Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

FINAL PROJECT

INTRODUCTION TO DIGITAL IMAGE PROCESSING

Instructor: **Mr. TRINH HUNG CUONG**

Student: **LE MINH ANH – 521H0004**

NGUYEN PHUC THINH – 521H0162

Class **: 21H50201**

Year **: 25**

HO CHI MINH CITY, 2024

Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

FINAL PROJECT

INTRODUCTION TO DIGITAL IMAGE PROCESSING

Instructor: **Mr. TRINH HUNG CUONG**

Student: **LE MINH ANH – 521H0004**

NGUYEN PHUC THINH – 521H0162

Class **: 21H50201**

Year **: 25**

HO CHI MINH CITY, 2024

ACKNOWLEDGEMENT

We would like to thank Ton Duc Thang University, Faculty of Information Technology for including this subject in the training chart of Software Engineering, Computer Science and for creating the best learning conditions for we as well as all students of the faculty. Our report is a product of what we have learned in the semester.

We would like to thank Mr. Trinh Hung Cuong for his patience, encouragement, and inspirational teaching style, which has inspired us to pursue excellence in the field of Enterprise Resource Planning.

Thank you, Mr. Trinh Hung Cuong, for your exceptional mentorship and dedication to our academic development.

Finally, we wish you good health and success in your noble career.

THIS PROJECT WAS COMPLETED AT
TON DUC THANG UNIVERSIY

We fully declare that this is our own project and is guided by Mrs. Huynh Ngoc Tu; The research contents and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

Besides that, the project also uses a number of comments, assessments as well as data from other authors, other agencies and organizations, with citations and source annotations.

Should any frauds were found, we will take full responsibility for the content of our report. Ton Duc Thang University is not related to copyright and copyright violations caused by us during the implementation process (if any).

Ho Chi Minh city, 4th Jun, 2024

Author

(Sign and write full name)

Le Minh Anh

Nguyen Phuc Thinh

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh January, 2024

(Sign and write full name)

Evaluation section for grading instructor

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh January 2024

(Sign and write full name)

SUMMARY

This article explains how to process digital images, with the topic being "Retrieving the time from the photo of a clock". Through the process of researching and completing this project, we realized that using the OpenCV library will make digital image processing easier, in addition, there is also a need for a combination of the NumPy library and several modules such as os, math to be able to consolidate the work more completely.

TABLE OF CONTENT

TABLE OF FIGURE

CHAPTER 1: SOLVING METHOD

1.1 The topic

Topic: Retrieving the time from the photo of a clock

Requirement:

- Input: At least 10 photos containing analog wall clocks with different content (different clock types, time difference of 1-2 hours between photos, clock hands do not coincide together).

Figure 1. 1 Clock 1

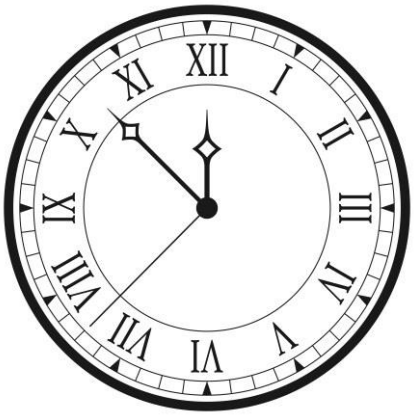


Figure 1. 2 Clock 2

Figure 1. 3 Clock 3

Figure 1. 4 Clock 4

Figure 1. 5 Clock 5

Figure 1. 6 Clock 6

Figure 1. 7 Clock 7

Figure 1. 8 Clock 8

Figure 1. 9 Clock 9

Figure 1. 10 Clock 10

- Output:
    - In each photo, draw rectangular frames surrounding the hour hand, minute hand, and second hand with difference colors. Drawing rectangular frames must be done automatically, not manually.
    - In each photo, output the time as hours:minutes:seconds.

1.2 The solving method

To solve the given problem, we will use digital image processing functions in the OpenCV library such as cv2.HoughLinesP, cv2.findCountours, and mathematical methods to calculate the necessary data for Determining time from clock images.

The detailed processes

- Step 1: Import necessary libraries

Import the necessary libraries and modules: OpenCV, NumPy, os, and Math using the import command. In there:

- os is a Python module that provides functions to interact with the operating system, allowing users to perform operations: create/delete folders, open/read/write files, etc.

