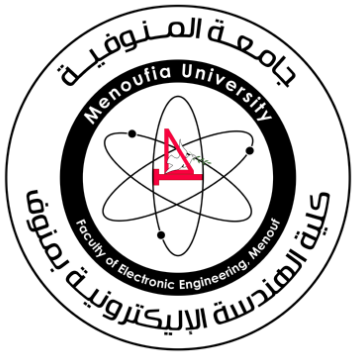


Image Processing Application Documentation

Created By:

Mahmoud Mohamed Abdelhafez Mohamed

Sec: 5

ID: 2000606

Digital Image Processing

Dr. Mohamed Berbar

❖ Introduction:

Digital image processing plays a pivotal role in numerous fields, ranging from medicine and engineering to entertainment and security. With the advancement of technology, the ability to manipulate, analyze, and extract valuable information from digital images has become increasingly important. In response to this growing demand, the Image Processing Application was developed as a versatile tool to facilitate image processing tasks.

The Image Processing Application is a Python-based graphical user interface (GUI) tool that integrates powerful libraries such as Tkinter, OpenCV, NumPy, and PIL. This application empowers users to perform a wide array of image processing operations with ease, offering a user-friendly interface for loading images, applying filters, detecting edges, segmenting objects, and extracting features.

❖ Classes and Methods:

Class: `ImageProcessingApp`

- **Description:** Represents the main application class responsible for initializing the GUI and handling image processing operations.

Method: `__init__(self, master)`

- **Description:** Initializes the ImageProcessingApp class and sets up the GUI.
- **Parameters:**
  - `master`: The master Tkinter window.

Method: `load_default_image(self)`

- **Description:** Loads a default image when the application starts.

Method: `load_image(self)`

- **Description:** Loads an image from the specified file path.

Method: `update_image(self, image)`

- **Description:** Updates the displayed image in the GUI.
- **Parameters:**
  - `image`: The image to be displayed.

Method: `add_buttons_and_sliders(self)`

- **Description:** Adds buttons and sliders for various image processing operations.

Method: `add_button(self, text, command,row)`

- **Description:** Adds a button with the specified text and command function.
- **Parameters:**
  - `text`: The text displayed on the button.
  - `command`: The function to be executed when the button is clicked.
  - `row`: The row position of the button in the GUI layout.

Method: `add_slider(self,from_, to_, default, command)`

- **Description:** Adds a slider with the specified range, default value, and command function.
- **Parameters:**
  - `from_`: The minimum value of the slider range.
  - `to_`: The maximum value of the slider range.
  - `default`: The default value of the slider.
  - `command`: The function to be executed when the slider value changes.

Method: `Zero_Slider(self)`

- **Description:** Resets the slider row and adds sliders for various image processing operations

➢ Page1_Methods

Method: `apply_lpf(self)`

- **Description:** A low pass filter is a type of filter that allows signals with a frequency lower than a certain cutoff frequency to pass through while attenuating signals with higher frequencies. In image processing, LPF is commonly used to smooth or blur images by reducing high-frequency components such as noise or sharp edges.

Method: `update_lpf(self, event)`

- **Description:** Updates the low-pass filter based on the slider value.

Method: `apply_hpf(self)`

- **Description:** A high pass filter is a type of filter that allows signals with a frequency higher than a certain cutoff frequency to pass through while attenuating signals with lower frequencies. In image processing, HPF is used to enhance or emphasize high-frequency components such as edges or fine details while suppressing low-frequency components.

Method: `update_hpf(self, event)`

- **Description:** Updates the high-pass filter based on the slider value.

Method: `apply_mean_filter(self)`

- **Description:** A mean filter, also known as an averaging filter, is a spatial domain filter that replaces each pixel in an image with the average value of its neighboring pixels. The size of the neighborhood (kernel) determines the extent of smoothing applied to the image. Mean filtering is effective in reducing noise and producing a smoother image.

Method: `update_mean_filter(self, event)`

- **Description:** Updates the mean filter based on the slider value.

