Computer Vision Final Project

PixCraft

AI-Enhanced Image Stitching and Edge Detection

By: Meqdad Darwish

Overview

Welcome to **PixCraft** (Unleash the Power of Pixels), where the fusion of creativity and technology embarks on an immersive journey into the realm of image processing excellence. Seamlessly integrating GUI libraries like Tkinter with OpenCV's robust algorithms and YOLO on the backend, PixCraft offers a seamless user experience.

With PixCraft, uploading images is effortless, allowing users to unlock a myriad of transformations with just a few clicks. Embark on a visual odyssey as you delve into various techniques, including edge detection using Difference of Gaussian, Canny Edge Detection, HSV Model, Human Detection, and more, all within an intuitive and refined interface.

Technical Stack

1. **Back-end:**
    a. OpenCV Algorithms for Edge Detection
    b. YOLOv8 Algorithm for Human Detection
    c. Numpy for Applying Filters on Images
2. **Front-end for User Interface:**
    a. Tkinter GUI Library for Building Interfaces
    b. PILLOW for Viewing Images

Algorithms and Features

In this project, I have crafted numerous image processing and computer vision algorithms, presenting them through straightforward interfaces that are convenient and user-friendly.

My project involves taking a set of images selected by the user, performing feature matching and stitching to create a single stitched image. Upon successful completion of the process, it provides the user with various operations that can be applied to the stitched image.

Implemented algorithms and available features in PixCraft for now:

- **Canny Edge Detection Algorithm**

  This feature utilizes the Canny edge detection algorithm to identify edges within images, aiding in various image processing tasks such as object recognition and boundary detection.

- **Difference of Gaussian (DoG) with morphological close operation and Slider**

  This functionality incorporates the Difference of Gaussian (DoG) method, enhanced with a morphological close operation and a slider to adjust the kernel size. It facilitates precise control over image smoothing and edge detection, contributing to improved feature extraction and analysis.

- **HSV Color Model**

  This feature enables the user to view images based on the Hue, Saturation, and Value (HSV) color model.

- **Human Detection using YOLO v8 algorithm**

  This feature integrates the YOLO v8 algorithm for human detection, allowing the identification and localization of human subjects within images. By leveraging deep learning techniques, it offers robust and efficient detection capabilities, essential for applications like surveillance, crowd monitoring, and human-computer interaction.

Project Structure

Throughout this project, I crafted the structure while developing its features, resulting in the following structure:

```
PixCraft/
│
├── backend/
│   ├── __init__.py
│   ├── imageProcessor.py
│   ├── imageStitcher.py
│   ├── utils.py
│   └── yolo_human_detection.py
│
├── gui/
│   ├── __init__.py
│   ├── imageSelector.py
│   ├── stitchedImage.py
│
└── app.py
```

Within this structure, the backend directory accommodates the image processing and computer vision algorithms, whereas the GUI directory hosts the user interfaces. Ultimately, app.py serves as the entry point for executing the entire application.

Here are concise descriptions for the role of each file in the provided structure:

- **`[backend] imageProcessor.py`**: This file is responsible for executing various image processing operations, including Canny edge detection, Difference of Gaussian, and HSV color model.
- **`[backend] imageStitcher.py`**: This file executes the image stitching operation. It verifies if there is more than one image available, conducts feature matching implicitly, and proceeds to stitch the images into a single stitched image. If only one image is found, it displays an error message to the user, explaining the cause of the error.
- **`[backend] utils.py`**: This file handles essential utilities required throughout the project, including displaying error, warning, and informational messages as needed. Additionally, it provides useful functions such as locating images within a directory based on file extensions like .jpg, .png, and others (it is essential for the human detection functionality).
- **`[backend] yolo_human_detection.py`**: This file is tasked with implementing the human detection process using the YOLO algorithm. It accepts an image and returns it with detected humans, along with bounding boxes and confidence percentages if any are found.
- **`[gui] imageSelector.py`**: This file facilitates user selection of images from their device and displays them within the same interface. Additionally, it enables users to delete selected images if necessary and initiates the stitching process. Furthermore, it manages the removal of output files after completing the human detection algorithm to maintain a clean project directory containing only essential files.
- **`[gui] stitchedImage.py`**: This file manages user interaction with the stitched image, presenting available operations for the user to execute. It displays the stitched image and provides various buttons for performing image processing and computer vision tasks, including Canny edge detection, Difference of Gaussian, HSV color model, and human-based detection. Additionally, it offers the option to save the stitched image if desired by the user.
- **`app.py`:** This file serves as the app's entry point, initiating by calling **`imageSelector.py`** to enable user image selection, and subsequently triggering the image stitching process, among other tasks.

