appliquer une échelle logarithmique sur l'axe des abscisses
- *"y"* : si on souhaite appliquer une échelle logarithmique sur l'axe des ordonnées
- *"xy"* ou *"yx"* : si on souhaite appliquer une échelle logarithmique sur nos deux axes

set_y_lim(y_min, y_max) : permet de définir une limite sur l'axe des ordonnées afin d'avoir un graphique dans une range convenable pour la lecture (majoritairement lorsqu'on a une échelle logarithmique sur l'axe des ordonnées)

convert_to_adrian_space() : permet de convertir toutes les courbes du graphique, les abscisses sont converties de fréquence en cpd vers taille angulaire en minutes et les ordonnées de sensibilité au contraste en luminance seuil. Cela permet ainsi de retrouver les mêmes unités que la courbe d'Adrian

convert_to_csf_space() : permet de convertir toutes les courbes du graphique, les abscisses sont converties de taille angulaire en minutes vers fréquence en cpd et les ordonnées de luminance seuil en sensibilité au contraste. Cela permet ainsi de retrouver les mêmes unités que la CSF

add_curve(x, y, title, curve_type, color, marker) : permet de rajouter une courbe au graphique :

- *x représente* les abscisses de la courbe
- *y* représente les ordonnées de la courbe
- *curve_type* représente le type de courbe (**CURVE** pour un tracé continue, et **POINTS** pour un tracé discret avec une série de points)
- *color* peremt de donner une couleur à la courbe (le nom doit être en anglais)
- *marker* représente la façon dont sont représentés nos points, en suivant les marqueurs disponibles dans **matplotlib** (*"."* , *"o", "x"* par exemple)

show() : permet d'afficher le graphique

PltImages.py

Permet de générer une grille d'images **matplotlib** et d'y ajouter différentes images pour ensuite les afficher

Paramètres d'initialisation :

rows : nombre de ligne dans la grille d'image

columns : nombre de colonne dans la grille d'images

Fonctions :

add_image(image, title, cmap, row, column, vmin, vmax, color_bar) : permet d'ajouter une image à la grille de l'image :

- *image* est l'array qui représente l'image
- *title* est le titre de l'image dans la grille
- *cmap* est la représentation de couleur de l'image qu'on souhaite affichée, selon les représentations de **matplotlib** (*"gray"*, *"hot"* par exmple)
- *row* est la ligne sur laquelle on souhaite placer l'image
- *column* est la colonne sur laquelle on souhaite placer l'image
- *vmin* est la valeur minimale à afficher dans notre représentation. Toutes les valeurs inférieures à cette valeur auront la même couleur
- *vmax* est la valeur maximale à afficher dans notre représentation. Toutes les valeurs supérieures à cette valeur auront la même couleur
- *color_bar* permet de préciser si on veut afficher l'échelle des couleurs à côté de la figure

show() : permet d'afficher la grille d'image

Adrian.py

Regroupe les fonctions liées à l'article d'Adrian (*Adrian, W. (1989). Visibility of targets: model for calculation. Lighting Research & Technology, 21(4):181–188*)

Fonctions :

get_luminance_target_threshold(alpha, luminance_background, age, exposure_time) : permet de retourner le Δ de la luminance seuil de la cible par rapport à la luminance de fond pour qu'elle soit visible, avec les paramètres suivants :

- *angular_size* : taille angulaire de la cible

- *luminance_background* : luminance de fond
- *age* : age du spectateur
- *exposure_time* : temps d'exposition de la cible

get_visibility(angular_size, luminance_target, luminance_bg, delta_luminance) : permet de retournée la visibilité d'une cible de taille angulaire *angular_size* et de luminance *luminance_target* sur un fond qui a comme luminance *luminance_bg*. On peut également préciser le Δ de la luminance seuil si on l'a calculé précédemment, sinon il sera déterminé pour la luminance de fond donné avec un âge de 23 ans (âge minimale) et une durée d'exposition de 2 secondes (durée maximale)

