Assignment 3 -Documentation

Group - K

Hardik Soni (20CS30023)

Khushboo Singhania (19CS30023)

- In this assignment, we have implemented some basic knowledge related to projective space and removing projective distortion.
- The GUI, binding user inputs, button commands, and helper functions are all defined in the **ImageApp** Class in the **main.py** file.
- The details of them are below.
- **Note-** In our implementation of the following functions/operations, we assume that the **i-axis(rows)** of the image to be **X-axis** (**vertically down**) and **j-axis(columns)** to be **Y-axis** (**horizontally right**). That is, a pixel located at the ith row and jth column, will have (x, y) coordinate = (i,j).

In **main.py** file:-

1. **def on_resize(self, event):** This function *"on_resize"* is an event handler that is called when the window containing the button bar is resized. Its purpose is to dynamically adjust the padding of each button in the button bar based on the new width of the window, this function ensures that the buttons in the button bar are evenly spaced and visually appealing even when the window is resized.
2. **def check(self,x,y):** This function checks whether the clicked coordinate lies within or atleast on the edges of the canvas of the input image.
3. **def push_oval(self, x,y):** This button pushes the new coordinate (x,y) into the list of points on the canvas.
4. **def calculate_length(self, p1, p2):** Calculate the length between two points using the Euclidean distance formula.

5. **def refresh(self):** This is the event handler function for the *"Refresh"* button, it erases all the points and lines drawn on the canvas, users may use this button in case the image has too many points / lines.
6. **def exit(self):** To exit the GUI.
7. **def open_image(self):** This `open_image` method is part of the **ImageApp** class and is used to open and display an image file (JPEG, JPG, or PNG) on a Tkinter canvas. Here's a breakdown of the function:
    a. Check if an image is already open in the canvas (`self.open_jpg`). If it is, display an error message and return without opening a new image.
    b. Use `filedialog.askopenfilename` to prompt the user to select an image file. Only files with the extensions `.jpeg`, `.jpg`, or `.png` are selectable.
    c. If a file is selected (`file_path` is not empty), proceed to open the image:
        i. Store the path of the selected image (`file_path`) in `self.path`.
        ii. Use OpenCV (`cv2`) to read the image from the file (`cv2.imread`) and convert its color format from BGR to RGB (`cv2.cvtColor(image, cv2.COLOR_BGR2RGB)`).
        iii. Convert the image from a NumPy array to a PIL Image (`Image.fromarray(image)`).
        iv. Convert the PIL Image to a Tkinter PhotoImage (`ImageTk.PhotoImage(image)`).
        v. Use `self.canvas.create_image` to display the image on the canvas at the specified coordinates (`self.x_ori`, `self.y_ori`) with the specified anchor (`tk.NW`, top-left corner).

    This method allows the user to select and display an image file on the Tkinter canvas, handling errors if an image is already open or if the selected file is not a valid image.
8. **def calculate_pixel_coordinate(self):** This `calculate_pixel_coordinate` method is part of the **ImageApp** class and is used to calculate and display the pixel coordinates of the last drawn point on the Tkinter canvas. Here's a breakdown of the function:

a. Check if `self.path` is `None`, which indicates that no image has been opened yet. If so, display an error message and return without calculating the pixel coordinates.
b. Check if there are fewer than 1 oval (point) drawn on the canvas (`len(self.ovals) < 1`). If there are no points, display an error message and return without calculating the pixel coordinates.
c. If there is at least one point present on the canvas, calculate the pixel coordinates of the last drawn point:
    i. Subtract `self.x_ori` from `self.last_x_coord` and `self.y_ori` from `self.last_y_coord` to get the coordinates relative to the image origin.
    ii. Display the pixel coordinates in a message box using `messagebox.showinfo`.

This method allows the user to calculate and display the pixel coordinates of the last drawn point on the canvas, provided that an image has been opened and at least one point has been drawn.

9. **def calculate_line_length(self):** This `calculate_line_length` method is part of the **ImageApp** class and is used to calculate and display the length of the last drawn line on the Tkinter canvas. Here's a breakdown of the function:
    a. Check if `self.path` is `None`, which indicates that no image has been opened yet. If so, display an error message and return without calculating the line length.
    b. Check if there are fewer than 1 line drawn on the canvas (`len(self.lines) < 1`). If there are no lines, display an error message and return without calculating the line length.
    c. If there is at least one line present on the canvas, calculate the length of the last drawn line:
        i. Retrieve the last drawn line from `self.lines`.
        ii. Calculate the length of the line using a suitable method (not shown in the provided code).
        iii. Display the length of the line in a message box using `messagebox.showinfo`.

This method allows the user to calculate and display the length of the last drawn line on the canvas, provided that an image has been opened and at least one line has been drawn. The last drawn line is stored in `self.lines`.

10. **def calculate_dual_conic(self):** This `calculate_dual_conic` method is part of the **ImageApp** class and is used to calculate and display the dual conic at infinity for a set of lines in a projective plane. Here's a breakdown of the function:
    a. Check if `self.path` is `None`, which indicates that no image has been opened yet. If so, display an error message and return without calculating the dual conic.
    b. It first defines arrays *points_h* and *points_v* containing points representing horizontal and vertical lines, respectively.
    c. It then computes the lines corresponding to these points and combines them into a single array `lines`.
    d. The code then prints the pairs of lines along with their endpoints.
    e. Next, it computes the dual conic at infinity by summing the outer products of all lines.
    f. Display the dual conic at infinity in the console and in a message box using `messagebox.showinfo`.

    This method demonstrates the calculation and visualization of the dual conic at infinity for a set of lines in a projective plane.

11. **def calculate_homo_matrices(self):** This `calculate_homo_matrices` method is part of the **ImageApp** class and is used to calculate homography matrices for transforming a painting with known corners to fit into target rectangles with different aspect ratios (2:3 and 3:4) and display the transformed images. Here's a breakdown of the function:
    a. Check if `self.path` is `None`, which indicates that no image has been opened yet. If so, display an error message and return without calculating the homography matrices.
    b. Load the image using OpenCV's `cv2.imread` method.

c. Define the corners of the target rectangles for aspect ratios 2:3 and 3:4.
d. Define the corners of the painting (you need to identify these points from the image).
e. Calculate the homography matrices using the corners of the painting and the target rectangles using `cv2.findHomography`.
f. Display the homography matrices in message boxes.
g. Create a Toplevel window to display the transformed images for each aspect ratio.
h. Transform the image using the calculated homography matrices (cv2.warpPerspective).
i. Save the transformed images to files.
j. Convert the transformed images to RGB format and create Tkinter PhotoImage objects from them.
k. Display the transformed images in the Toplevel windows using Label widgets.

This method demonstrates the calculation of homography matrices for perspective transformation and the visualization of the transformed images to fit into different aspect ratios.

12. **def perform_affine_rectification(self):** This `perform_affine_rectification` method is part of a class and is used to perform affine rectification on an image using specified points. Here's a breakdown of the function:
    a. Check if **self.path** is **None**, which indicates that no image has been opened yet. If so, display an error message and return without performing the affine rectification.
    b. Load the image using OpenCV's **cv2.imread** method.
    c. Define sets of points (*pt1* to *pt8*) that correspond to the endpoints of lines with respect to whom we are gonna perform affine rectification.
    d. Calculate the transform matrix **H** using *l1, l2, l3,* and *l4*, calculated using *pt1* to *pt8*.
    e. Create an output folder if it doesn't exist.
    f. Create a Toplevel window (**affine_rec**) to display the rectified image.

g. Apply the affine transformation using **cv2.warpPerspective**.
h. Save the rectified image to a file.
i. Convert the rectified image to RGB format and create a Tkinter PhotoImage object.
j. Display the rectified image in the Toplevel window using a Label widget.

This method demonstrates how to perform affine rectification on an image using specified points and visualize the rectified image.

13. **def perform_metric_rectification(self):** This `perform_metric_rectification` method is part of the **ImageApp** class and is used to perform metric rectification on an image using specified points. Here's a breakdown of the function:
    a. Check if `self.path` is `None`, which indicates that no image has been opened yet. If so, display an error message and return without performing the metric rectification.
    b. Load the image using OpenCV's `cv2.imread` method.
    c. Define a target rectangle in the metric space (`rect`).
    d. Define the corresponding points in the input image (`points`).
    e. Calculate the homography matrix (`homography_matrix`) using `cv2.findHomography`.
    f. Compute the true aspect ratio from the transformed image using the homography matrix.
    g. Display the true aspect ratio in a message box.
    h. Create an output folder if it doesn't exist.
    i. Create a Toplevel window (`met_rec`) to display the rectified image.
    j. Perform the metric rectification using `cv2.warpPerspective`.
    k. Save the rectified image to a file.
    l. Convert the rectified image to RGB format and create a Tkinter PhotoImage object.
    m. Display the rectified image in the Toplevel window using a Label widget.

This method demonstrates how to perform metric rectification on an image using specified points and visualize the rectified image.

14. **def on_canvas_click(self, event)**: This on_canvas_click method is part of a class and handles mouse click events on a canvas. Here's a breakdown of the function:

    a. Convert the event coordinates (event.x and event.y) to canvas coordinates (x and y) using self.canvas.canvasx and self.canvas.canvasy.
    b. Check if the drawing mode is set to "line" or "point" using self.drawing_mode.get().
    c. If the drawing mode is "line":
    d. Check if the clicked coordinates are valid using the check method (not provided in the code snippet).
    e. If the coordinates are valid:
    f. Print the clicked pixel coordinates relative to a reference point (140.0, 130.0).
    g. If self.li is 0 (indicating the start of a new line), set the last clicked coordinates as the starting point (self.last_x_coord and self.last_y_coord), push an oval marker to the canvas, and update self.li to 1.
    h. If self.li is 1 (indicating the end of a line), push another oval marker, create a line on the canvas between the last and current clicked coordinates, calculate the length of the line using calculate_length method (not provided in the code snippet), display the length in a message box, update the last clicked coordinates, reset self.li to 0, and store the line object in self.lines.
    i. If the coordinates are invalid, display an error message.
    j. If the drawing mode is "point":
    k. Check if the clicked coordinates are valid.

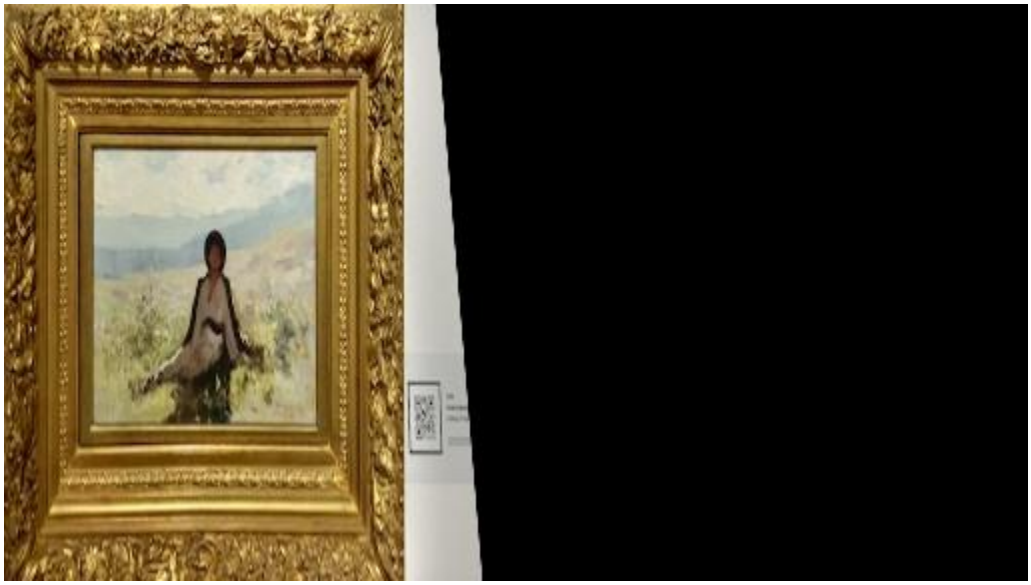
l. If valid, print the clicked pixel coordinates relative to a reference point (140.0, 130.0), push an oval marker to the canvas, and display the pixel coordinates of the drawn point in a message box.

m. If invalid, display an error message.

n. If the drawing mode is neither "line" nor "point", display an error message.

Results:-

Transformed Image under 2:3 Aspect Ratio Target Rectangle



Transformed Image under 3:4 Aspect Ratio Target Rectangle

Affine Rectification

Metric Rectification