Challenges and Design Decisions

Choosing Tkinter for Desktop App Development: Overcoming Tech Stack Challenges

The initial challenge involved deciding on the appropriate tech stack. I had two options to consider: embarking on web app development using a framework like Flask or FastAPI, or pursuing desktop app development utilizing popular Python GUI tools such as Tkinter or PyQt.

Ultimately, I opted for desktop app development using Tkinter for several reasons: Firstly, Tkinter is bundled with Python's standard libraries, making it a readily available component post-Python installation on any PC. Secondly, it offers a straightforward development process and provides a fast-performing environment. Lastly, running the project is simple; just execute the code without the need for an internet connection or launching a web browser.

Evolution of Project Structure: From Single File to Organized Framework

Deciding on the optimal project structure was a progressive journey. Initially, I began with a single file encompassing all components. As I delved into the implementation of the main window and researched image selection features, I realized the need for a more organized approach. This led to the creation of a separate window for stitched images and prompted the consideration of segregating image processing functionalities from user interface development.

Subsequently, as the functionalities within the stitched image window expanded, I recognized the necessity to separate it from the image selection view. In parallel, I started integrating buttons into the stitched image window while concurrently developing backend functionalities in the backend directory. Additionally, to improve code reusability and scalability, I refactored certain functionalities into classes, leveraging object-oriented programming (OOP) principles. Recognizing the recurring need for error messaging, I consolidated error management into a utils.py file, progressively augmenting its functions.

After navigating through this developmental process, the project structure outlined above emerged.

Optimizing Human Detection: From Algorithm Selection to Implementation

In this task, extensive research was conducted into human detection processes, involving testing multiple algorithms such as MobileNet SSD v2, YOLO v5, and YOLO v8.1. Through experimentation, it was determined that simplifying human detection could be achieved effectively using YOLO v8.1, which exhibited high accuracy on test images. Additionally, it was observed that YOLO v8.1 automatically stores resulting images in a predefined path, specifically in the "runs/detect/predict" directory. To streamline this process, code was implemented to locate and display images from this location, along with code to remove auto-generated directories upon program window closure.

Furthermore, a class was developed to handle human detection implementation, including a method to determine the presence of persons in the provided image. In instances where no persons are detected, a warning message is displayed to the user, ensuring informative feedback throughout the detection process.

Edge Detection Parameters

In Canny edge detection, I determined the upper and lower threshold values by calculating the median value of the image pixels and then adjusting one standard deviation to the left and right, adhering to the Gaussian distribution:

```
median = np.median(gray_image)

lower_edge = median * 0.68

upper_edge = median * 1.32
```

This approach ensures adaptability of the threshold values across various images. Unlike visual selection methods such as histogram-based approaches, which may yield satisfactory results for some images but not for others, this method provides a more consistent and reliable outcome.

In Difference of Gaussian (DoG) algorithm, after conducting numerous tests on various images, I settled on the following parameter values:

```
first_sigma = 3
```

```
gaussian_kernel_1 = (19, 19)

second_sigma = 5

gaussian_kernel_2 = (31, 31)
```

These sigma values were chosen to achieve sharp edges for image objects while avoiding excessive blurring. Larger sigma values resulted in overly blurred edges. Therefore, these specific sigma values were deemed optimal. As for the kernel size, I adhered to the recommended approach of setting it to six times the sigma value, ensuring it was an odd number.

Afterward, I implemented the code for morphological opening and closing. However, I realized the importance of allowing users to choose the preferable operation for the stitched image. Through several tests, I found it necessary to enable users to assess the output of both operations. Consequently, I devised a mechanism whereby the user determines the morphological operation to be applied. By default, the operation is set to morphological opening to remove noise from the background, but users have the option to modify this selection as desired.

Error Handling and User Experience

While testing certain features, it became apparent that users might encounter errors, such as selecting an incorrect kernel size for the DoG algorithm using the slider. This would trigger an error in the program, rendering it unable to apply the filter to the image. To address this issue, I implemented a checking mechanism to ensure the parameters are valid before applying the filter.

Furthermore, I introduced a visual cue for users utilizing the kernel size slider, displaying a note label in red to indicate that even numbers should not be used for this task. In the event that a user inadvertently selects an incorrect number, the program displays an error message explaining the issue and redirects the user back to the slider to select the appropriate number.

Additionally, I've addressed anticipated errors, such as attempting to stitch only one image or encountering failures during feature matching and stitching. In such cases, I've implemented

error messages to elucidate the cause and prompt the user to select more than one image or inform them of the stitching failure.

Moreover, I've incorporated informational messages for successful operations. For instance, upon saving an image, an informational message confirms the successful completion of the operation.

Furthermore, I've accounted for scenarios where users may attempt human detection on images devoid of humans. In these instances, a warning message is displayed, indicating the absence of humans with a confidence level above 50%.

How to Scale PixCraft with Extra Features

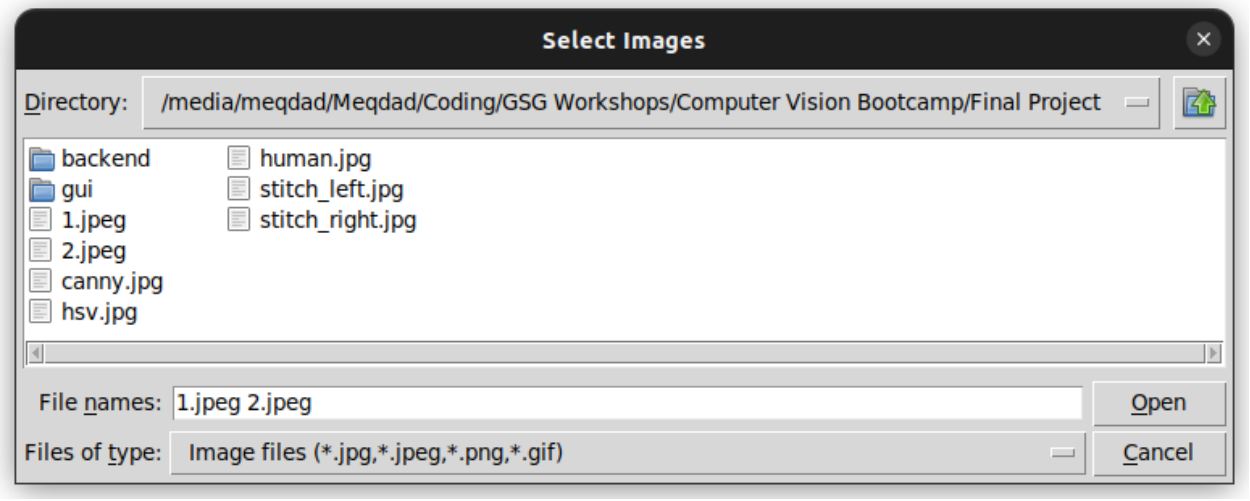
With the described project structure, incorporating or modifying features is straightforward. Throughout the project development, I prioritized scalability and flexibility to facilitate ease of implementation for both myself and potential collaborators.

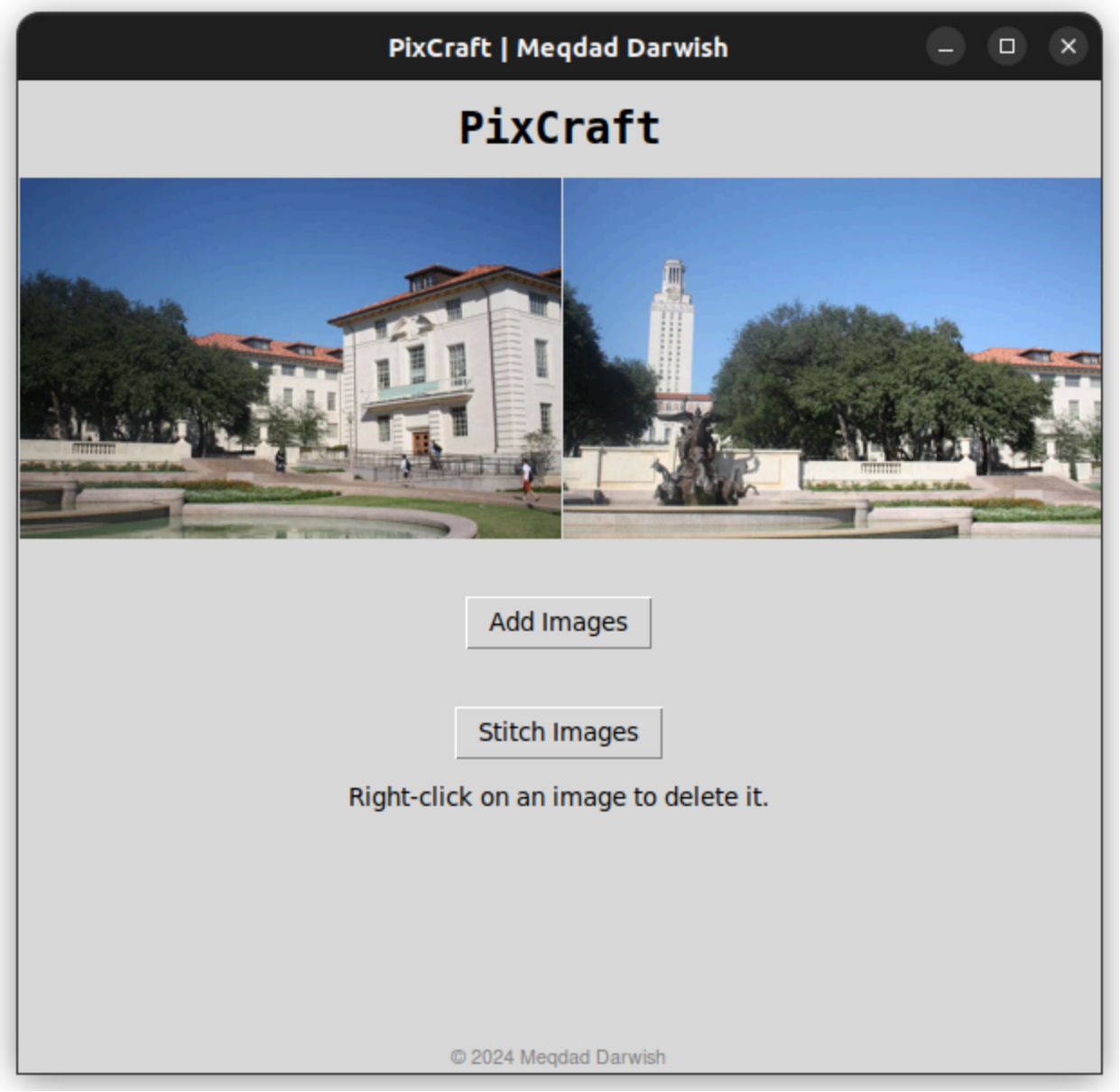
For instance, let's consider the process of adding a binarization feature to the program:

- Implement the image processing function for binarization in **`imageProcessor.py`**
- Introduce a button in **`stitchedImage.py`** to initiate the binarization process.
- Create a method within the **`stitchedImageApp`** class to call the image processing function.
- Establish a link between the button and the method, like this example: `command=self.apply_binarization`
- Ensure the method `_prepare_image_view()` is appropriately configured to handle the image and window title.

Utilize existing feature codes as a reference template for implementing new features, and remember to take a look at the docstrings of classes and functions for guidance on passing arguments.

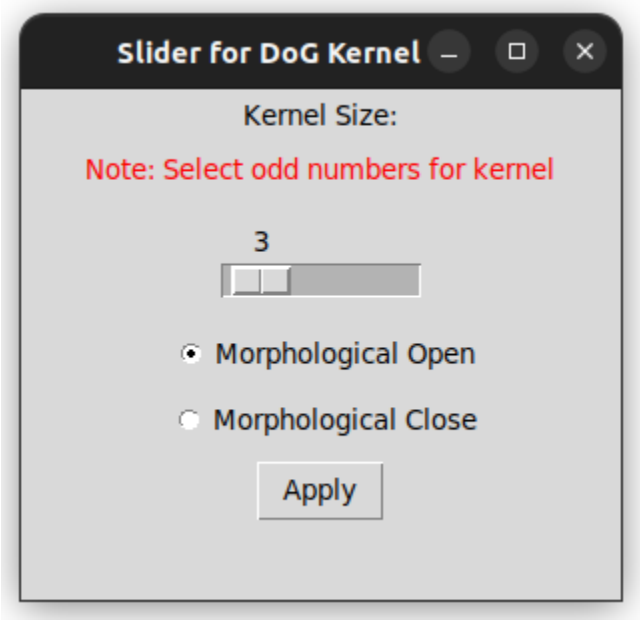
Screenshots for PixCraft

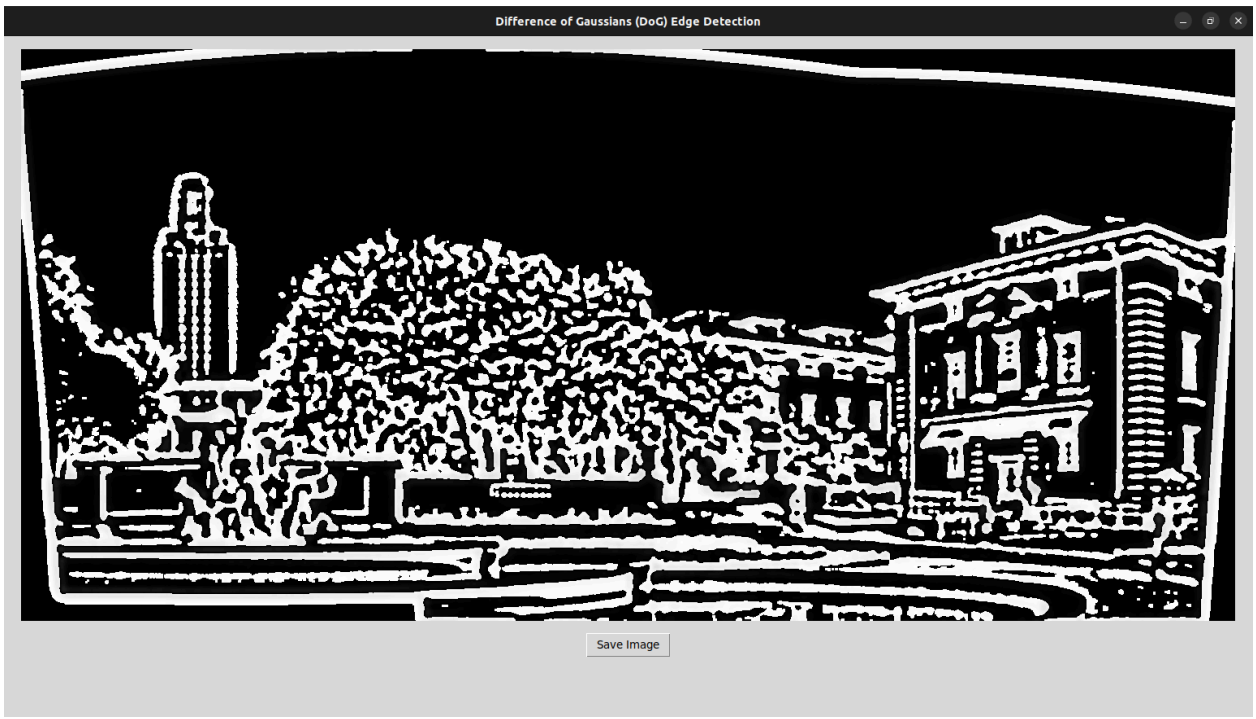

Select Images
Directory:
/media/meqdad/Meqdad/Coding/GSG Workshops/Computer Vision Bootcamp/Final Project
backend
gui
1.jpeg
2.jpeg
canny.jpg
hsv.jpg
human.jpg
stitch_left.jpg
stitch_right.jpg
File names:
1.jpeg 2.jpeg
Open
Files of type:
Image files (*.jpg,*.jpeg,*.png,*.gif)
Cancel

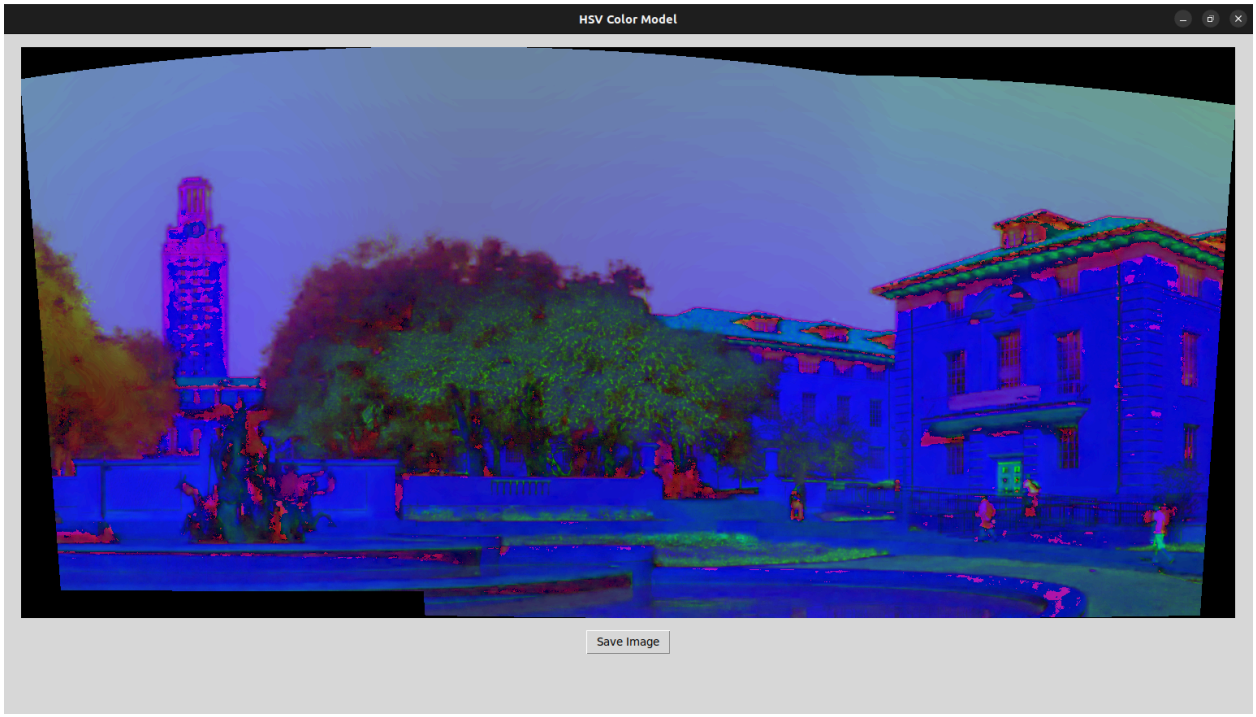

PixCraft | Meqdad Darwish
PixCraft
Add Images
Stitch Images
Right-click on an image to delete it.
© 2024 Meqdad Darwish


Stitched Image | PixCraft
Available Operations
Canny Edge Detection
DoG Edge Detection
HSV
Human Detection
Save Stitched Image

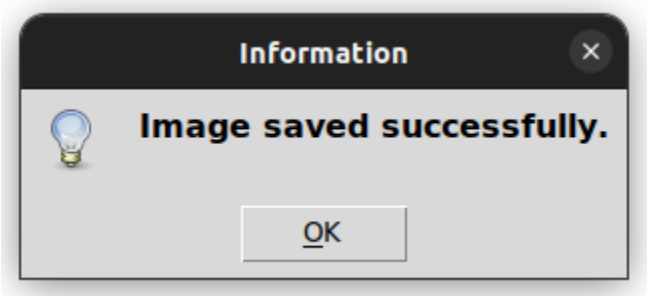

Canny Edge Detection
Save Image

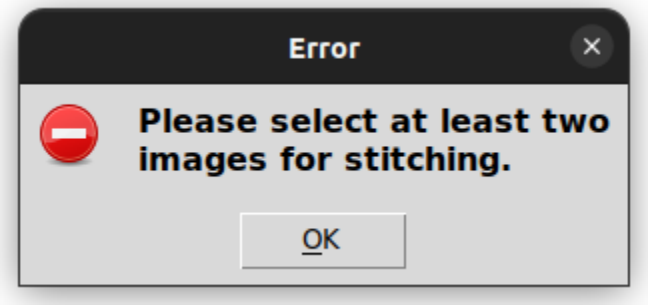

Slider for DoG Kernel
Kernel Size:
Note: Select odd numbers for kernel
3
Morphological Open
Morphological Close
Apply

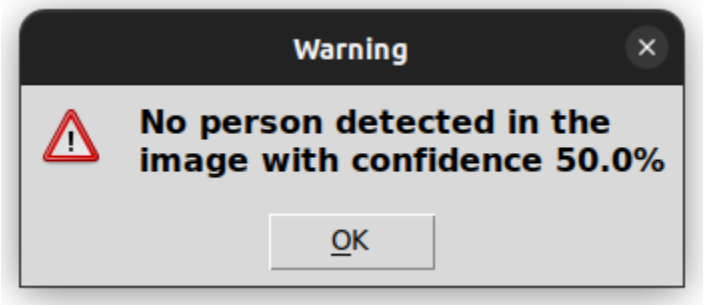

Difference of Gaussians (DoG) Edge Detection
Save Image


HSV Color Model
Save Image


Human Detection using YOLO
person 0.55
person
Save Image


Information
Image saved successfully.
OK


Error
Please select at least two images for stitching.
OK


Warning
No person detected in the image with confidence 50.0%
OK