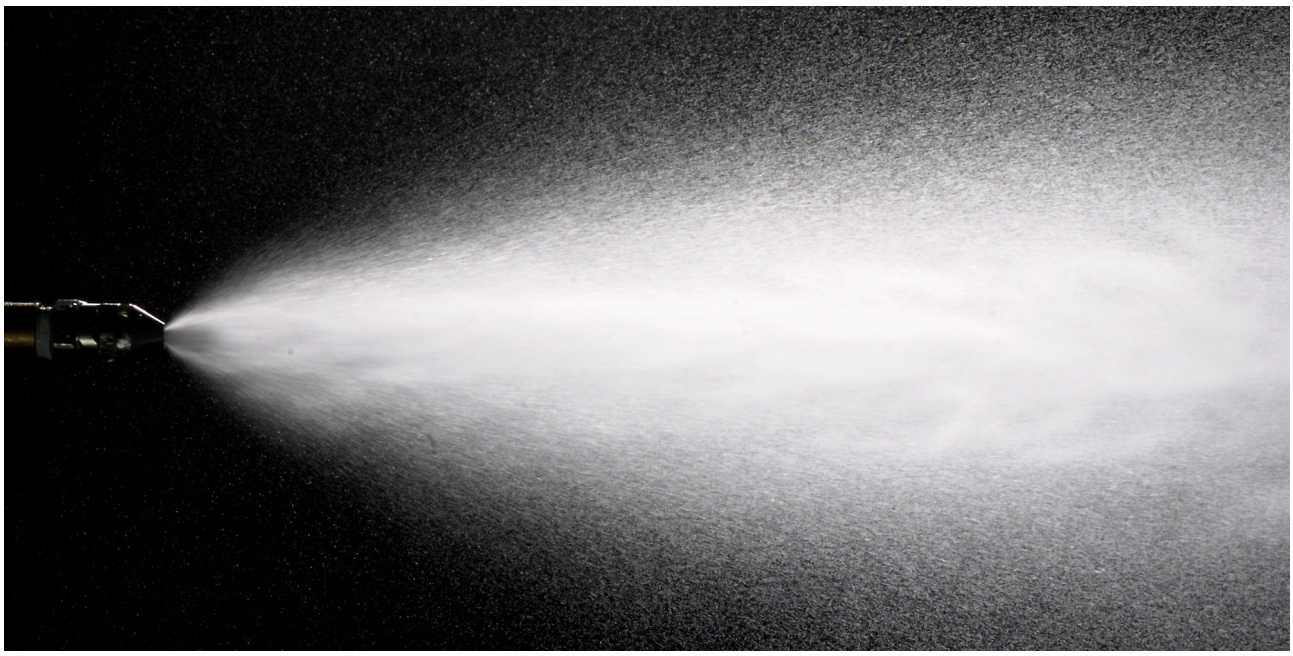
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

Projet de Physique P6
STPI/P6/2023 - 45

Angle d'ouverture d'un spray : la programmation appliquée à la méca-flu par le traitement d'images



Étudiants :
Tom Poget
Babatunde Affolabi
Lola Poupinel
Malak Mrini

Enseignant-responsable du projet :
Téodor Chazelle

Date de remise du rapport : 17/06/2023

Référence du projet : STPI/P6/2023 – 45

Intitulé du projet : Angle d'ouverture d'un spray : la programmation appliquée à la méca-flu par le traitement d'images

Type de projet : bibliographie - modélisation

Objectifs du projet :
— Renseignement sur l'historique du spray et son fonctionnement
— Initiation aux différentes méthodes de traitement d'images
— Approfondissement des acquis en Python
— Calcul de l'angle d'ouverture d'un spray grâce au traitement d'images

Mots-clés du projet : spray, angle d'ouverture, traitement d'images

Institut National des Sciences Appliquées de Rouen
Département Sciences et Techniques Pour l'Ingénieur
685 Avenue de l'Université BP 08 - 76801 Saint-Étienne-du-Rouvray
Tél : +33 2 32 95 66 21 - Fax : +33 2 32 95 66 31

Table des matières

Introduction

En équipe de quatre, nous avons travaillé tout au long du semestre sur ce projet intitulé «*Angle d'ouverture d'un spray : la programmation appliquée à la méca-flu par le traitement d'images*», projet durant lequel nous avons pu approfondir nos connaissances et renforcer nos acquis en informatique et notamment en traitement d'images. Nous avons pu apprendre à mieux manipuler le language de programmation Python pour le processus de traitement d'images. Un important travail bibliographique a été également fait sur l'historique des spray, leur fonctionnement, leurs différentes utilisations... ainsi que sur le processus de traitement d'images.

L'objectif de ce projet étant de parvenir à **déterminer l'angle d'ouverture d'un spray**, nous avons d'abord commencé par traiter un jeu d'images fourni par notre encadrant, tout en adaptant notre code à tout jeu d'images qu'on peut rencontrer afin de rendre notre méthode efficace à grande échelle.

Organisation du travail

Afin de mener à bien ce projet, nous nous sommes répartis les tâches ainsi : Malak s'est occupée du travail documentaire concernant les spray, Lola s'est quant à elle chargée de la recherche portant sur les images, les processus permettant de les traiter ainsi que du poster. Tom et Babatunde se sont occupés de la partie informatique du projet qui consiste à élaborer un code permettant de traiter des jeux d'images de spray afin de pouvoir déterminer précisément l'angle de celui-ci.
La rédaction du rapport a quant à elle été faite de façon collaborative sur la plateforme Overleaf afin d'assurer la cohérence et la cohésion de celui ci.

Le tableau ci-dessous récapitule les différentes tâches réalisées par les différents membres du groupe :

Lola Poupinel	**Malak Mrini**
bibliographie : images et traitement	bibliographie : sprays
Babatunde Affolabi	**Tom Poget**
normalisation, binarisation, cropping automatique, régression linéaire, calcul de l'angle	normalisation, chargement d'images, cropping manuel, régression linéaire, calcul de l'angle

Chapitre 1

Images et traitement

1.1 D'abord, qu'est ce qu'une image ?

Le terme "image" est utilisé dans énormément de domaines avec plusieurs signifiations possibles.

Nous nous intéressons ici aux images dites numériques. C'est-à dire qu'elles sont soit traitées, soit stockées, soit crées sous forme binaire. La raison étant que le traitement se fait en grande partie informatiquement et donc que les ordinateurs ne peuvent fonctionner qu'avec ce genre de données.

Afin de pouvoir manipuler une image, il faut avant tout savoir comment la représenter.

1.1.1 Les différents types d'images[12]

Il existe deux types d'image numériques : les images vectorielles et les images matricielles. Pour ce projet, nous nous intéresserons exclusivement aux images matricielles. Les images vectorielles représentent des données pouvant être aisément décrites mathématiquement. Par exemple, un cercle, dont les composantes sont le diamètre et la position de son centre. Ces images sont utilisées pour des plans ou des schémas. Les images matricielles sont quant à elles composées de pixels. Ces images peuvent être vues comme un tableau à double entrée où chaque croisement correspond à un pixel. Chaque pixel stocke des informations, comme par exemple une couleur.

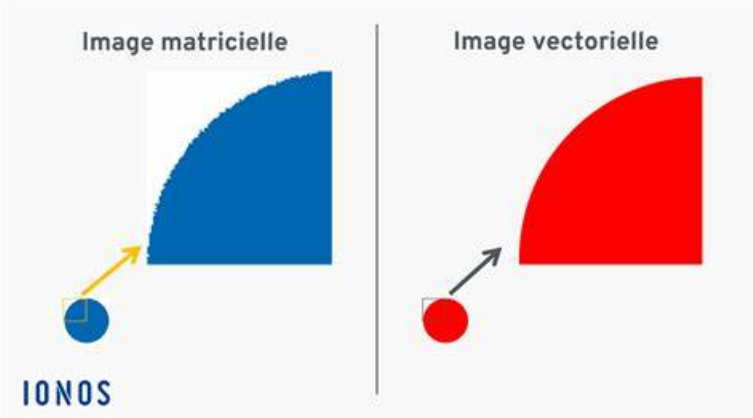


FIGURE 1.1 – Comparaison image vectorielle et matricielle

En informatique, les couleurs sont codées grâce à la méthode du RVB (rouge, vert, bleu) ou RGB (red, green, blue) en anglais . Cette méthode est très simple ; imaginons une photo en noir et blanc. Le blanc vaut 255 et le noir 0 (255 étant le nombre de valeur maximum que nous pouvons mettre dans un octet), nous aurions alors une image en nuances de gris. Maintenant, étendons cette méthode sur des nuances de rouge, de vert et de bleu, soit 255 x 255 x 255 nuances de couleurs (16 581 375). L'œil humain ne peut en voir d'autres différentes de celle-ci. [9]

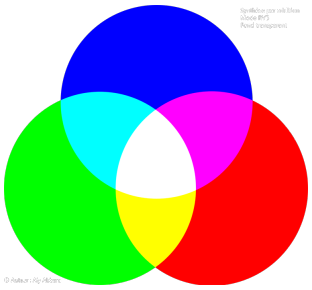


FIGURE 1.2 – Spectre de couleurs

1.2 Comment stocker une image ?

Le stockage d'une image peut s'effectuer de multiples manières. Il peut être soit externe soit interne. Par exemple, nous pouvons utiliser un disque dur externe, un CD, un DVD, des cartes mémoires ou encore des clés USB. Pour un stockage en interne, un système qui est de plus en plus utilisé est le cloud ou le drive. Nous utilisons pour notre projet cette solution. Celui ci permet d'avoir une image de très bonne qualité et accessible à tout le groupe. De plus, étant donné que nous travaillons sur des jeux de données de l'ordre d'une centaine d'images, nos échantillons sont enregistrés sous le format ".tiff", qui nous permet d'avoir accès à toutes nos images informatiquement en passant par une fonction déjà existante en python.

1.3 Le traitement d'image

1.3.1 Qu'est ce que le traitement d'image ?

Le traitement d'image consiste à étudier, améliorer des images ou encore en extraire des informations grâce à une approche informatique ou bien grâce aux mathématiques appliquées.

Ce sous - ensemble du traitement du signal passe aussi par le fait de comprendre les conditions dans lesquelles la photo a été prise. Sur ce point, beaucoup de facteurs rentrent en compte comme par exemple les conditions d'éclairage, la résolution d'acquisition et le mode de codage utilisé lors de la numérisation ou encore le réglage optique utilisé par exemple.

Le traitement d'image est donc une notion très large. Que ce soit l'objet étudié ou la méthode prise, les domaines sont très variés. Mais tout d'abord, avant de vouloir les modifier, parlons de l'acteur principal : l'image. En effet, une image n'est pas aussi simple que l'on ne pense.

FIGURE 1.3 – Amélioration de la qualité de l'image à l'aide de la méthode ACP sur Python

1.3.2 Les différentes méthodes de traitement d'image[4]

Dans les cas les plus généraux, le traitement d'image passe par les étapes suivantes :
- Acquisition : c'est à dire l'échantillonnage et la quantification
- Analyse globale de l'image et transformations ponctuelles
- Opérations entre images
- Amélioration, filtrage et segmentation
- Interprétation et sémantique

Afin de recueillir nos images, nous avons, pour respecter la problématique de notre projet, utilisé l'ombroscopie. Cette partie se fait en commun avec les membres de l'autre groupe de projet qui ont quant à eux fait l'installation qui permet de recueillir les images.

1.3.3 Les facteurs pouvant altérer le traitement d'une image[6]

Comme mentionné précédemment, certains facteurs altèrent le traitement d'une image.
- La résolution d'acquisition (nombre de pixel en hauteur et en largeur)
- Le mode de codage utilisé lors de la numérisation, qui déterminent le degré de précision des éventuelles mesures de dimensions,
- Les réglages optiques utilisés (dont la mise au point) qui déterminent par exemple la netteté de l'image (réglage dépendant entièrement de l'appareil qui capture l'image)
- Les conditions d'éclairage, qui déterminent une partie de la variabilité des images traitées,
- Le bruit de la chaîne de transmission d'image (les transferts d'image trop récurrent par exemple)

Il est important d'avoir une image de la meilleure qualité possible pour pouvoir la traiter avec la plus haute précision. Pour cela, nous pouvons effectuer quelques réglages au préalable qui nous permettent de traiter l'image plus facilement et avec un jeu de données de meilleure qualité. Avoir une bonne exposition est une des premières choses dont il faut s'assurer, il en va de même pour un bon contraste et une saturation correcte. La netteté de la photo peut quant à elle s'améliorer grâce aux nombres de pixels qui peuvent être modifiés en passant par de la programmation. Python nous permet déjà de le faire extrêmement bien de façon très simple, c'est une des raisons pour lesquelles nous avons utilisé ce language.

1.4 L'ombroscopie[5][7]

L'ombroscopie est une méthode de visualisation. Son principe est le suivant : l'ombre portée d'un objet transparent est placée dans un faisceau de lumière statique. Toute la technique est basée sur le phénomène de déviation ou d'obstruction des faisceaux lumineux. Ainsi, nous avons des images en noir et blanc avec des teintes de gris qui sont projetées contre une paroi ou une caméra afin de recueillir les images.

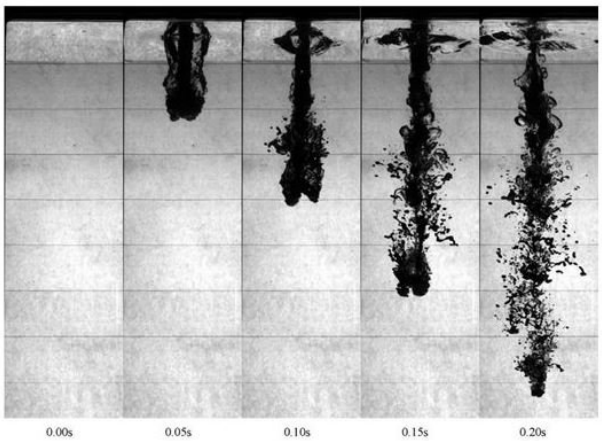


FIGURE 1.4 – Images obtenues grace à l'ombroscopie

Chapitre 2

Un spray : késako ?

2.1 Un peu d'histoire [10]

L'apparition des boissons gazeuses pressurisées en France à partir de 1790 a introduit le concept d'aérosol. Ce fut pendant la première guerre mondiale que le terme aérosol a été utilisé pour la première fois par Frederick G. Donnan pour décrire l'aérosolisation, des nuages de particules microscopiques dans l'air.

Ce n'est qu'en 1927 qu'un inventeur chimiste Norvégien nommé Erik Rotheim fit breveter l'utilisation d'un récipient sous pression muni d'une valve permettant de diffuser plusieurs produits. En 1931, son procédé a été vendu aux Américains.

La vente du premier insectide sous la forme d'une petite bonbonne contenant un liquide sous pression et munie d'une valve à vis s'est faite aux États-Unis en 1941 et constitue donc la première mise en application de ce principe,

La commercialisation de générateurs d'aérosols à utilisation plus commune et quotidienne comme les diffuseurs de crème chantilly, de peinture ou de laque coiffante a vu le jour après la guerre, à partir de 1948.

Cette même mise en marché de générateurs d'aérosol a commencé à se démocratiser en France au milieu des années 1950 notamment avec des produits alimentaires, des produits d'entretien, des médicaments ou encore des produits cosmétiques.

L'industrie quant à elle utilise les générateurs d'aérosols pour des fins techniques plus élaborées.

FIGURE 2.1 – Contenant d'insecticide datant des années 1950

2.2 Définition et fonctionnement [3]

Définition

Un générateur d'aérosol constitue un système de distribution qui génère un colloïde de particules solides fines ou de gouttelettes liquides, dans l'air ou dans un autre gaz. Ce brouillard est ce qu'on appelle spray.

Constituants

Ce système comprend plusieurs composants :

— **La bouteille :** ou la canette, qui contient la charge utile et un propulseur sous pression. Celle-ci peut être faite d'acier étamé ou d'aluminium.

— **La vanne :** bien que ce soit un composant non visible, son rôle est primordial car celle-ci assure l'étanchéité du récipient ainsi que sa propreté, et régule le débit du produit pendant l'utilisation.

— **L'actionneur :** ou le bouton, qui contrôle l'angle, la quantité, la forme et la finesse du produit pulvérisé. Le produit et l'utilisation de l'aérosol influencent grandement le choix de l'actionneur.

Fonctionnement

Quand on appuie sur l'actionneur, l'ouverture de la vanne du conteneur est enclenchée. Il y a donc expulsion de la charge utile à travers une petite ouverture sous forme d'aérosol ou de brouillard. Le gaz propulseur exerce une pression sur le produit actif et la solution de solvant, forçant le liquide à traverser le tube plongeur et à travers la vanne lorsqu'il est ouvert. Ceci permet au produit d'être expulsé avec le propulseur sous forme de gouttelettes, de mousse, de pâte ou de poudre.

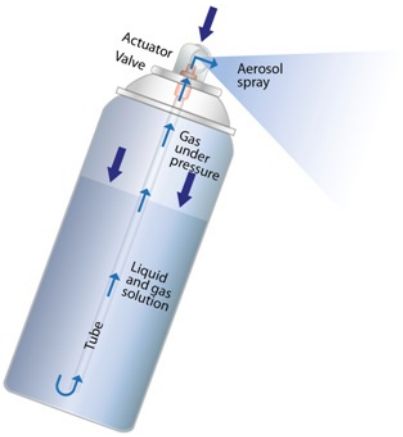


FIGURE 2.2 – Schéma annoté d'un générateur d'aérosol

2.3 Différents types de buses de spray [1]

Buses à cône complet

Les buses à cône complet ont tendance à produire des gouttes de taille moyenne à grande qui forment un motif de pulvérisation conique uniforme solide avec une zone d'impact ronde. Elles fournissent un impact élevé par unité de surface. Les angles de pulvérisation vont de 17° à 170°.

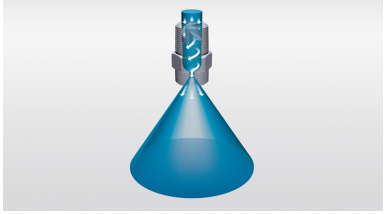


FIGURE 2.3 – Buses à cône complet

Buses à cône creux

Les buses à cône creux produisent des gouttes de petite à moyenne taille qui forment un motif de cône creux avec une zone d'impact en forme d'anneau. Elles permettent une distribution uniforme et une atomisation efficace des liquides à une large gamme de débits et de pressions, notamment les plus basses. Les angles de pulvérisation vont de 43° à 180°.

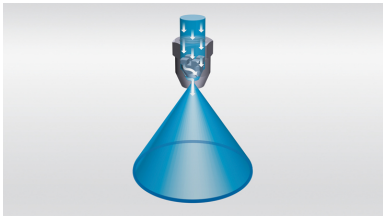


FIGURE 2.4 – Buses à cône creux

Buses de pulvérisation plates

Les buses de pulvérisation plates produisent des gouttes de petite à moyenne taille formant un motif de pulvérisation plat ou de type feuille. Les angles de pulvérisation vont de 0° à 110°.

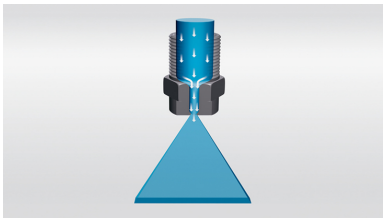


FIGURE 2.5 – Buses de pulvérisation plates

2.4 Applications d'un spray [11]

- **Pulvérisation de carburant** (injecteurs de carburant pour les moteurs à essence et diesel et moteurs fusée, atomiseurs pour les moteurs à réaction et les chaudières à vapeur)
- **Production d'électricité** (tours de pulvérisation pour éliminer les solides particulaires, pour la distribution d'eau)
- **Industrie alimentaire** (séchage par atomisation, enrobage, nettoyage et désinfection)
- **Fabrication** (application d'adhésifs, lubrification, dégraissage, pulvérisations à haute pression pour ébavurer des pièces usinées, peinture par pulvérisation, écorçage, séchage)
- **Protection incendie** (gicleurs fixes, systèmes de brumisation, systèmes de déluge, systèmes de tunnel d'eau)
- **Chaux et ciment** (systèmes de refroidissement et de conditionnement, alimentation en combustible, suppression de la poussière des matières premières)
- **Chimie, pétrochimie et pharmaceutique** (tours de pulvérisation de réactifs, catalyseur de craquage de fluide de séchage par atomisation pour le raffinage du pétrole, nettoyage des filtres et des centrifugeuses, enrobage de comprimés, revêtement)
- **Traitement des déchets** (injection de déchets liquides dans les incinérateurs à haute température)
- **Applications agricoles** (pulvérisation d'herbicides/insecticides/pesticides, refroidissement des animaux)
- **Produits de consommation** (produits d'entretien, laque à cheveux, déodorant, parfum)

2.5 Les facteurs qui influent sur l'angle d'ouverture d'un spray [8]

Un spray peut être caractérisé par la taille moyenne de ses gouttelettes (diamètre moyen) et son angle d'ouverture.

Les facteurs qui influent sur l'angle d'ouverture d'un spray sont :

— **Les propriétés du fluide** (densité, masse volumique, viscosité..)
— **Le type de buse**
— **La pression d'alimentation**

La conception de la buse participe dans l'influence importante de la pression d'alimentation sur la valeur de l'angle de pulvérisation. En général, avec une pression plus importante, les buses à cône complet produiront des angles plus étroits, les buses à jet plat un angle de pulvérisation plus large, tandis que les buses en spirale seront moins affectées par les changements de pression. Toutes les buses ne fonctionneront pas correctement avec des valeurs de pression très faibles avec une dégradation marquée des performances, des chutes plus importantes, un schéma de pulvérisation mal défini, des valeurs d'angle de pulvérisation plus faibles...

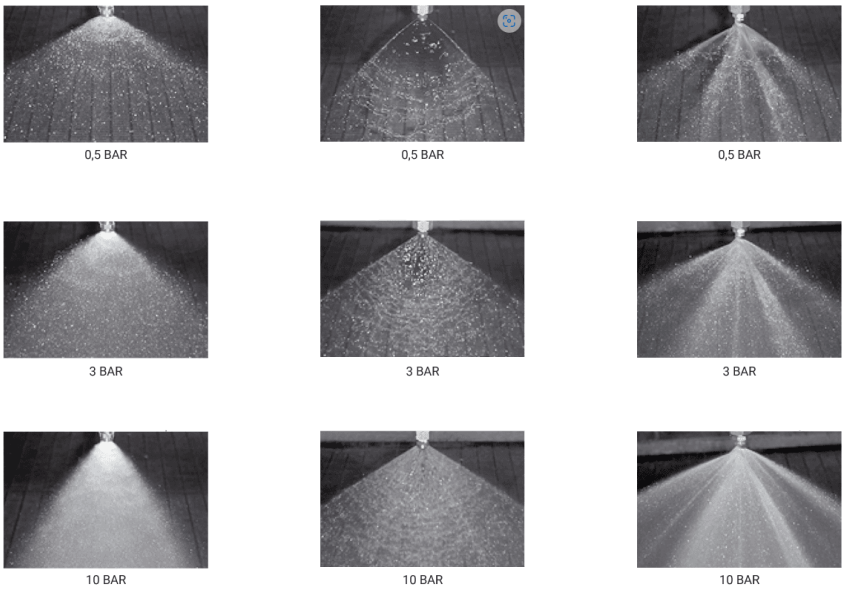


FIGURE 2.6 – Influence de la pression d'alimentation sur l'angle d'ouverture d'un spray pour des buses conique complète, à jet plat, et spirale [2]

Exemple d'étude de l'influence des propriétés du fluide et de la pression d'alimentation sur l'angle d'ouverture d'un spray

Dans le cadre de l'étude «*Spray angle characteristics of Carotino-Diesel blends*»[8], on retrouve une analyse de l'influence des propriétés du fluide ainsi que de la pression d'injection sur l'angle d'ouverture d'un spray.
Pour 3 valeurs de pression d'injection différentes (8, 10 et 12 bar), on fait varier le pourcentage de Carotino dans le mélange.
Le Carotino est constitué de 80% d'huile de palme et de 20% d'huile de Canola. Ainsi, en ajouter à différents pourcentages fait varier les propriétés physiques de ce mélange, notamment sa viscosité. Les mélanges nommés B0, B5, B10, B15, B20 et B25 contiennent respectivement des pourcentages de plus en plus élevés de Carotino.

Les images obtenues grâce à une caméra DSLR (digital single lens reflex) ont été traitées à l'aide du logiciel Solidworks afin de trouver la valeur de l'angle d'ouverture du spray.

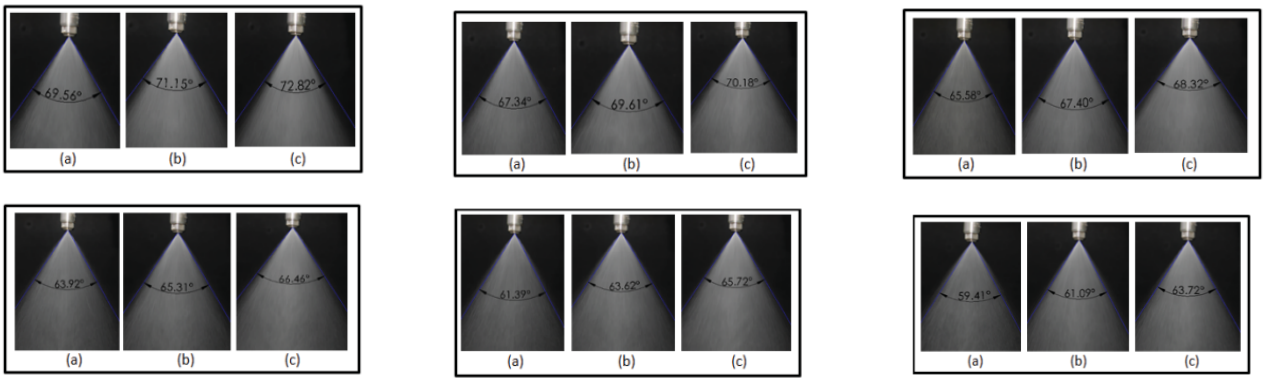


FIGURE 2.7 – Différentes valeurs de l'angle d'ouverture pour 8, 10, et 12 bar respectivement pour différents poucentages de Carotino dans le mélange (B0, B5, B10, B15, B20 et B25 respectivement)

Conclusions

Concernant les variations de pression, on a constaté que plus la pression augmente, plus l'angle est grand. Quant au pourcentage de Carotino dans le mélange, plus il est important, moins l'angle est grand.

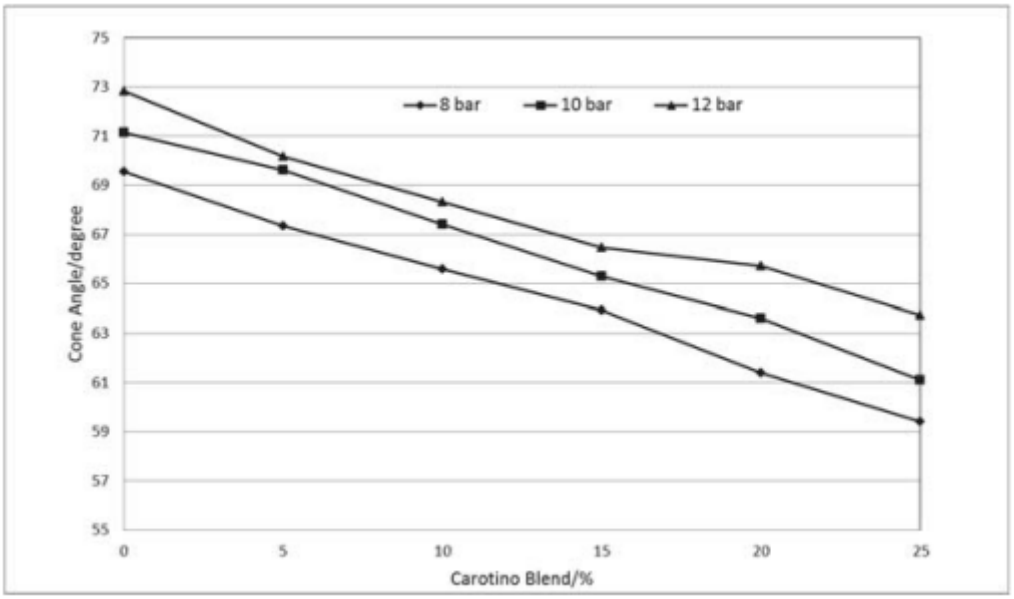


FIGURE 2.8 – Influence de la pression et du pourcentage de Carotino sur l'angle d'ouverture du spray

Chapitre 3

Mise en Application

Les images de sprays ont été traitées à l'aide du langage de programmation *Python* et des bibliothèques suivantes : *OpenCV-python*, *scikit-learn*, *numpy*, *PIL* et *pyautogui*. Leur installation sera donc nécessaire à l'utilisation du programme. Il utilise également les bibliothèques *os*, *math* et *platform*, qui font partie de l'installation de Python par défaut. Afin faire fonctionner ce programme au mieux, il est nécessaire de lui fournir 3 séries d'images (contenant le même nombre d'images de la même taille) :

— Une série d'images représentant le spray lui-même, en fonction du temps ;
— Une série d'images représentant le fond sur lequel le spray a été filmé ;
— Une série d'images prise avec l'objectif de la caméra obstrué.

Il est également possible de ne fournir que l'image du jet, mais cela a un impact massif sur la qualité du traitement et subséquemment sur la précision du résultat. Dans le but de déterminer l'angle du cône de spray, nous avons suivi la démarche suivante : Cropping des images (ou Recadrage), Normalisation des images, Binarisation, Création d'une image « écart-type », Régression linéaire et enfin Calcul de l'angle.

3.1 Cropping des images

C'est l'opération qui consiste à supprimer les parties périphériques des images qui ne contiennent pas le spray, autrement dit dénuées d'informations utiles. Cette étape permet de minimiser le coût (en temps et en énergie) du traitement d'image. En effet, le premier échantillon sur lequel nous avons travaillé était un ensemble de 3 matrices de 120 images de taille 2048 x 2048 pixels. Le cropping permet, dans le cas de cet exemple, de plutôt traiter pour un même résultat 3 matrices de 120 images de taille 1822 x 715 pixels (soit 31,06 % de la quantité de données originale). Dans ce but, nous avons créé deux fonctions, la première pour un recadrage automatique et la deuxième pour un recadrage manuel.

— **Le recadrage automatique, "*cropCanny()*"**
Il se base sur la détection de contour de Canny appliqué sur une image du spray. On obtient une approximation du contour du spray, à une marge d'erreur donnée près. Après voir déterminé les limites du spray (à gauche, à droite, au dessus et en dessous), on recadre chacune des images en conséquence. La fonction qui permet l'utilisation de la détection de contour est fournie par OpenCv : cv.Canny()

— **Le recadrage manuel, "*cropManual()*"**
Cette fonction permet à l'utilisateur d'afficher une image du spray et de sélectionner la plus petite portion d'image le contenant intégralement. La portion d'image sélectionnée est alors utilisée pour recadrer toutes les autres images, que ce soit de la même catégorie (IE les autres images du spray en fonction du temps) ou non (IE les images du fond et avec caméra obstruée).

3.2 Normalisation des images

C'est une étape importante du prétraitement des images. Elle permet de réduire les anomalies et imprécisions de l'image dues aux changements de luminosité en fonction du temps et à la caméra elle-même. En

normalisant les images, on réduit ces variations afin de mieux mettre en évidence le motif (la forme du spray). La normalisation se base ainsi sur la différence des valeurs de luminosité de chaque pixel entre les images du spray, de l'arrière-plan sur lequel celui-ci a été filmé, des images capturées avec l'objectif de caméra obstrué (qui révèlent les éventuelles anomalies lumineuses propres à la caméra elle-même, c'est-à-dire les points des images numériques qui ne sont pas entièrement noirs alors qu'ils devraient l'être). Avec ces données, nous avons choisi d'appliquer à chaque pixel la relation suivante, issue d'un article de 2005 par MM. Blaisot et Yon[13] :

$$\mathbf{P}_n(i,j) = \beta\, \mathbf{P}_{norm} \frac{\mathbf{P}_{init}(i,j) - \mathbf{P}_0(i,j)}{\mathbf{P}_B(i,j) - \mathbf{P}_0(i,j)}$$

Avec β un coefficient qui dépend de la stabilié temporelle de l'intensité lumineuse (ici la moyenne de celle de l'image du spray), $\mathbf{P}_{init}(i,j)$ le niveau de gris du pixel (i,j), $\mathbf{P}_B(i,j)$ le niveau de gris du pixel (i,j) en arrière plan sans objet, $\mathbf{P}_0(i,j)$ le niveau de gris moyen du pixel (i,j) obtenu après l'obturation du capteur, $\mathbf{P}_{norm}$ le niveau de gris moyen de l'arrière plan. Finalement, $\mathbf{P}_n(i,j)$ est le niveau de gris du pixel de l'image normalisée.

3.3 Binarisation

La binarisation ou seuillage est la forme la plus simple de segmentation par région. Celle-ci permet de mettre en évidence et d'isoler les différents objets présents dans une image. Binariser une image revient donc à segmenter l'image mais en deux classes seulement : le fond et l'objet, ici le spray. Dans le cadre de notre étude, deux classes sont largement suffisantes. Pour binariser les images, nous avons eu recours au seuillage global dont le principe est d'utiliser une valeur seuil à partir de laquelle on définit à quelle classe un pixel appartient :

$$\forall\,(i,j) \in N\text{x}M\ B(i,j) = \begin{cases} 1,\ si\ I(i,j) > S & (pixel\ blanc) \\ 0,\ sinon & (pixel\ noir) \end{cases}$$

Avec
- NxM : nombre de colonnes et de lignes de l'image ;
- B : Image binarisée
- I : Image d'origine
- S : Seuil de binarisation, valeur de pixel qui sépare le mieux le fond et l'objet sur l'image

Pour définir le seuil de manière automatique, nous avons utilisé la méthode d'Otsu fournie par OpenCV. Celle-ci calcule la variance interclasse entre le fond et l'objet pour toutes les valeurs S possibles (de 0 à 255). Cette variance quantifie la dissemblance qui existe entre les pixels des deux classes. En conséquence, plus la variance interclasses est haute, moins les classes se ressemblent. Le seuil est le S qui correspond à la plus haute variance.

3.4 Calcul de l'écart-type

Une fois les images normalisées et binarisées du spray, on calcule leur écart-type en fonction du temps. On obtient ainsi une nouvelle image, cette fois unique (et non une matrice de plusieurs images) représentant la quantité de variation temporelle de chaque pixel de nos images précédentes. Puisque le fond des images d'origine reste identique en fonction du temps, et que le centre du jet reste en permanence sombre (il y a toujours du produit pour l'assombrir), les seules parties de ces images qui varient sont les bords du jet. Aussi, les zones les plus claires de l'image "écart-type" ainsi obtenue sont les deux bords du jet.

3.5 Calcul de l'angle du spray

Nous avons désormais, grâce aux étapes précédentes, une unique image qui représente les deux bords du jet. Intuitivement, il nous reste donc à déterminer "l'angle que ces deux bords forment entre eux". En termes plus mathématiquement rigoureux, on cherche donc à identifier deux vecteurs, chacun de même direction qu'un des bords du spray, et à calculer l'angle qu'ils forment entre eux.
Pour cela, nous avons une nouvelle fois implémenté deux méthodes, l'une automatique et l'autre manuelle :

— **Méthode automatique** Cette méthode, potentiellement la plus précise mais aussi moins fiable, consiste dans un premier temps à diviser l'image en deux entre les deux bords du spray. Chacune de ces deux images contient donc une zone claire (dans le meilleur des cas, une ligne, et le plus souvent, un cône) qui suit un des bords du spray, et que l'on peut simplifier en une droite.
Pour ce faire, on effectue sur ces deux images une régression linéaire, à l'aide de la fonction *LinearRegression* de la bibliothèque *Scikit-Learn*. On récupère alors les coefficients de cette régression, ce qui nous donne une fonction affine correspondant à la droite définissant la direction du bord du spray. On évalue alors celle-ci en deux points (arbitraires, sa norme n'ayant pas d'importance - nous avons ici choisi le haut et le bas de l'image) et on crée un vecteur à partir des valeurs obtenues (sous la forme $(x - f(x), y - f(y))$; autrement dit, on créé le vecteur qui transforme le point appartenant à notre droite d'abcisse x en celui appartenant également à la droite d'abcisse y). Ces étapes étant effectuées sur les deux images, on a donc un vecteur qui "suit" le bord gauche du spray et un qui "suit" le bord droit. Notons que plus le jet s'éloigne de l'embout, plus il est susceptible de varier dans sa direction et plus il risque d'y avoir des gouttelettes qui s'éloignent du reste du spray. Cela signifie que sur l'image représentant l'écart-type, plus l'on s'éloigne de la position de l'embout, plus fortes et plus dispersées sont les variations (plus simplement, "plus il y a de blanc"), comme on peut le voir sur cette image :

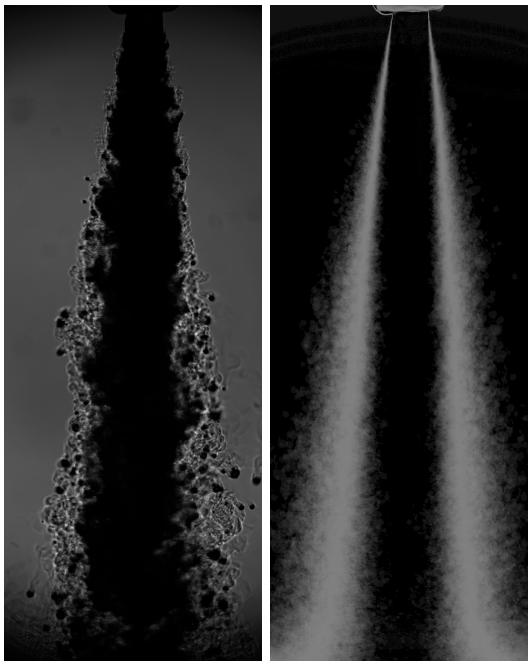


FIGURE 3.1 – Image "écart-type" montrant la position des deux bords d'un spray

Or, puisqu'il y a pour cette raison beaucoup plus de données vers le bas de l'image, la régression (soit la droite qui, de toutes celles possibles, est la plus proche de tous les points) est plus influencée par la direction que les bords du spray prennent loin de l'embout que proche de lui. Comme ce que nous cherchons à obtenir est l'angle du spray en sortie d'embout (sachant qu'il a tendance à diminuer avec la distance), cela fausse grandement les résultats. Nous avons donc décidé d'ajouter un curseur qui contrôle l'importance que donne la régression au haut de l'image par rapport au bas (bien sûr, pour nous assurer que le haut de l'image soit toujours là où se situe l'embout, nous avons donné à l'utilisateur la possibilité de tourner les images dans le bon sens). Ainsi, en fonction de ce curseur que nous avons appelé "adhérence au haut du spray", l'utilisateur peut forcer la régression à calculer la direction des bords du spray en sortie d'embout.

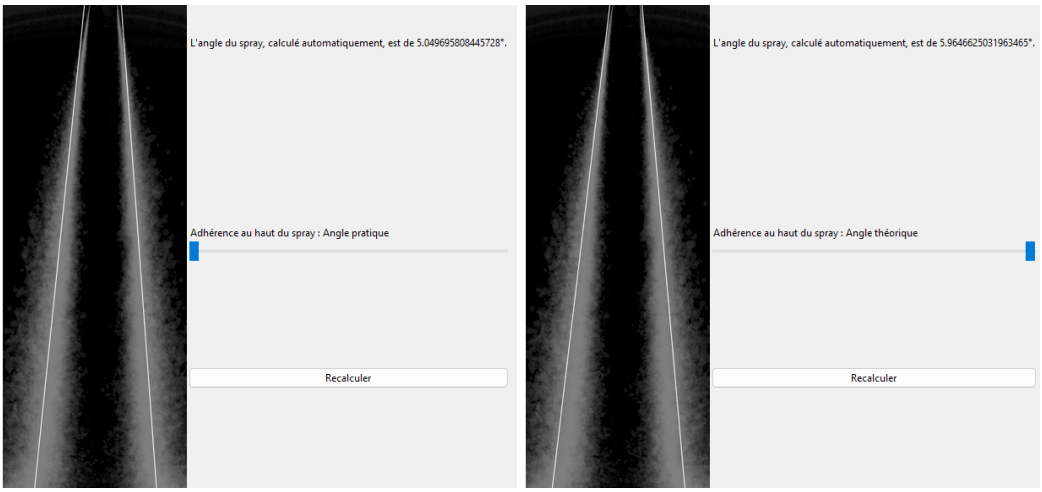


FIGURE 3.2 – Résultats de la régression en fonction du curseur d'adhérence au haut de l'image

Il faut cependant noter que les imprécisions lors des étapes précédentes (une mauvaise qualité d'image, une mauvaise normalisation, un élaguage trop sévère...) peuvent avoir une assez grande influence sur le résultat, d'où l'existence d'une deuxième méthode.

— **Méthode manuelle** Cette méthode est prévue pour vérifier, ou si nécessaire remplacer, la méthode automatique dans le cas où elle fournirait un résultat inattendu. Elle consiste assez simplement à afficher l'image et à permettre à l'utilisateur d'y tracer un segment le long de chaque bord. On récupère alors les abcisses et ordonnées de départ et d'arrivée de ces deux segments et on s'en sert, comme précédemment, pour créer deux vecteurs qui définissent la direction des bords du jet.

Finalement, indifféremment de la méthode, on obtient deux vecteurs dont les directions correspondent à celles de nos bords. Il nous reste donc seulement à calculer leur angle, selon la formule :

$$\theta = \arccos \frac{\|\vec{u}\| \, \|\vec{v}\|}{\vec{u} \cdot \vec{v}}$$

Avec $\vec{u}$ et $\vec{v}$ deux vecteurs et θ l'angle entre eux.

3.6 Mise à l'épreuve par une analyse physique

Enfin, nous avons essayé d'éprouver notre programme au travers d'une analyse physique. Pour ce faire, nous avons comparé les angles de plusieurs jets en fonction de leur pression.

Pression ambiante (Bar)	Température Ambiante (°C)	Débit (mL/min)	Angle (°)
45	60	380	8,12
50	60	380	8,03
55	60	380	7,9

Ces résultats semblent montrer qu'une augmentation de la pression ambiante fait légèrement diminuer l'angle d'ouverture du spray (dans l'ordre d'un dixième de degré pour 5 Bars supplémentaires). Ces résultats restent cependant à manipuler avec précaution, car il est difficile de déterminer l'incertitude inhérente à ces calculs. Par ailleurs, certaines anomalies peuvent parfois, inévitablement, influencer les valeurs.

Conclusion et perspectives

Dans le cadre de ce projet, notre objectif était de déterminer informatiquement l'angle d'ouverture d'un spray. Nous avons tout d'abord commencé par un travail bibliographique important sur les images ainsi que leur traitement, mais aussi sur les sprays, leur fonctionnement, leur applications, etc...

Ensuite, nous nous sommes consacrés au traitement d'un jeu d'image fourni par notre encadrant. Pour ce faire, nous avons choisi de suivre la démarche suivante :

FIGURE 3.3 – processus suivi pour le traitement d'images

Grâce à cette méthode, on a pu estimer de manière assez précise la valeur de l'angle d'ouverture du spray pour un jeu d'images donné.

Enfin, afin de faire un lien entre la documentation sur les sprays et la partie informatique, nous avons effectué une analyse physique en comparant les angles sur un autre jeu d'images cette fois-ci faisant intervenir plusieurs valeurs de pression différentes. Notre programme nous a permis de constater que l'augmentation de la pression ambiante implique la diminution de l'angle du spray.

Ce projet nous a permis de développer nos capacités de travail en groupe, d'organistion et d'adaptabilité. La répartition de tâches s'est faite assez naturellement et rapidement, chose qui nous a permis d'avancer chacun sur la partie qui suscitait le plus son interêt.

Ce projet nous a aussi permis d'élaborer une méthode pour le traitement d'image dans un langage de programmation (Python) qui était jusque là, à certains d'entre nous, très peu familier. Ceci nous a permis d'apprendre à mieux le manipuler et par conséquent, ce projet nous a donc apporté beaucoup sur le plan pédagogique et a beaucoup influé notre choix d'orientation.

Pour autant, ce projet pourrait être poussé plus loin sur plusieurs aspects ; il serait par exemple possible de rendre le programme plus efficace, de tester de nouvelles méthodes afin de tenter d'améliorer la précision des résultats, de permettre à l'utilisateur de calculer l'angle avec d'autres types d'images...

Bibliographie

[1] Spraying Systems Co. *Différents types de buses*. URL : https://www.spray.com/en-gb/resources/spraying-basics/spray-patterns.

[2] Spraying Systems Co. *Influence de la pression*. URL : https://www.pnr.eu/nozzle-anatomy-spray-angle/.

[3] FEA : European Aerosol Federation. *Fonctionnement interne d'un spray*. URL : https://www.aerosol.org/about-aerosols/.

[4] UMLV Jean Fruitet. *Outils et méthodes pour le traitement d'images par ordinateur*. URL : https://pierrekueny.fr/AncienSite/ati/partie2/rendu/shaders/an_imag.pdf.

[5] Techniques de l'ingénieur. *Ombroscopie*. URL : https://www.techniques-ingenieur.fr/glossaire/ombroscopie.

[6] Maxicours. *Caractéristiques d'une image numérique*. URL : https://www.maxicours.com/se/cours/caracteristiques-d-une-image-numerique/.

[7] ENS de Lyon Olivier Dequincey - Damien Mollex. *Visualisation par ombroscopie de systèmes convectifs à une et deux couches*. URL : https://planet-terre.ens-lyon.fr/ressource/ombroscopie-convection.xml.

[8] Muhammad Roslan Rahim et Mohammad Nazri Mohd Jaafar. *Spray angle characteristics of carotino-diesel blends*. URL : https://pubs-aip-org.ezproxy.normandie-univ.fr/aip/acp/article/2339/1/020241/1028956/Spray-angle-characteristics-of-carotino-diesel?searchresult=1.

[9] Encyclopédie Universalis. *Image numérique et image de synthèse*. URL : https://www.universalis.fr/encyclopedie/image-numerique-et-image-de-synthese/.

[10] Wikipédia. *Histoire du spray*. URL : https://en.wikipedia.org/wiki/Aerosol_spray_dispenser#History.

[11] Wikipedia. *Applications des sprays*. URL : https://en.wikipedia.org/wiki/Spray_(liquid_drop)#Applications.

[12] Wikipedia. *Image numérique*. URL : https://fr.wikipedia.org/wiki/Image_num%C3%A9rique.

[13] J. B. Blaisot - J. Yon. "Droplet size and morphology characterization for dense sprays by image processing: application to the Diesel spray". In : *Experiments in Fluids* 39.6 (2005), p. 977-994. URL : https://link.springer.com/article/10.1007/s00348-005-0026-4.

Table des figures

Annexes

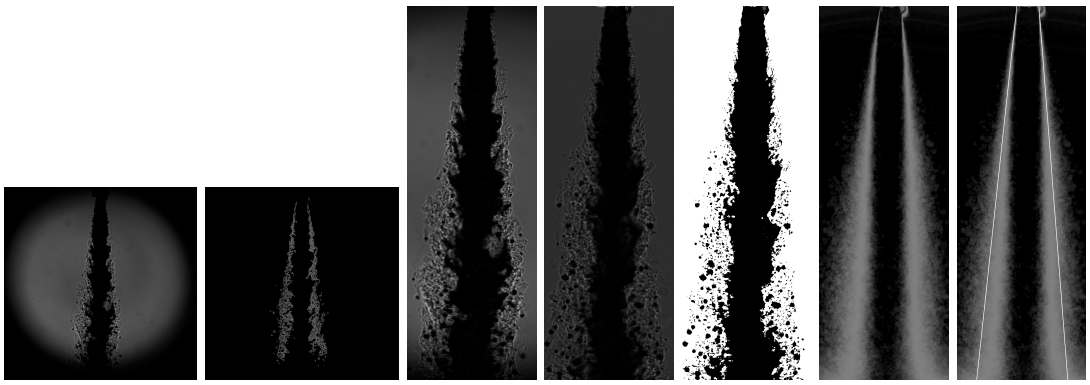


FIGURE 3.4 – Image au fil des différentes étapes de traitement : originale, détection de contour avec Canny, recadrage, normalisation, binarisation, écart-type et enfin régression

Code

```
import matplotlib.pyplot as plt
import cv2
import numpy
import numpy as np
import os
import math
import pyautogui
from sklearn.linear_model import LinearRegression
import platform


##################Chargement des images####################
def load_images(*paths):
    """Charge les images des chemins en paramètres"""
    res = ()
    for path in paths:
        if type(path) is str:
            if ":" in path:
                if os.path.isfile(path):
                    _bool, _image = cv2.imreadmulti(path)
                    _image = np.array(_image)
                    res = res + (_image,)
                else:
                    imageSet = ()
                    for file in os.listdir(path):
                        _bool, _image = cv2.imread(os.path.join(path, file))
                        imageSet = imageSet + (_image,)
                    imageSet = np.array(imageSet)
```

```
                    res = res + (imageSet,)
            else:
                if os.path.isfile(os.path.join(os.getcwd(), "A Traiter", path)):
                    _bool, _image = cv2.imreadmulti(os.path.join(os.getcwd(), "A Traiter", path))
                    _image = np.array(_image)
                    res = res + (_image,)
                else:
                    imageSet = ()
                    for file in os.listdir(os.path.join(os.getcwd(), "A Traiter", path)):
                        _bool, _image = cv2.imread(os.path.join(os.getcwd(), "A Traiter", path, file))
                        imageSet = imageSet + (_image,)
                    imageSet = np.array(imageSet)
                    res = res + (imageSet,)
        else:
            raise TypeError("Image paths must be strings")
    if len(paths) == 1:
        return res[0]
    else:
        return res


##################Cropping###################
def cropCanny (*images):
    """Recadre automatiquement les images"""
    edge = cv2.Canny(images[0][2], 50, 150, 3)
    write_images_to_file(folder="Edge", Edge=edge)
    lignes, colonnes = edge.shape
    left_bound = 0
    while np.sum(edge[:, left_bound]) < 0.00001:
        left_bound +=1
    right_bound = colonnes-1
    while np.sum(edge[:, right_bound]) < 0.00001:
        right_bound -=1
    top = 0
    while np.sum(edge[top, :]) < 0.00001:
        top +=1
    bottom = lignes-1
    while np.sum(edge[bottom, :]) < 0.00001:
        bottom -=1
#    left_bound -= int(colonnes/100)
#    right_bound += int(colonnes/100)
#    top -= int(lignes/120)
#    bottom += int(lignes/120)
    resultat = ()
    for image in images:
        image_temp = ()
        for frame in image:
            frame = frame[top:bottom+1,left_bound:right_bound+1]
            image_temp = image_temp + (frame,)
        image_temp = np.array(image_temp)
        resultat = resultat + (image_temp,)
    return resultat


def crop_manual(*images):
    """Permet de recadrer manuellement les images"""
    res = ()
    x_start, y_start, x_end, y_end = 0, 0, 0, 0
```

```
    tracing, done, traced_one = False, False, False
    image = images[0][0]
    def trace_box(event, x, y, flags, *userdata):
        nonlocal x_start, y_start, x_end, y_end, tracing, image, done, traced_one
        if event == cv2.EVENT_LBUTTONDOWN:
            x_start, y_start, x_end, y_end = x, y, x, y
            tracing = True
        elif event == cv2.EVENT_MOUSEMOVE:
            if tracing:
                x_end, y_end = x, y
        elif event == cv2.EVENT_LBUTTONUP:
            x_end, y_end = x, y
            tracing = False
            traced_one = True
        elif event == cv2.EVENT_RBUTTONDOWN:
            done = True
    cv2.namedWindow("image", cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO)
    cv2.setWindowProperty("image", cv2.WND_PROP_ASPECT_RATIO, cv2.WINDOW_KEEPRATIO)
    if platform.system() == 'Linux':
        cv2.setWindowProperty("image", cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
    cv2.setMouseCallback("image", trace_box)
    while not done:
        i = image.copy()
        if not tracing and not traced_one:
            cv2.imshow("image", image)
        else:
            cv2.rectangle(i, (x_start, y_start), (x_end, y_end), (255, 0, 0), 2)
            cv2.imshow("image", i)
        cv2.waitKey(5)
    refPoint = [(x_start, y_start), (x_end, y_end)]
    cv2.destroyAllWindows()
    for image in images:
        res = res + (image[:,refPoint[0][1]:refPoint[1][1], refPoint[0][0]:refPoint[1][0]],)
    if len(images) == 1:
        return res[0]
    else:
        return res


##################Ecriture dans fichier####################
def write_images_to_file(folder, **images):
    """Retranscrit les images dans un fichier .tiff (à plusieurs couches si nécessaire), dans le sous-d
    [folder] du dossier courant """
    path = os.path.join(os.getcwd(), folder)
    os.makedirs(path, exist_ok=True)
    for image in images:
        if images.get(image).ndim > 2:
            if not (cv2.imwritemulti(f'{path}/image{image}.tif', images.get(image))):
                raise Exception("Something went wrong when writing images to files.")
        else:
            if not (cv2.imwrite(f'{path}/image{image}.tif', images.get(image))):
                raise Exception("Something went wrong when writing images to files.")


##################Normalisation####################
def normalize(*images):
    """Applique la bonne méthode de normalisation, en fonction du nombre d'images chargées"""
    if len(images) == 1:
```

```
        res = ()
        for image in images[0]:
            res = res + (cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX),)
        return np.array(res)
    elif len(images) == 3:
        return normalize_formula(images[0], images[1], images[2])


def normalize_formula(image_jet, image_background, image_noise):
    """Normalise les images en utilisant les images de fond et de bruit"""
    P_norm = np.mean(image_jet)
    z, y, x = np.shape(image_jet)
    beta = image_background[:, int(y / 4), 0] / image_jet[:, int(y / 4), 0]
    beta = beta[:, np.newaxis, np.newaxis]
    imageBack_Noise = (image_background - image_noise)
    imageBack_Noise = np.where(imageBack_Noise <= 0, np.ones_like(imageBack_Noise) / 100, imageBack_Noi
    image_jet = beta * P_norm * (image_jet - image_noise) / imageBack_Noise
    image_jet = image_jet.astype('uint8')
    return image_jet


##################Binarisation####################
def threshold(images):
    """Binarise les images"""
    res = np.zeros_like(images)
    for i, image_t in enumerate(images):
        thresh_value, image_t = cv2.threshold(image_t, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
        res[i] = image_t
    return res


##################Ecart-Type####################
def standard_deviation(images):
    """Créé une matrice à deux dimensions de l'écart-type des images"""
    imageEcart_type = np.std(images, axis=0)
    return imageEcart_type


##################Régression####################
def dichotomy(image, center):
    """Coupe une image en deux autres selon la valeur x définie comme centre"""
    leftimage = image[:, :(center+1)]
    rightimage = image[:, (center+1):]
    return leftimage, rightimage


def linearRegression(image, adherence):
    """Effectue une régression linéaire et renvoie les coefficients correspondants"""
    thresh_value, imagethr = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    XY = np.nonzero(imagethr)
    XY = np.array(XY)
    X = XY[0, :].reshape(-1, 1)
    y, x = X.shape
    weights = np.ones((y*x))
    weights[0:int(y*x/15)] = adherence
    weights[int(y*x/15)+1:2*int(y*x/15)] = adherence/2
    weights[2*int(y*x/15)+1:3*int(y*x/15)] = 1
    weights[3 * int(y * x / 15) + 1:y] = 1/adherence
```

```
    linearR = LinearRegression().fit(X, XY[1, :].reshape(-1, 1), weights)
    return linearR.coef_[0][0], linearR.intercept_[0]


##################Calcul de l'angle####################
def get_vectors_manual(imageSTD):
    """Permet de tracer manuellement une droite le long du spray et renvoie le vecteur correspondant"""
    x_start, y_start, x_end, y_end = 0,0,0,0
    tracing, done, traced_one = False, False, False
    image = imageSTD
    def trace_lines(event, x, y, flags, *userdata):
        nonlocal x_start, y_start, x_end, y_end, tracing, image, done, traced_one
        if event == cv2.EVENT_LBUTTONDOWN:
            x_start, y_start, x_end, y_end = x, y, x, y
            tracing = True
        elif event == cv2.EVENT_MOUSEMOVE:
            if tracing:
                x_end, y_end = x, y
        elif event == cv2.EVENT_LBUTTONUP:
            x_end, y_end = x, y
            tracing = False
            traced_one = True
        elif event == cv2.EVENT_RBUTTONDOWN:
            done = True

    cv2.namedWindow("image", cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO)
    cv2.setWindowProperty("image", cv2.WND_PROP_ASPECT_RATIO, cv2.WINDOW_KEEPRATIO)
    cv2.setWindowProperty("image", cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
    cv2.setMouseCallback("image", trace_lines)
    while not done:
        i = image.copy()
        if not tracing and not traced_one:
            cv2.imshow("image", image)
        else:
            cv2.line(i, (x_start, y_start), (x_end, y_end), (255, 0, 0), 2)
            cv2.imshow("image", i)
        cv2.waitKey(5)
    xs1, xe1, ys1, ye1 = x_start, x_end, y_start, y_end
    cv2.destroyAllWindows()
    if ye1<ys1:
        xs1, xe1, ys1, ye1 = xe1, xs1, ye1, ys1
    return xs1-xe1, ys1-ye1


def get_vectors_reg(coef):
    """Crée des vecteurs à partir des coefficients de régresssion"""
    return 100, coef*100


def compute_angle_from_vectors(spray_vector_1, spray_vector_2 = (0,-1)):
    """Calcule l'angle entre deux vecteurs"""
    if spray_vector_2 == (0,-1):
        return math.acos(np.dot(spray_vector_1, spray_vector_2) / (
                    np.linalg.norm(spray_vector_1) * np.linalg.norm(spray_vector_2))) / (2 * math.pi) *
    else:
        return math.acos(np.dot(spray_vector_1, spray_vector_2) / (
                    np.linalg.norm(spray_vector_1) * np.linalg.norm(spray_vector_2))) / (2 * math.pi) *
```

```
##################GUI####################
import tkinter as tk
from tkinter import ttk
from PIL import ImageTk, Image


class GUI(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)
        self._frame = None
        self.switch_frame(menuframe)
        self.title("Calcul d'angle de spray")

    def switch_frame(self, frame_class):
        """Change de frame"""
        new_frame = frame_class(self)
        if self._frame is not None:
            self._frame.destroy()
        self._frame = new_frame

    def turnLeft(self):
        """Tourne les images vers la gauche"""
        global imageJet, imageBackground, imageNoise
        imageJet = imageJet.swapaxes(-2,-1)[...,::-1,:]
        if type(imageBackground) == numpy.ndarray:
            imageBackground = imageBackground.swapaxes(-2,-1)[...,::-1,:]
        if type(imageNoise) == numpy.ndarray:
            imageNoise = imageNoise.swapaxes(-2,-1)[...,::-1,:]
        self.switch_frame(resizeframe)

    def turnRight(self):
        """Tourne les images vers la droite"""
        global imageJet, imageBackground, imageNoise
        imageJet = imageJet.swapaxes(-2,-1)[...,::-1]
        if type(imageBackground) == numpy.ndarray:
            imageBackground = imageBackground.swapaxes(-2,-1)[...,::-1]
        if type(imageNoise) == numpy.ndarray:
            imageNoise = imageNoise.swapaxes(-2,-1)[...,::-1]
        self.switch_frame(resizeframe)

    def switch_to_resize_frame(self, frame_class, path1, path2, path3):
        global imageJet, imageBackground, imageNoise, imageStd, angle, coeffL, coeffR, interceptL, inte
        if (path1 != "Jet") and (path1 != ""):
            if (path2 != "Fond") and (path2 != "") and (path3 != "Noir") and (path3 != ""):
                imageJet, imageBackground, imageNoise = load_images(path1, path2, path3)
            else:
                imageJet = load_images(path1)
            self.switch_frame(frame_class)
        else:
            pass

    def resize_manual(self):
        global imageJet, imageBackground, imageNoise, imageStd, angle, coeffL, coeffR, interceptL, inte
        self._frame.destroy()
        if type(imageBackground) == numpy.ndarray:
            imageJet, imageBackground, imageNoise = crop_manual(imageJet, imageBackground, imageNoise)
        else:
```

```
            imageJet = crop_manual(imageJet)
        resized = True
        self._frame = resizeframe(self)

    def resize_auto(self):
        global imageJet, imageBackground, imageNoise, imageStd, angle, coeffL, coeffR, interceptL, inte
        if type(imageBackground) == numpy.ndarray:
            imageJet, imageBackground, imageNoise = cropCanny(imageJet, imageBackground, imageNoise)
        else:
            imageJet = cropCanny(imageJet)[0]
        resized = True
        self.switch_frame(resizeframe)

    def switch_to_result_frame(self):
        global imageJet, imageBackground, imageNoise, imageStd, angle, coeffL, coeffR, interceptL, inte
        if type(imageBackground) == numpy.ndarray:
            write_images_to_file(folder='Cropped', JetCropped=imageJet, BackgroundCropped=imageBackgrou
        else:
            write_images_to_file(folder='Cropped', JetCropped=imageJet)
        if type(imageBackground) == numpy.ndarray:
            imageJet = normalize(imageJet, imageBackground, imageNoise)
        else:
            imageJet = normalize(imageJet)
        write_images_to_file(folder='Normalized',Normalized=imageJet)
        imageBinarized = threshold(imageJet)
        write_images_to_file(folder='Binarized', Binarized=imageBinarized)
        imageStd = standard_deviation(imageBinarized)
        imageStd = imageStd.astype('uint8')
        write_images_to_file(folder='STD', STD=imageStd)
        imageGauche, imageDroite  = dichotomy(imageStd, center)
        coeffL, interceptL = linearRegression(imageGauche, adherence)
        coeffR, interceptR = linearRegression(imageDroite, adherence)
        vectorL = get_vectors_reg(coeffL)
        vectorR = get_vectors_reg(coeffR)
        angle = compute_angle_from_vectors(vectorL, vectorR)
        self.switch_frame(resultframe)

    def updateAdherence(self, value):
        global adherence
        adherence = value
        self.switch_to_result_frame()

    def recompute_regression(self, adherenceValue):
        global imageJet, imageBackground, imageNoise, imageStd, angle, coeffL, coeffR, interceptL, inte
        adherence = adherenceValue
        imageGauche, imageDroite = dichotomy(imageStd, center)
        coeffL, interceptL = linearRegression(imageGauche, adherence)
        coeffR, interceptR = linearRegression(imageDroite, adherence)
        vectorL = get_vectors_reg(coeffL)
        vectorR = get_vectors_reg(coeffR)
        angle = compute_angle_from_vectors(vectorL, vectorR)
        self.switch_frame(resultframe)

    def compute_angle_manual(self, imageStd):
        global angle, used_manual
        vectorL = get_vectors_manual(imageStd)
        vectorR = get_vectors_manual(imageStd)
        angle = compute_angle_from_vectors(vectorL, vectorR)
```

```
        used_manual = True
        self.switch_frame(resultframe)

    def define_center(self, imageJet):
        global center
        x_cir, y_cir = 0, 0
        done, traced_one = False, False
        image = imageJet
        def trace_circle(event, x, y, flags, *userdata):
            nonlocal x_cir, y_cir, image, done, traced_one
            if event == cv2.EVENT_LBUTTONDOWN:
                x_cir, y_cir = x, y
                traced_one = True
            elif event == cv2.EVENT_RBUTTONDOWN:
                done = True
        cv2.namedWindow("image", cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO)
        cv2.setWindowProperty("image", cv2.WND_PROP_ASPECT_RATIO, cv2.WINDOW_KEEPRATIO)
        cv2.setWindowProperty("image", cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
        cv2.setMouseCallback("image", trace_circle)
        while not done:
            i = image.copy()
            if not traced_one:
                cv2.imshow("image", image)
            else:
                cv2.circle(i, (x_cir, y_cir), 5, (255, 0, 0), -1)
                cv2.imshow("image", i)
            cv2.waitKey(5)
        cv2.destroyAllWindows()
        center = x_cir


class menuframe(ttk.Frame):
    global imageJet, imageBackground, imageNoise, imageStd, angle, coeffL, coeffR, interceptL, intercep
    def __init__(self, master):
        ttk.Frame.__init__(self, master, padding="3 3 12 12")
        self.focus_set()
        self.grid(column=0, row=0, sticky="N, W, E, S")
        ttk.Label(self, text="Veuillez entrer le chemin des images (nom seulement si elles sont dans le
            column=1, row=1, sticky="W, E")
        ttk.Label(self, text="Les images doivent être des multiples (.tiff) ou des séquences d'images,
            column=1, row=2, sticky="W, E")
        path1, path2, path3 = tk.StringVar(), tk.StringVar(), tk.StringVar()
        path1_entry = ttk.Entry(self, width=7, textvariable=path1)
        path1_entry.bind("<Return>", (lambda event: master.switch_to_resize_frame(resizeframe, path1.ge
        path1_entry.grid(column=1, row=3, sticky="W, E")
        path1_entry.insert(0, "Jet")
        path2_entry = ttk.Entry(self, width=7, textvariable=path2)
        path2_entry.grid(column=1, row=4, sticky="W, E")
        path2_entry.insert(0, "Fond")
        path3_entry = ttk.Entry(self, width=7, textvariable=path3)
        path3_entry.bind("<Return>", (lambda event: master.switch_to_resize_frame(resizeframe, path1.ge
        path3_entry.grid(column=1, row=5, sticky="W, E")
        path3_entry.insert(0, "Bruit")
        for child in self.winfo_children():
            child.grid_configure(padx=5, pady=5)
        ttk.Button(self, text="Suivant", command=lambda event: master.switch_to_resize_frame(resizefram
```

```
class resizeframe(ttk.Frame):
    global imageJet, imageBackground, imageNoise, imageStd, angle, coeffL, coeffR, interceptL, intercep
    def __init__(self,master):
        global imageJet, imageBackground, imageNoise, imageStd, angle
        ttk.Frame.__init__(self, master, padding="3 3 12 12")
        self.focus_set()
        self.grid(column=0, row=0, sticky="N, W, E, S")
        width, height = pyautogui.size()
        y, x = imageJet[0].shape
        self.imgJet = ImageTk.PhotoImage(image=Image.fromarray(imageJet[0]).resize((int(x*(height-500)/
        ttk.Button(self, text="Tourner vers la gauche",
                   command=lambda: master.turnLeft()).grid(
            column=1, row=2, sticky="W, E")
        ttk.Button(self, text="Tourner vers la droite",
                   command=lambda: master.turnRight()).grid(
            column=2, row=2, sticky="W, E")
        ttk.Label(self, image=self.imgJet).grid(
            column=1, row=1, columnspan=2, sticky="W, E")
        ttk.Button(self, text="Redimensionner manuellement",
                   command=lambda: master.resize_manual()).grid(
            column=1, row=4, columnspan=2, sticky="W, E")
        ttk.Button(self, text="Redimensionner automatiquement",
                   command=lambda: master.resize_auto()).grid(
            column=1, row=5, columnspan=2, sticky="W, E")
        if resized:
            self.bind("<Return>", (lambda event: master.switch_frame(centerframe)))
            ttk.Button(self, text="Suivant",
                   command=lambda: master.switch_frame(centerframe)).grid(
            column=1, row=6, columnspan=2, sticky="W, E")
        else:
            self.bind("<Return>", (lambda event: master.resize_auto()))


class centerframe(ttk.Frame):
    global imageJet, imageBackground, imageNoise, center
    def __init__(self,master):
        global imageJet, imageBackground, imageNoise, imageStd, angle, center
        ttk.Frame.__init__(self, master, padding="3 3 12 12")
        self.focus_set()
        self.grid(column=0, row=0, sticky="N, W, E, S")
        width, height = pyautogui.size()
        y, x = imageJet[0].shape
        center = int(x / 2)
        self.imgJet = ImageTk.PhotoImage(image=Image.fromarray(imageJet[0]).resize((int(x * (height - 5
        ttk.Label(self, image=self.imgJet).grid(column=1, row=1, sticky="W, E")
        ttk.Button(self, text="Redéfinir le centre de l'image", command=lambda: master.define_center(im
            column=1, row=2, sticky="W, E")
        self.bind("<Return>", (lambda event: master.switch_to_result_frame()))
        ttk.Button(self, text="Suivant",
                   command=lambda: master.switch_to_result_frame()).grid(
            column=1, row=6, columnspan=2, sticky="W, E")


class resultframe(ttk.Frame):
    global imageJet, imageBackground, imageNoise, imageStd, angle, coeffL, coeffR, interceptL, intercep
    def __init__(self, master):
        ttk.Frame.__init__(self, master, padding="3 3 12 12")
        self.grid(column=0, row=0, sticky="N, W, E, S")
```

```
        self.focus_set()
        adherenceSlider = tk.IntVar()
        adherenceSlider.set(round(np.log10(adherence)))
        width, height = pyautogui.size()
        y, x = imageStd.shape
        imageStdWithRegression = imageStd.copy()
        cv2.line(imageStdWithRegression, (int(coeffL * 0 + interceptL), 0), (int(coeffL * y + intercept
        cv2.line(imageStdWithRegression, (int(coeffR * 0 + interceptR + center), 0), (int(coeffR * y +
        write_images_to_file(folder='WithReg', WithReg = imageStdWithRegression)
        self.imgStdWithRegression = ImageTk.PhotoImage(
            image=Image.fromarray(imageStdWithRegression).resize((int(x * (height - 500) / y), height -
        ttk.Label(self, image=self.imgStdWithRegression).grid(
            column=1, row=1, rowspan=6, sticky="W, E")
        if used_manual:
            ttk.Label(self, text=f"L'angle du spray, calculé manuellement, est de {angle}°.").grid(
            column=2, row=1, sticky="W, E")
        else:
            ttk.Label(self, text=f"L'angle du spray, calculé automatiquement, est de {angle}°.").grid(
            column=2, row=1, sticky="W, E")
        ttk.Label(self, text=f"Adhérence au haut du spray{' : Angle théorique' if adherenceSlider.get()
            column=2, row=3, sticky="W, E, S")
        ttk.Scale(self, from_=0, to_=5, variable=adherenceSlider).grid(column=2, row=4, sticky="N, W, E
        ttk.Button(self, text="Recalculer",
                   command=lambda: master.recompute_regression(10**int(adherenceSlider.get()))).grid(
            column=2, row=5, sticky="W, E")
        self.bind("<Return>", (lambda event: master.recompute_regression(10**int(adherenceSlider.get())
        ttk.Button(self, text="Recalculer manuellement", command=lambda: master.compute_angle_manual(im
            column=1, row=7, columnspan=2, sticky="W, E, S")



if __name__ == '__main__':
    imageJet, imageBackground, imageNoise, imageStd = 0, 0, 0, 0
    angle, coeffL, coeffR, interceptL, interceptR, center = 0, 0, 0, 0, 0, 0
    resized, used_manual = False, False
    adherence = 5
    GUI = GUI()
    GUI.mainloop()
```