- OpenCV is a library that allows image and video processing in Python

- Math is a Python module that provides basic mathematical functions

- NumPy is a Python library that provides functions for manipulating arrays and matrices.

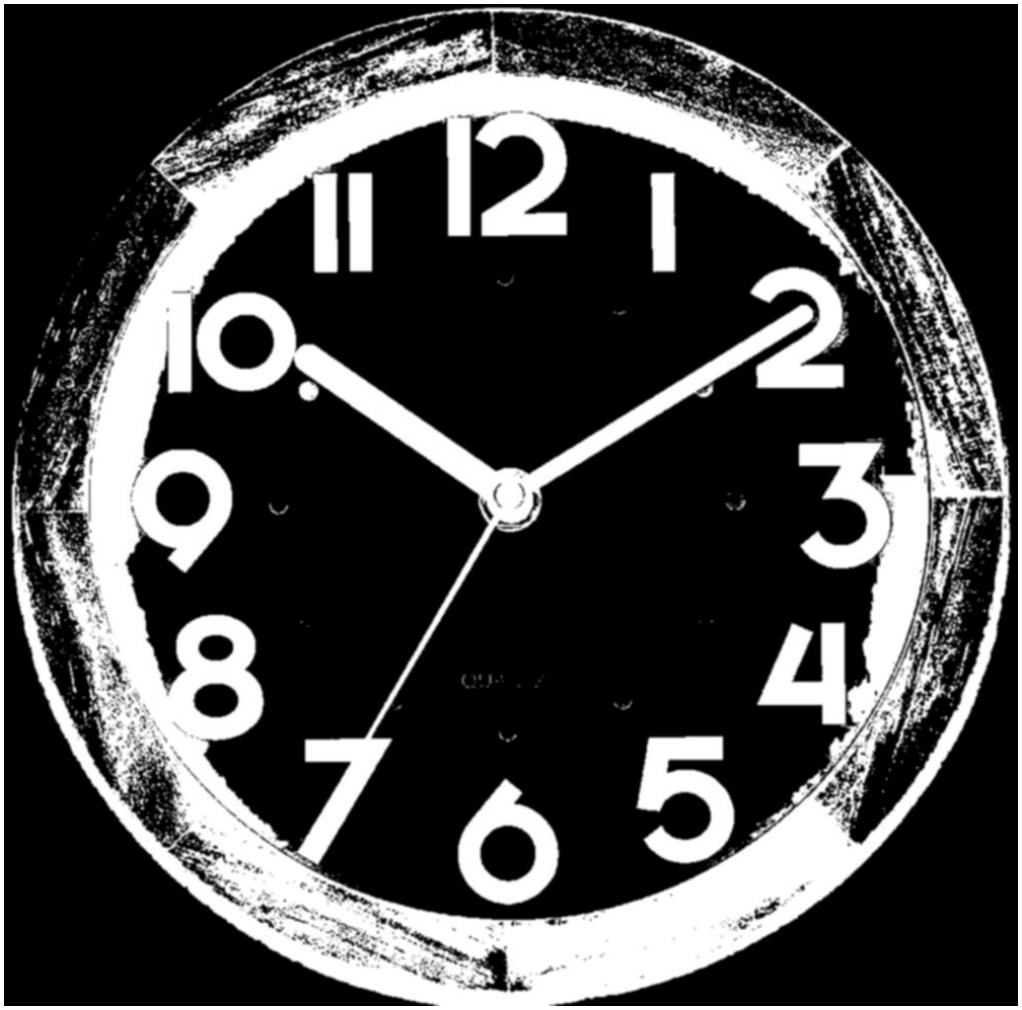
```
pip install opencv-python
pip install numpy
pip install os-win
pip install math
```

- Step 2: Image Preprocessing

First, get the size of the image by taking the height and width. After having the width and height data, calculate the scaling factor to resize the image. Then, perform the following steps in turn to process image quality:

- Convert image from BGR color space to HSV by using function ***cv2.cvtColor***.
- Invert color value in HSV space by using function ***cv2.bitwise_not***.