Method: `apply_median_filter(self)`

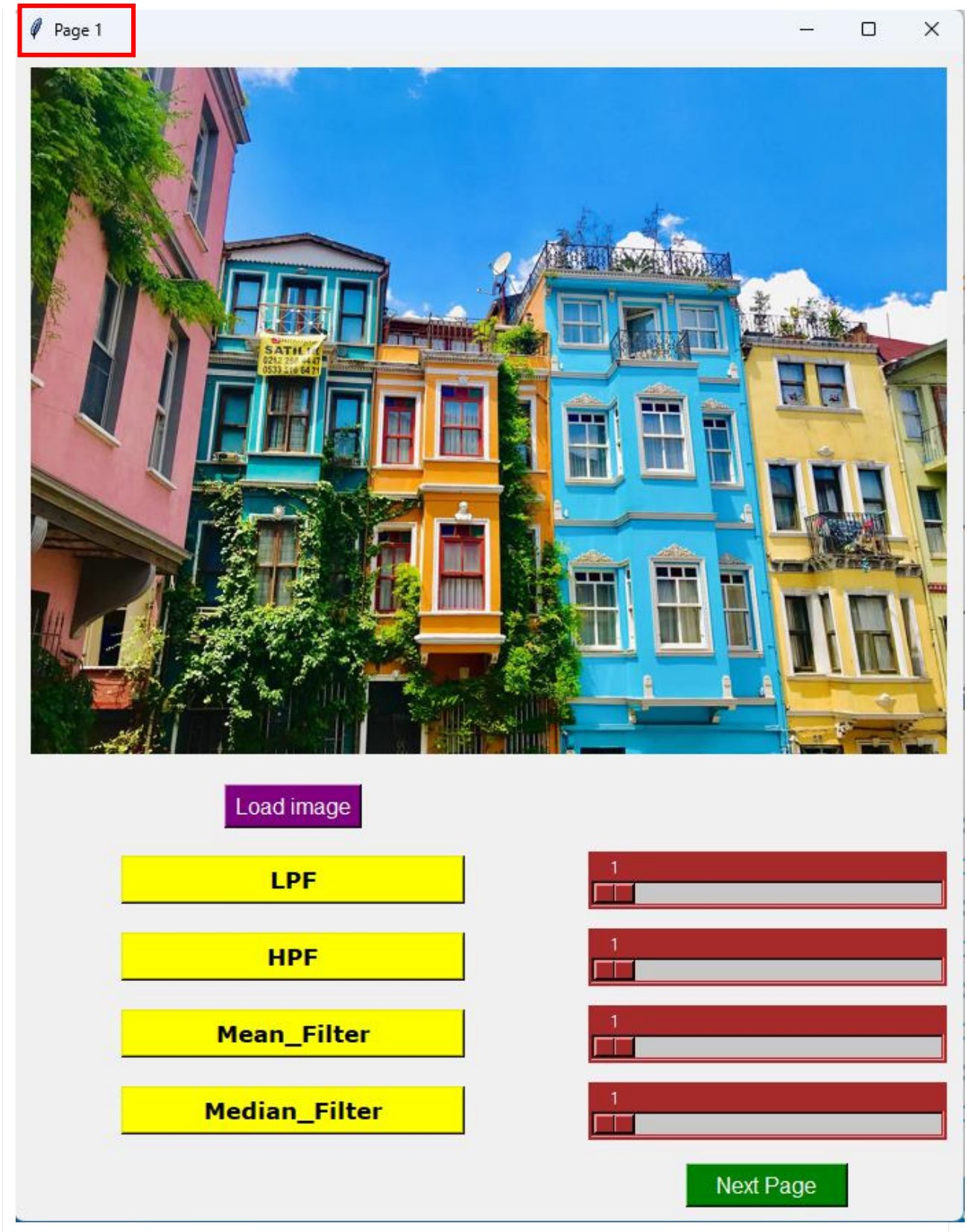
- **Description:** A median filter is a nonlinear spatial domain filter that replaces each pixel in an image with the median value of its neighboring pixels. Unlike mean filtering, which computes the average, median filtering selects the middle value in the neighborhood, making it robust to outliers and impulse noise.

Method: `update_median_filter(self, event)`

- **Description:** Updates the median filter based on the slider value.

Method: `open_page2(self)`

- **Description:** Opens another page in the application.


Page 1
Load image
LPF
HPF
Mean_Filter
Median_Filter
1
1
1
1
Next Page

➢ Page2_Methods

Method: `apply_roberts_edge_detector(self):`

- **Description:** The Roberts edge detector is one of the earliest edge detection algorithms based on computing the gradient magnitude using 2x2 convolution kernels. The Roberts kernels emphasize diagonal changes in intensity, making them sensitive to edges oriented at 45-degree angles. It detects edges by calculating the difference between pixel intensities in adjacent pixel pairs.

Method: `apply_prewitt_edge_detector(self):`

- **Description:** The Prewitt edge detector is another gradient-based edge detection algorithm similar to the Sobel operator. It computes the gradient magnitude of an image by convolving it with Prewitt kernels in the horizontal and vertical directions. Like Sobel, it emphasizes areas of significant intensity changes in the image, highlighting edges or boundaries.

Method: `apply_sobel_edge_detector(self):`

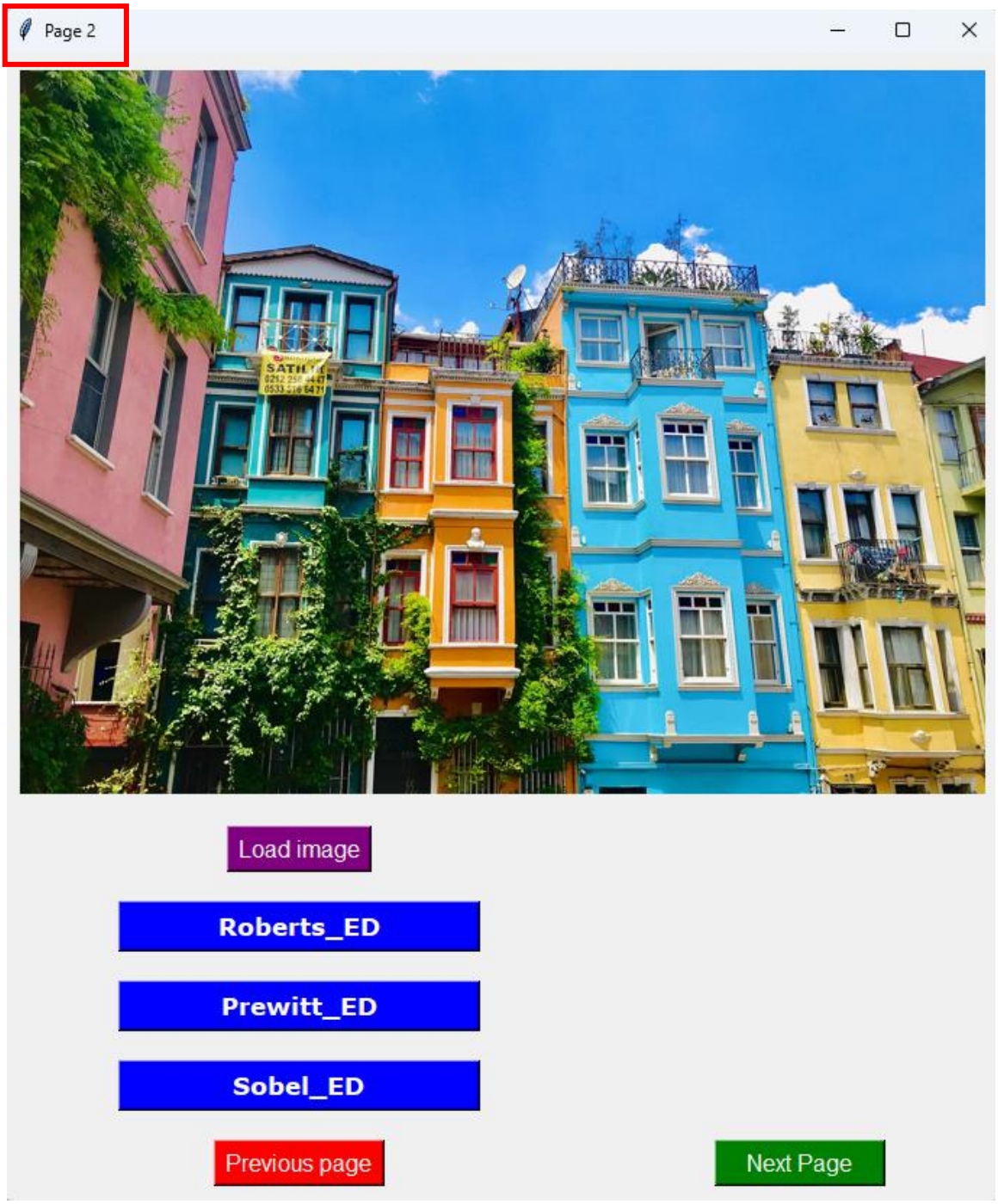
- **Description:** The Sobel edge detector is a gradient-based edge detection algorithm that computes the gradient magnitude of an image by convolving it with Sobel kernels in the horizontal and vertical directions. It highlights areas of significant intensity changes in the image, which typically correspond to edges or boundaries between objects.

Method: `open_page3(self):`

- **Description:** Opens page 3 of the application.

Method: `open_page1(self):`

- **Description:** Opens page 1 of the application



➢ Page3_Methods

Method: `apply_erosion(self):`

- **Description:** Erosion is a morphological operation that removes pixels from the boundaries of objects in an image. It works by sliding a structuring element (also known as a kernel) over the image and replacing the center pixel with the minimum pixel value within the kernel's neighborhood. Erosion tends to shrink or erode the boundaries of objects, making them thinner or disconnected.

Method: `update_erosion(self, event):`

- **Description:** Updates the erosion operation based on the selected kernel size.

Method: `apply_dilation(self):`

- **Description:** Dilation is a morphological operation that adds pixels to the boundaries of objects in an image. It works by sliding a structuring element over the image and replacing the center pixel with the maximum pixel value within the kernel's neighborhood. Dilation tends to expand or dilate the boundaries of objects, making them thicker or more connected.

Method: `update_dilation(self, event):`

- **Description**: Updates the dilation operation based on the selected kernel size.

Method: `apply_open(self):`

- **Description:** Opening is a morphological operation that combines erosion followed by dilation. It helps remove small objects and smooth out the boundaries of larger objects while preserving the overall shape and structure. Opening is performed by applying erosion first to remove small features or noise and then applying dilation to restore the object's original size.

Method: `update_open(self, event):`

- **Description:** Updates the opening operation based on the selected kernel size.

Method: `apply_close(self):`

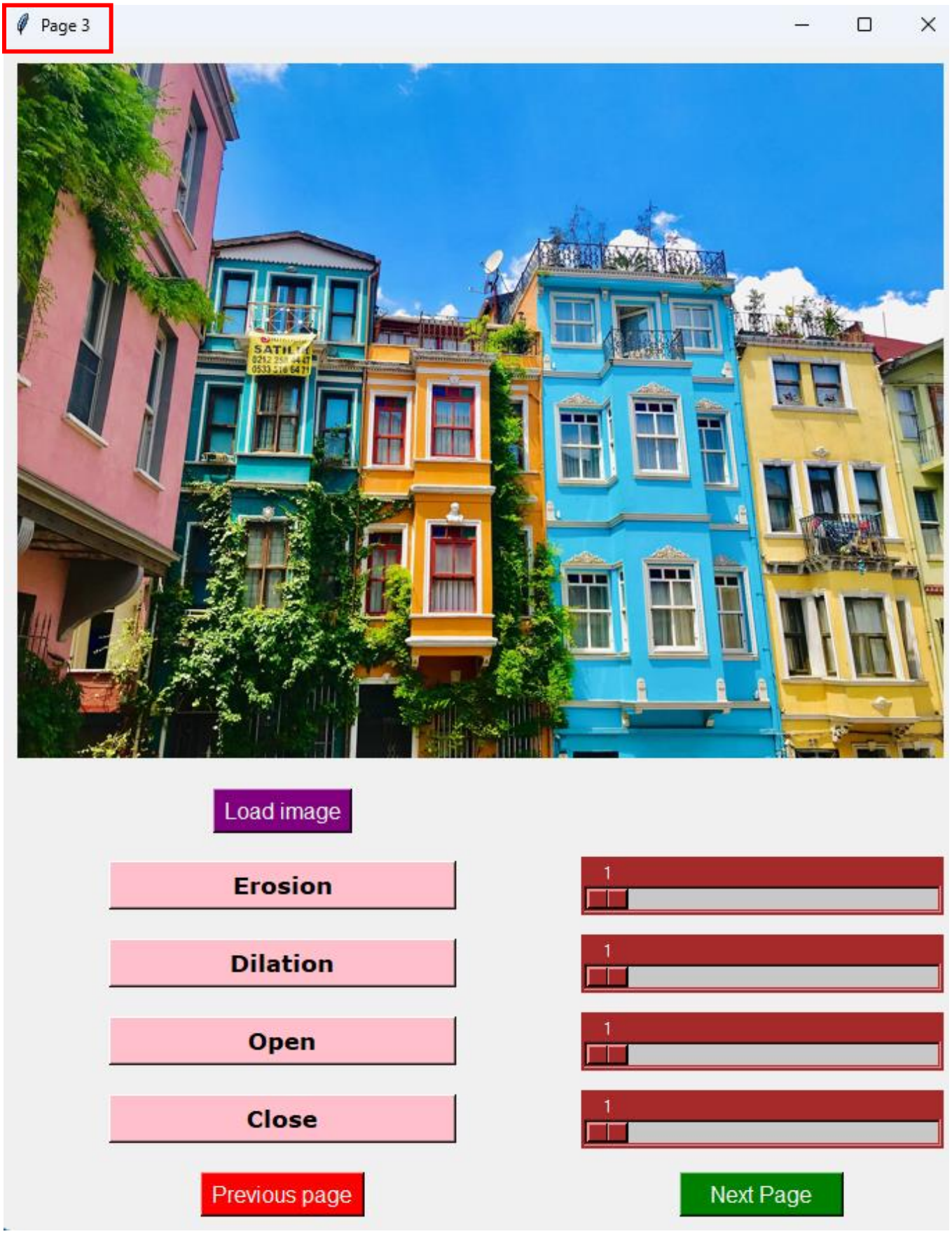
- **Description:** Closing is a morphological operation that combines dilation followed by erosion. It helps fill in small gaps, connect broken edges, and smooth out the boundaries of objects while preserving their overall shape and structure. Closing is performed by applying dilation first to close gaps or holes within objects and then applying erosion to restore the object's original size.

Method: `update_close(self, event):`

- **Description:** Updates the closing operation based on the selected kernel size.

Method: `open_page4(self):`

- **Description:** Opens page 4 of the application.


Page 3
Load image
Erosion
Dilation
Open
Close
1
1
1
1
Previous page
Next Page

➢ Page4_Methods

Method: `apply_thresholding_segmentation(self):`

- **Description:** Thresholding segmentation is a technique used to separate objects or regions of interest from the background in an image based on pixel intensity values. It involves setting a threshold value, above or below which pixels are classified as belonging to the foreground (object) or background. Thresholding segmentation is a simple yet effective method for image segmentation, especially in cases where objects have distinct intensity differences from the background.

Method: `update_thresholding_segmentation(self, event):`

- **Description:** Updates the thresholding segmentation operation based on the selected threshold value.

Method: `apply_region_split_merge_segmentation(self):`

- **Description**: Region split and merge segmentation is a hierarchical segmentation technique that recursively partitions an image into regions based on pixel similarity criteria and then merges adjacent regions to form larger, homogeneous regions. It begins by dividing the image into smaller regions (splitting) and then iteratively merges adjacent regions that meet certain similarity conditions (merging). This process continues until no further merging is possible, resulting in a segmentation hierarchy.

Method: `apply_hough_circle_transform(self):`

- **Description:** The Hough circle transform is a feature detection technique used to identify circular shapes or patterns within an image. It is an extension of the Hough transform, which is commonly used for detecting straight lines. The Hough circle transform works by converting image space coordinates to a parameter space representation, where circles are represented by their center coordinates and radii. By accumulating votes for possible circle parameters, the algorithm can robustly detect circles of varying sizes and positions in the image.

Method: `open_page3(self):`

- **Description:** Opens page 3 of the application.