BartenCSF.py

Regroupe les fonctions liées à la CSF de Barten

Fonctions :

get_barten_csf(freq, mean_luminance, stimulus) : Calcule l'inverse du contraste seuil pour une fréquence *freq*, une luminance moyenne *mean_luminance* ainsi que la taille du *stimulus*. La fréquence donnée peut soit être un **float**, soit un array de **float**, et retournera en conséquence un **float** ou un array de **float**

get_barten_csf_squared(freq, mean_luminance, stimulus, threshold) : Calcule l'inverse du contraste seuil pour une fonction carré de fréquence principale *freq*, une luminance moyenne *mean_luminance* ainsi que la taille du *stimulus*. La fréquence donnée peut soit être un **float**, soit un **array de float**, et retournera en conséquence un **float** ou un **array de float**. La fonction va alors calculer pour chaque fréquence donnée la CSF de ses différentes harmoniques, jusqu'à ce que la valeur de la CSF pour cette harmonique soit inférieure à un certain seuil *threshold*.

SDoG.py

Regroupe les fonctions liées à la génération de nos DoG, de leur poids pour notre SDoG

Fonctions :

get_fitting_sigma(frequencies) : retourne les σ optimaux permettant d'isoler les fréquences *frequencies* (peut être soit un **float** soit un **array de float** et renvoie autant de σ qu'il y a de fréquences en paramètres)

get_fitting_weight(frequencies, function_to_mimic) : retourne les poids optimaux afin de faire correspondre les DoG optimales (pour fréquences *frequencies*), à la fonction donnée en paramètre *function_to_mimic* (pour ces mêmes fréquences). Le paramètre *frequencies* peut être soit un **float** soit un **array de float** et la fonction renvoie autant de poids qu'il y a de fréquences en paramètres

get_weight_regression(frequency_range, function_to_mimic, dog_count) : effectue une régression linéaire avec une fonction d'erreurs afin de trouver les σ ainsi que les poids permettant d'obtenir une SDoG la plus proche possible de la fonction qu'on cherche à reproduire *function_to_mimic* sur l'intervalle de *frequency_range*. On peut préciser le nombre de DoG qu'on souhaite créer, de base ce nombre étant à 6. Cette fonction renvoi une liste de σ et de poids

ConversionFunction.py

Regroupe les différentes fonctions de conversion d'unités, notamment sur le rapport entre degrés et pixels et les différentes définitions du contraste.

Fonctions :

convert_minutes_to_frequency_cpd(angular_size) : Convertit une taille angulaire en minutes d'arc en une fréquence spatiale en cycles par degré (cpd)

convert_frequency_cpd_to_minutes(frequency) : Convertit une fréquence spatiale (cpd) en une taille angulaire en minutes d'arc

convert_minutes_to_pixels(angular_size, distance_from_screen) : Convertit une taille angulaire en minutes d'arc en pixels, en fonction de la distance à l'écran

convert_angular_size_in_pixels(angular_size, distance_from_screen) : Convertit une taille angulaire en degrés en pixels, en fonction de la distance à l'écran

convert_pixels_in_angular_size(pixel_width, distance_from_screen) : Convertit une largeur en pixels en une taille angulaire en degrés

convert_frequency_cpd_in_pixel_period(frequency_cpd, distance_from_screen) : Convertit une fréquence spatiale en cycles par degré en période en pixels, selon la distance à l'écran

convert_pixel_period_in_frequency_cpd(pixel_period, distance_from_screen) : Convertit une période en pixels en fréquence spatiale en cycles par degré

get_michelson_contrast(luminance_target, luminance_background) : Calcule le contraste de Michelson d'une cible par rapport à son fond

get_target_luminance(luminance_background, contrast) : Calcule la luminance d’une cible à partir du contraste de Michelson et de la luminance de fond

convert_visibility_to_luminance_threshold(visibility, luminance_background, luminance_target) : Calcule la luminance seuil de visibilité d’une cible lorsqu’on a réussi à calculer la visibilité d’une cible en connaissant sa luminance et la luminance de fond

convert_to_adrian_space(frequencies, sensitivity, luminance_background) : Permet de convertir une liste de fréquence et leur sensibilité au contraste associée pour une luminance de fond, en taille angulaire et en seuil de luminance. Cette fonction renvoi donc deux listes.

convert_to_csf_space(size_list, luminance_threshold, luminance_background) : Permet de convertir une liste de taille angulaire et leur luminance seuil associée pour une luminance de fond en liste de fréquences et en sensibilité au contraste. Cette fonction renvoi donc deux listes.