- Balance the brightness of the image using the CLAHE method by using function ***cv2.createCLAHE***.
- Use the CLAHE method to adjust the brightness of the V channel in the HSV image.
- Use thresholding to segment the image by using function ***cv2.threshold***.
- Use Gaussian blur to smooth the image by using function ***cv2.GaussianBlur.***
- After these step, the image will look like this:



- Step 3: Clock Detection

  In this step, we detect the clock by circle size or rectangle size. se the Hough method to find circles in the image with the ***cv2.HoughCircles*** function, then find and get the coordinates and radius of the largest circle. Here, the largest circle is the clock.

  In case a circle cannot be found, a rectangle will be found. We will find objects in the image using the ***cv2.findContours*** function. Similar to the circle, we will also find and get the data of the largest rectangle.

  After this step, it will show the the x, y and the radius of the clock.

```
x:  498 y:  483 radius:  419
```

- Step 4: Detect Line Segments

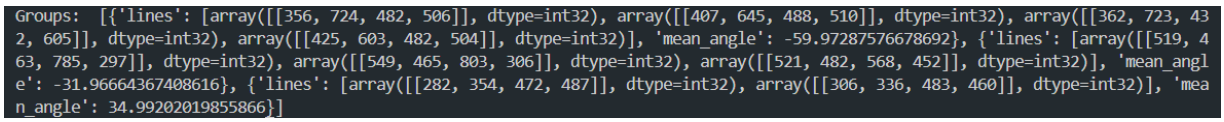
  Use the Canny filter to find edges in the image using the ***cv2.Canny*** function and use the Hough method to find lines in the boundary image using ***cv2.HoughLinesP***.

```
Lines:  [[[356 724 482 506]]

 [[519 463 785 297]]

 [[549 465 803 306]]

 ...

 [[414 940 454 972]]

 [[609 889 672 866]]

 [[521 482 568 452]]]
```

- Step 5: Group lines

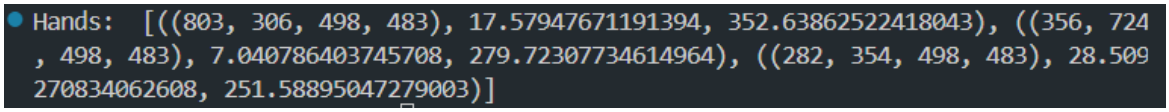
  We will group the lines close together and almost parallel to each other. Check whether lines belong to any group or not.

\



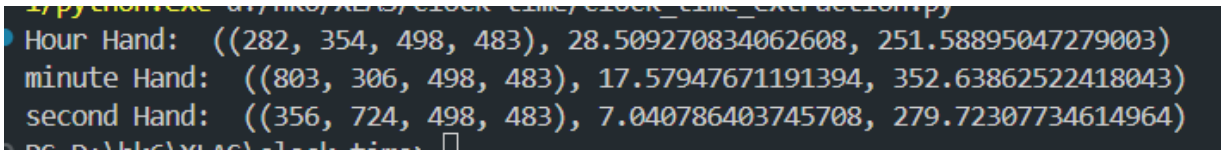
- Step 6: Dectect hand line

  This step has the purpose of finding the farthest endpoint from the clock center of a line segment among line segments. First, find the point farthest from the center of the clock. Then, create line with maximum thickness and the farthest point from the center of the clock. Finally, arrange in order of decreasing thickness, the 3 lines with the greatest thickness will be in the same order as the hour hand, minute hand, and second hand.



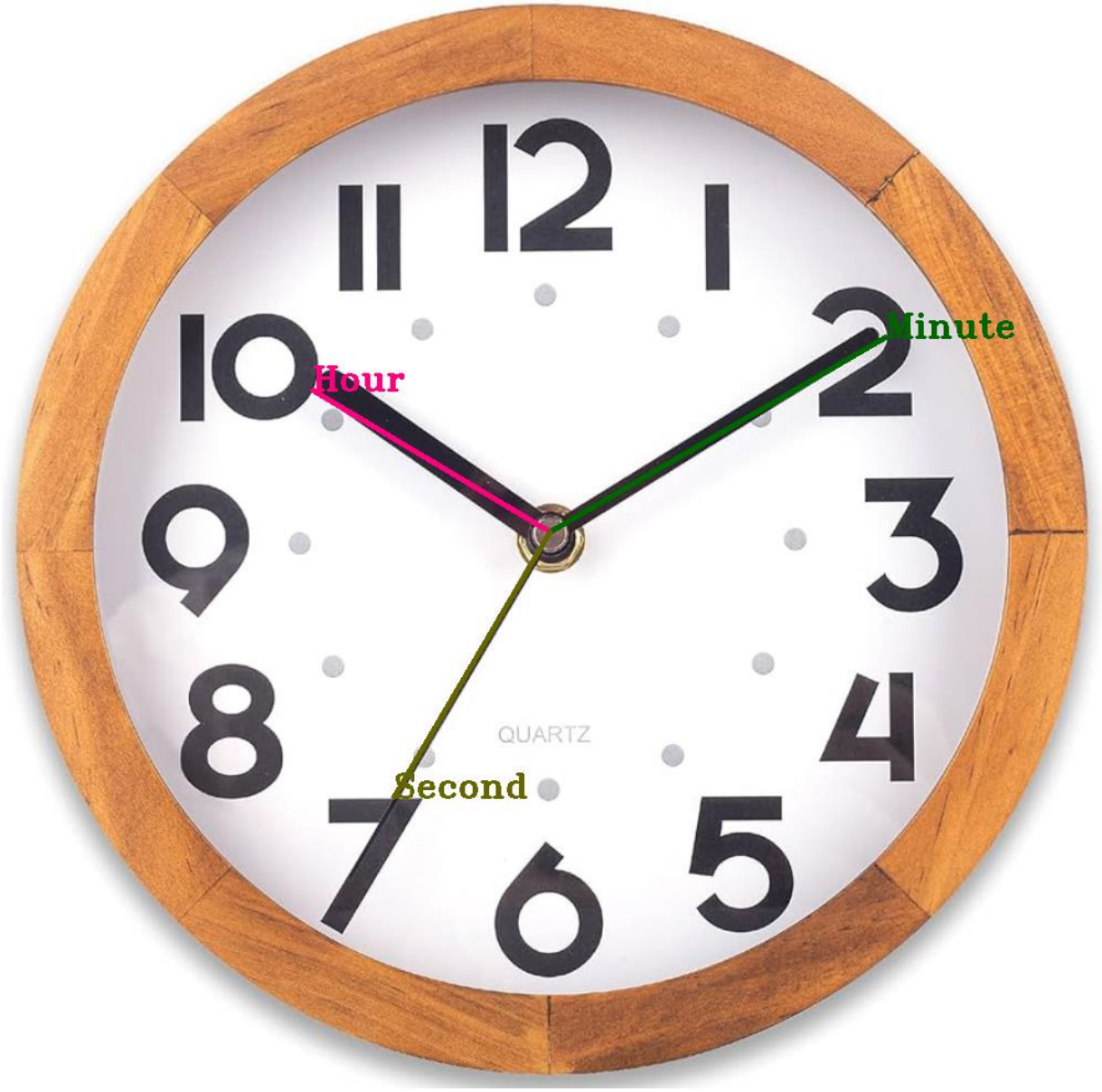
- Step 7: Determine hands

  Determine the hour hand, second hand, and minute hand based on the thickness and length of the clock hand. First, determine the second hand as the thinnest hand. After finding the second hand, the second hand will be removed from the list of clock hands. Now in the list of clock hands, there are only 2 hands left, the hour hand has the shortest length and the last is the minute hand.



- Step 8: Draw frames

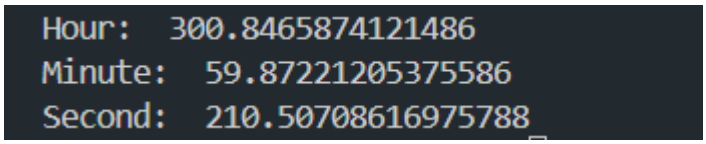
  Draws a frame around and labels the clock hands back on the image. In which ***cv2.line*** is used to draw a frame around the clock hands and ***cv2.putText*** is used to label the clock hands.



- Step 9: Calculates the angle of the clock hands

  Calculate the angle between the clock hand (set as vector u) and the horizontal reference line (set as vector v). Use the cosine formula to calculate

the cosine between these two vectors. