**Image Processing Toolbox**

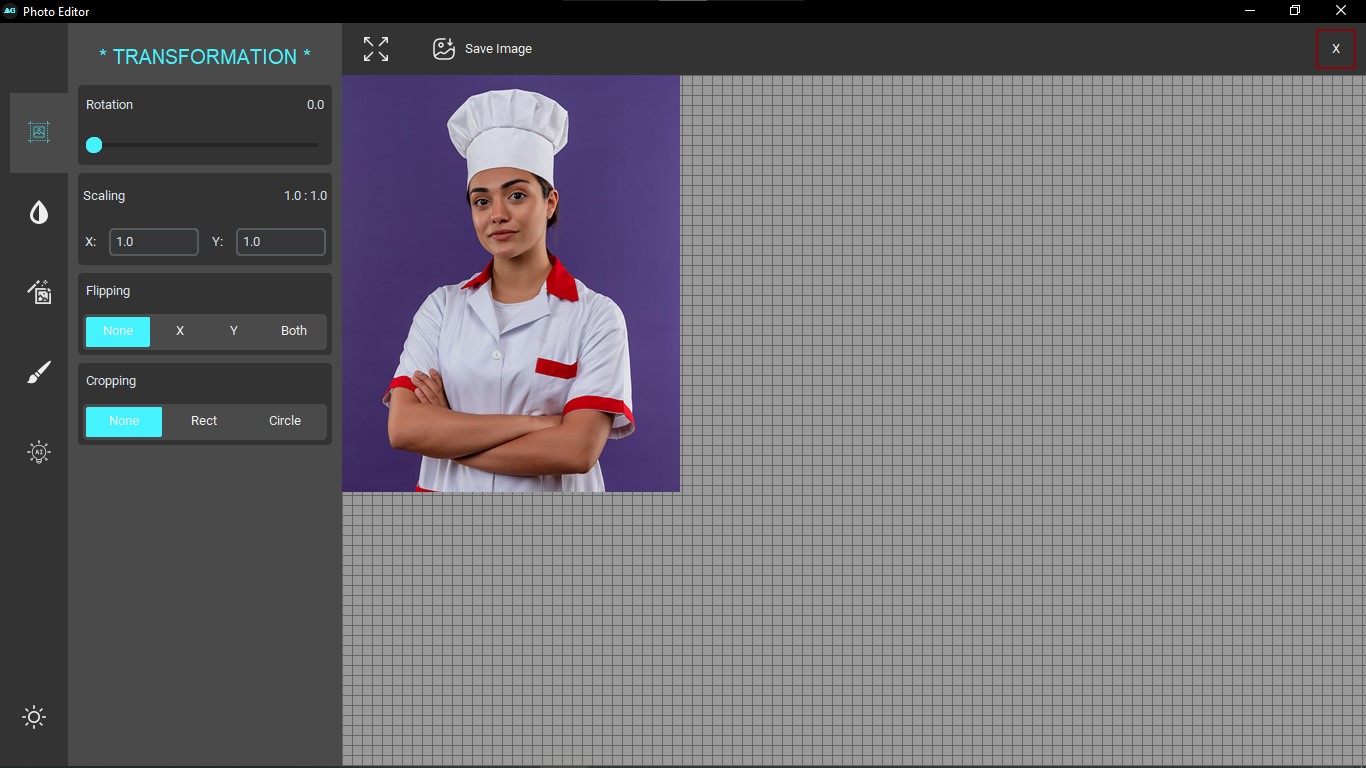
**SN: Abdulrhman Sayed Goda**

**ID: 4200802**

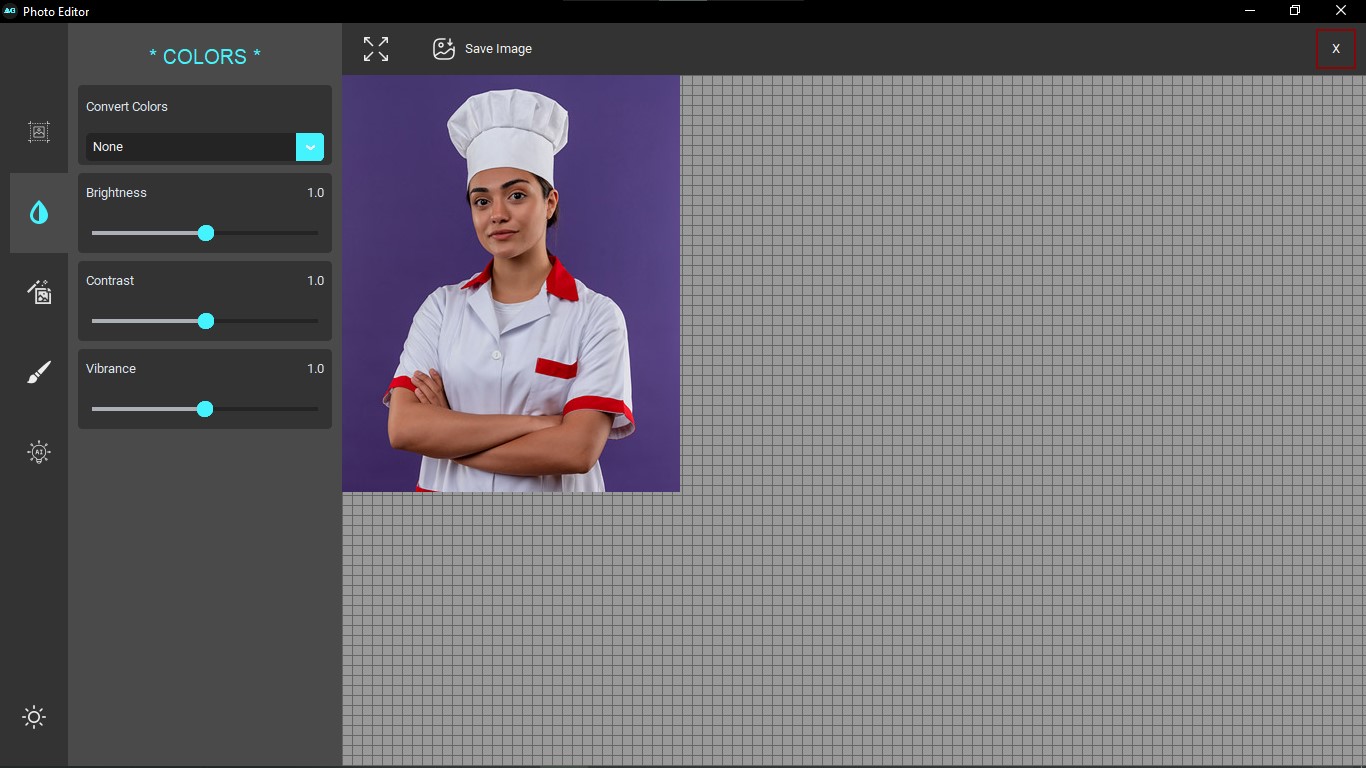
**Layout of the Toolbox**

2.Window To Load The Image

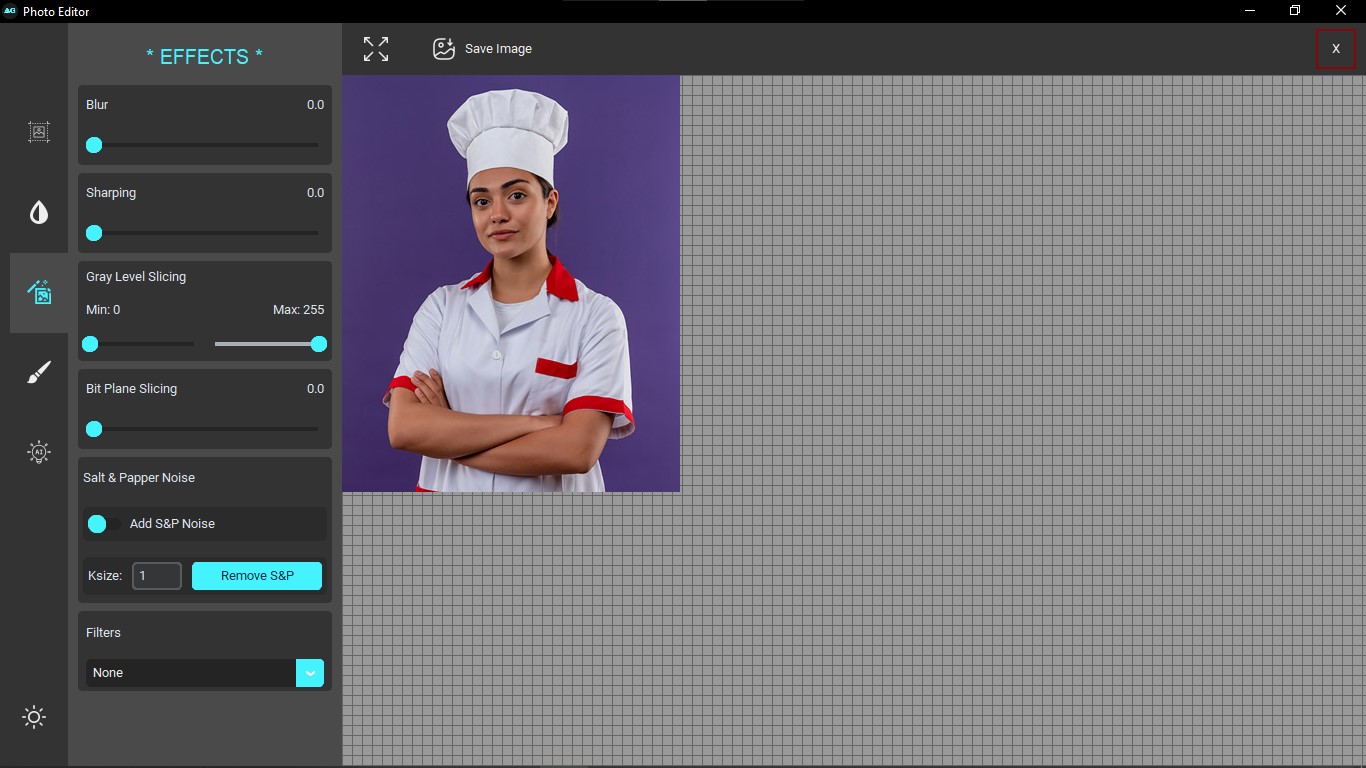
1.Start Window Loader

****

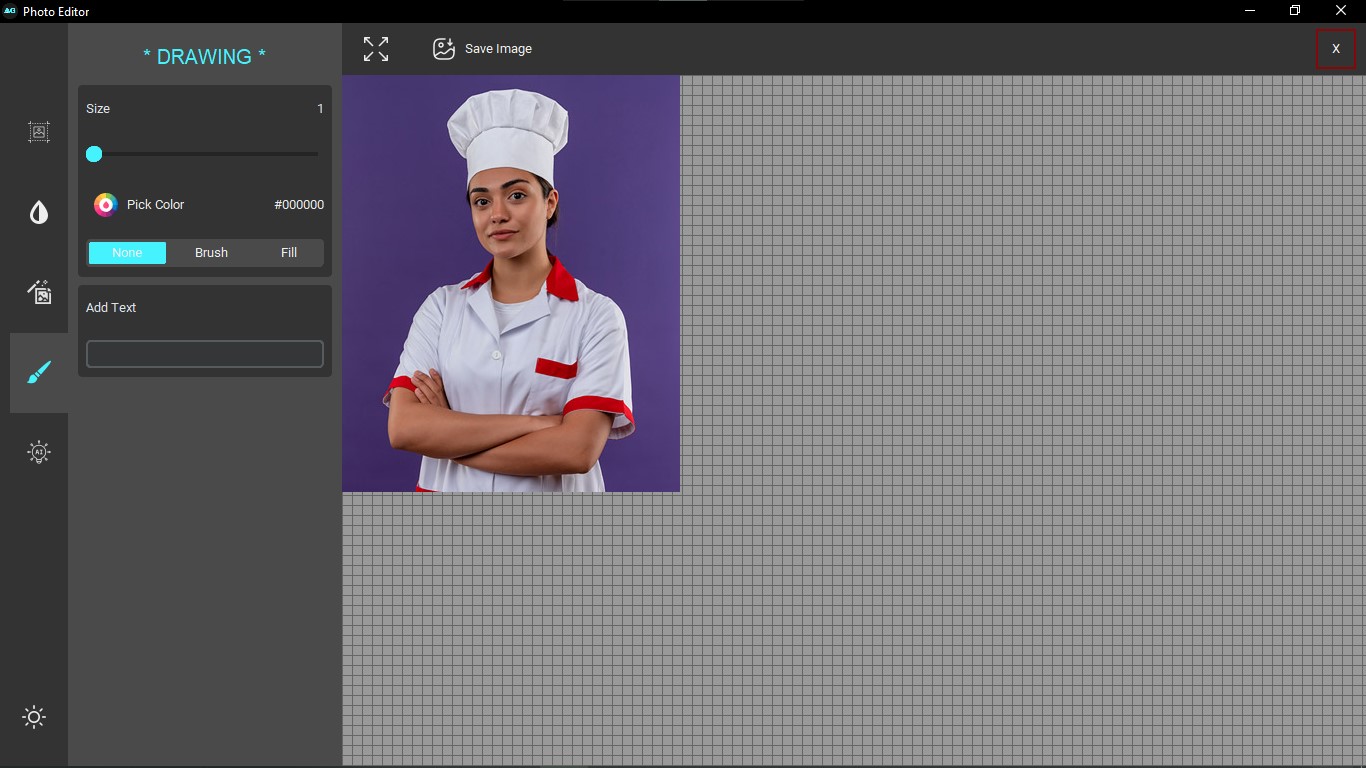
3.Transformation Control Widgets

****

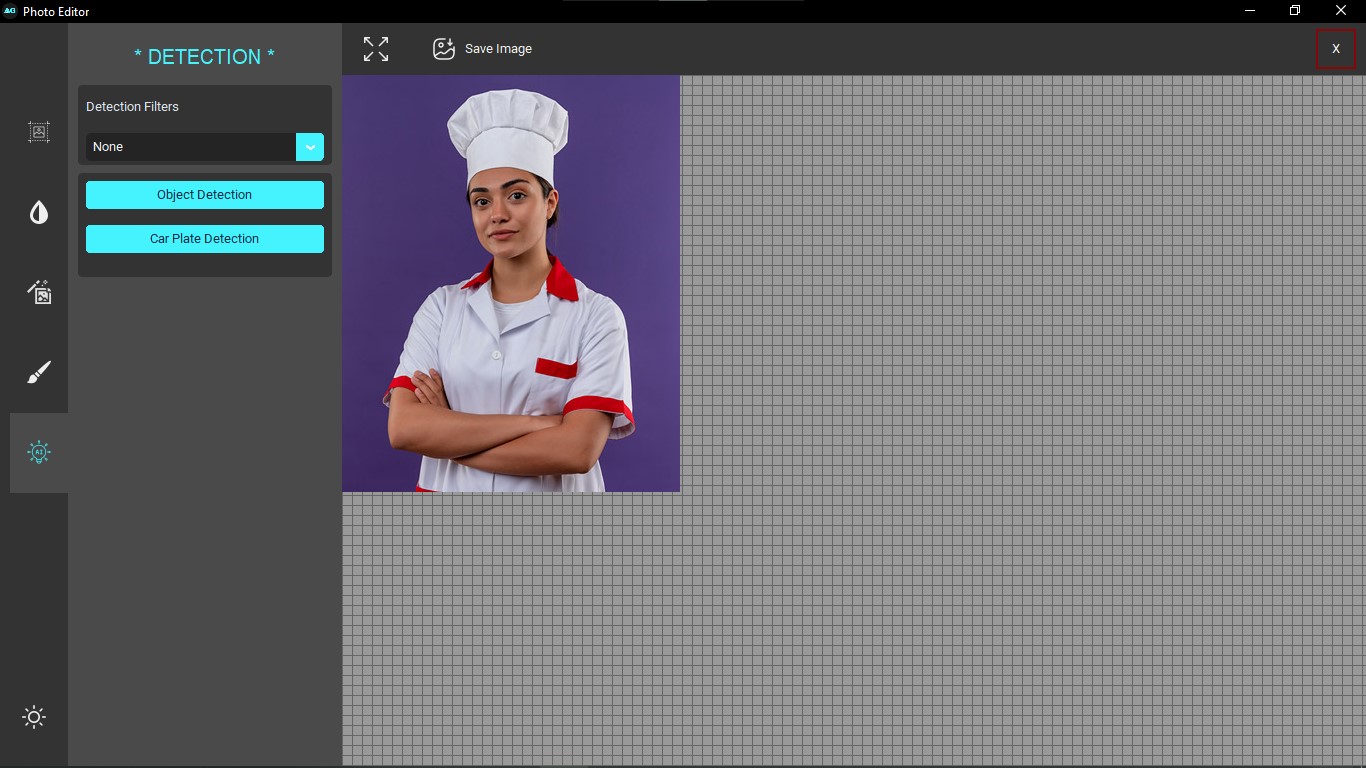
4.Color Control Widgets

****

5.Effects Control Widgets

****

6.Drawing Control Widgets

****

7.Detection Control Widgets

**Functions of the Toolbox**

1. **Rotation:**

Adjust the values of angle and other parameters based on your specific requirements.

center = (image.shape[1] // 2, image.shape[0] // 2)

rotation\_matrix = cv2.getRotationMatrix2D(center, data\_vars['rotate'][0].get(), scale=1.0)

image = cv2.warpAffine(image, rotation\_matrix, (.image.shape[1], image.shape[0]))

1. **Flip:**

You can choose the appropriate flip operation based on your specific requirements. The cv2.imshow functions are used to display the original and flipped images.

flip\_var = data\_vars['flip'][0].get()

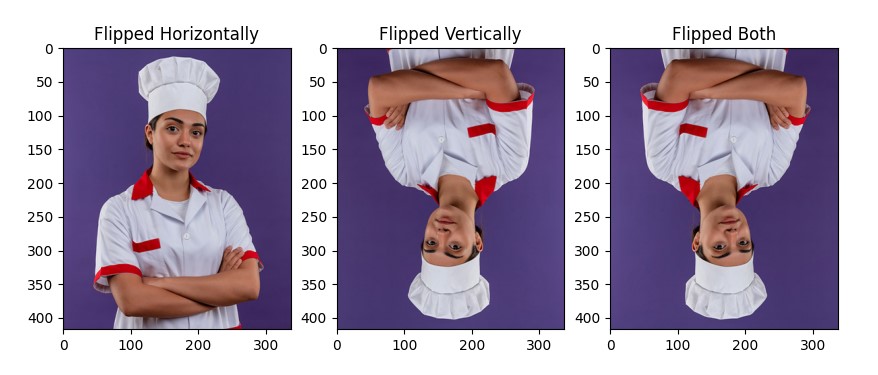
if flip\_var == 'X':

 image = cv2.flip(image, 1)

elif flip\_var == 'Y':

 image = cv2.flip(image, 0)

elif flip\_var == 'Both':

 image = cv2.flip(image, -1)

1. **Scaling:**

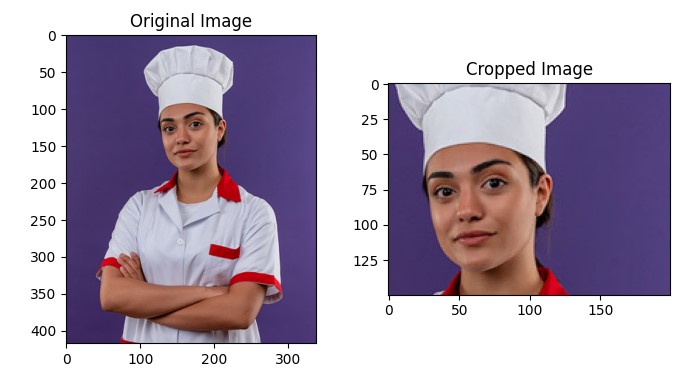
Adjust the desired\_width and desired\_height based on your specific requirements. You can also specify the scaling factors directly instead of providing the target width and height.

A person in a chef's uniform

Description automatically generatedimage = cv2.resize(image, (0, 0), fx= data\_vars['scale\_x'][0].get(), fy= data\_vars['scale\_y'][0].get())

1. **Cropping:**

By Drawing a rectangle or circle based on buttons the user selects to choose mode, the coordinates of these shapes will apply crop on the image.

****image[start\_y:end\_y, start\_x:end\_x]

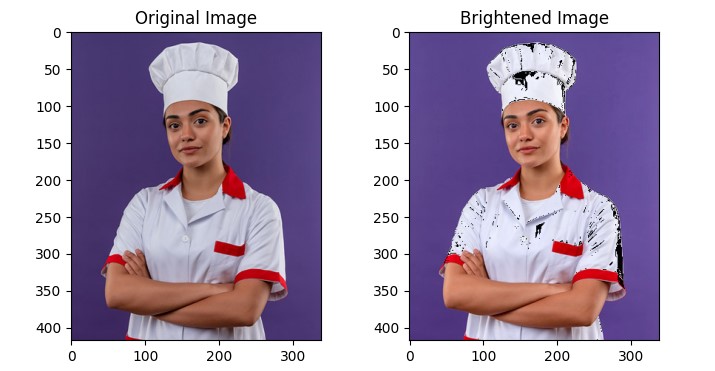
1. **Brightness:**

In OpenCV, you can change the brightness of an image by adjusting the pixel values.

img\_float = image.astype(np.float32) / 255.0

img\_float = img\_float + data\_vars['brightness'][0].get() / 100.0 # -100 : 100

img\_float = np.clip(img\_float, 0.0, 1.0)

image = (img\_float \* 255).astype(np.uint8)

1. **Contrast:**

by appling CLAHE to the lightness channel of an image in the LAB color space, enhancing the contrast locally. This technique is commonly used to improve the visual quality of images, especially in cases where the original image has low contrast.

lab = cv2.cvtColor(self.image, cv2.COLOR\_RGB2LAB)

        l, a, b = cv2.split(lab)

        clahe = cv2.createCLAHE(clipLimit=self.data\_vars['contrast'][0].get(), tileGridSize=(8, 8))

        cl = clahe.apply(l)

        limg = cv2.merge((cl, a, b))

        self.image = cv2.cvtColor(limg, cv2.COLOR\_LAB2RGB)

1. **Vibrance:**

Vibrance is often associated with the intensity of colors in an image without affecting skin tones. By manipulating the color channels in a color space like LAB, you can control the vibrance.

img\_float = image.astype(np.float32) / 255.0

vibrance = data\_vars['vibrance'][0].get()

vibrance\_matrix = np.array([[1 + vibrance, 0, 0], [0, 1 + vibrance, 0], [0, 0, 1 + vibrance]])

img\_float = cv2.transform(img\_float, vibrance\_matrix)

img\_float = np.clip(img\_float, 0.0, 1.0)

A collage of a person in a chef's uniform

Description automatically generatedimage = (img\_float \* 255).astype(np.uint8)

1. **Blur:**

The purpose of changing the blur can include noise reduction, image smoothing, and preparation for further processing.

kernel\_size = (5, 5)

blurred\_image = cv2.GaussianBlur(image, kernel\_size, 0)

1. **Sharpening:**

You can achieve a sharpening effect by applying a kernel to the image using convolution. One common sharpening filter is the Laplacian filter.

The purpose of sharpening an image is to enhance edges and fine details, making them more prominent.

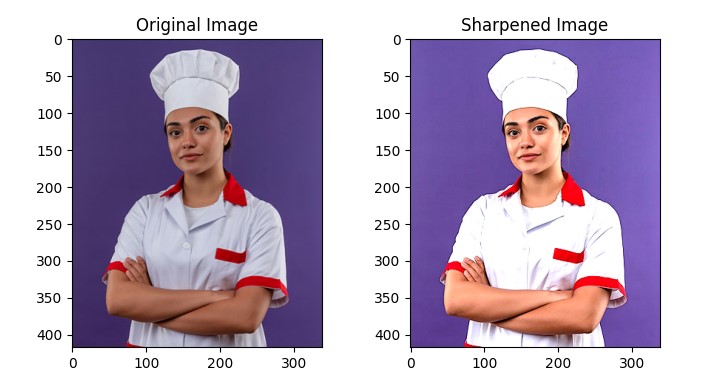
# Convert the image to grayscale

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Apply the Laplacian filter for sharpening

sharpened\_image = cv2.Laplacian(gray\_image, cv2.CV\_64F)

# Convert back to uint8

sharpened\_image = np.uint8(np.abs(sharpened\_image))

1. **Gray Level Slicing (intensity slicing or contrast stretching):**

Used to enhance the visibility of specific intensity ranges in an image.

min\_intensity = 100

max\_intensity = 200

# Create a binary mask for the specified intensity range

binary\_mask = cv2.inRange(image, min\_intensity, max\_intensity)

# Apply the binary mask to the original image

result\_image = cv2.bitwise\_and(image, image, mask=binary\_mask)

****

1. **Bit Plane Slicing:**

Is a technique in image processing to convert gray scale image into a binary image.

Used in image compression (representing an image with fewer bits).

gray = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)

bit\_plane = (gray >> data\_vars['bit\_plane\_slicing'][0].get()) & 1

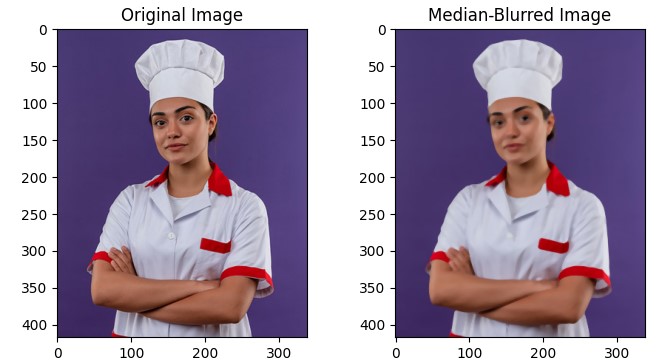
self.image = bit\_plane \* 255

1. **Median Blur:**

Used to reduce the impact of salt-and-pepper noise while preserving edges and details in an image.

The median blur replaces each pixel value with the median value of its neighboring pixels, which is effective in removing outliers caused by the noise.

image = cv2.medianBlur(image, ksize=(3,3))

1. **Emboss Filter:**

Filter or convolution operation that is applied to an image to create a three-dimensional (3D) effect by highlighting the edges. Used to enhance edges by making them more prominent.

kernel\_emboss = np.array([[0, -1, -1],[1,  0, -1],[1,  1,  0]])

image = cv2.filter2D(image, -1, kernel\_emboss)

1. **Canny Detection:**

It’s An edge detection operator that aims to identify edges in an image. It helps identify regions of rapid intensity change, which often correspond to object boundaries.

blurred = cv2.GaussianBlur(image, (3, 3), 0)

edges = cv2.Canny(blurred, 50, 150)

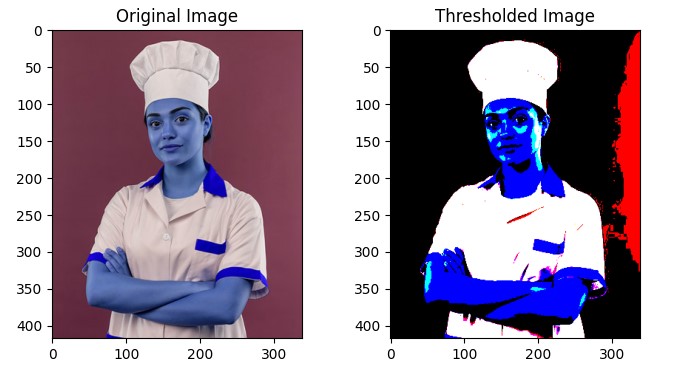
A person in chef uniform and a picture of a chef

Description automatically generatedimage = cv2.cvtColor(edges, cv2.COLOR\_GRAY2RGB)

1. **Thresholding:**

Used to converting an image into a binary (black and white) image based on pixel intensity values.

Helpful in (Image Segmentation, Object Detection, Noise Reduction, Edge Detection)

\_, image = cv2.threshold(image, 128, 255, cv2.THRESH\_BINARY)

1. **Contour:**

Represent the boundaries of objects or regions within an image. A contour is a curve joining all the continuous points along a boundary that have the same color or intensity.

Used In (Object Detection, Shape Analysis, Shape Recognition)

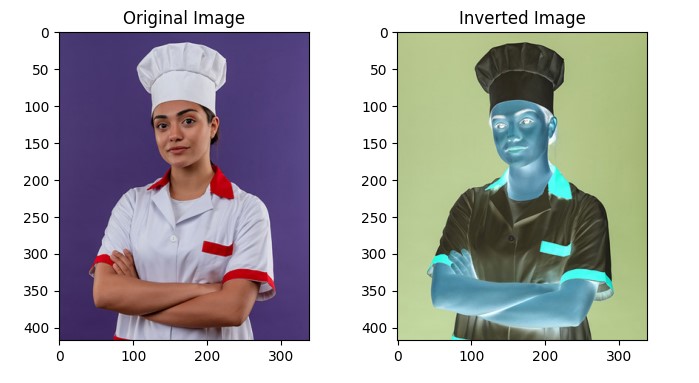
\_, thresh = cv2.threshold(gray, 128, 255, cv2.THRESH\_BINARY)

contours, \_ = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cv2.drawContours(image, contours, -1, (0, 255, 0), 2)

1. **Inverting:**

Is reversing the intensity values of each pixel. The purpose of inverting an image can vary depending on the specific application or task.

image = 255 - image

1. **Filter2D:**

Is a function in the OpenCV library that performs 2D convolution, a fundamental operation in image processing and computer vision. Convolution involves sliding a kernel (a small matrix or filter) over an input image and computing the weighted sum of pixel values at each position.

Used In (Image Filtering, Feature Detection, Image Enhancement)

kernel = np.ones((3, 3), np.int8) / 9

image = cv2.filter2D(image, cv2.CV\_8UC1, kernel)

1. **Histogram Equalization:**

It is a method which applied to a dark or washed out images in order to improve image contrast by spreading out the frequencies of the image.

image = cv2.equalizeHist(gray)

1. **Logarithmic Transformation:**

Used to map a narrow range of dark input values into a wider range of output values.

The transformation function used is 𝒔=𝒄∗𝒍𝒐𝒈𝒆(𝟏+𝒓)

l\_t = 1 \* np.log1p(gray)

image = (255 \* (l\_t - np.min(l\_t)) / (np.max(l\_t) - np.min(l\_t))).astype(np.uint8)

1. **Power Transformation (Gamma Correction):**

Used to map a narrow range of dark input values into a wider range of output values or vice versa depending on γ value

The transformation function used is 𝒔=𝒄∗𝒓𝜸

p\_t = np.power(gray, 0.5)

image = (255 \* (p\_t - np.min(p\_t)) / (np.max(p\_t) - np.min(p\_t))).astype(np.uint8)

1. **Draw:**

Using Canvas properties to draw connected lines over the image, and the user can be apple to choose size of the line and color also.

for lis in draw\_data:

    for i in range(0, len(lis)-1):

        cv2.line(image, lis[i], lis[i+1], color, draw\_size.get())

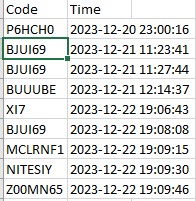
1. **Put Text:**

Using Canvas properties to put text over the image, and the user can be apple to choose text size and the color, and the coordinates to place.

cv2.putText(image, data\_vars['text'][0].get(),text\_position, cv2.FONT\_HERSHEY\_SIMPLEX, draw\_size.get(), color, 2)

1. **OCR (Optical Character Recognition):**

is commonly used for license plate recognition in the context of Automatic License Plate Recognition (ALPR) systems. ALPR systems use computer vision techniques and OCR to automatically extract text information from license plates in images or video streams.



Steps involved in license plate detection and recognition using OCR:

1. Image Acquisition:

Obtain an image containing one or more vehicles with visible license plates.

1. Preprocessing:

Apply preprocessing techniques to enhance the image quality and improve the accuracy of subsequent processing steps. Common preprocessing steps include:

* Resize the image to a standard size.
* Convert the image to grayscale.
* Apply image smoothing or denoising.
* Adjust the image contrast and brightness.

1. License Plate Localization (Detection):

Use object detection or image segmentation techniques to locate and extract regions of interest (ROI) in the image that likely contain license plates.

1. Character Segmentation:

For each detected license plate region, segment individual characters or character groups. This step is crucial for OCR accuracy.

Techniques may include contour analysis, connected component analysis, or deep learning-based segmentation.

1. Character Recognition (OCR):

Apply OCR algorithms to recognize the characters in each segmented region. OCR software or libraries (such as Tesseract OCR) are commonly used for this task.

Train the OCR model on a dataset of license plate characters to improve accuracy, especially if the license plate font is non-standard.

1. Output:

Provide the final recognized license plate number as output in image with the code as a text, and also in excel sheet contains each car license detected and the time for detection.

1. **Object Detection:**

Using YOLO object detection and opencv preprocessing method to apply object detection on images that might contain objects.

The output of this process will result as an image with boundary box that represents the object, and at the top of the box the object name and the accuracy of the process will be placed.



**Image Segmentation:**

Image segmentation is a process of partitioning an image into multiple segments or regions based on certain characteristics, such as pixel intensity, color, texture, or other visual properties.

The goal of image segmentation is to group pixels that share similar attributes and represent meaningful parts or objects within the image.

Image segmentation is an important step in various computer vision and image processing applications, including object recognition, scene understanding, and medical image analysis.

Image segmentation Techniques:

* Thresholding
  + Simple thresholding: Pixels with intensity values above or below a certain threshold are classified into different segments.
  + Adaptive thresholding: Threshold values are locally adjusted based on the pixel neighborhood.
* Contour-Based Segmentation
  + Identify and extract contours in an image, representing boundaries of objects or regions.
  + Useful for scenarios where objects have distinct boundaries.
* K-Means Clustering
  + Group pixels into clusters based on their color similarity.
  + Effective for color image segmentation.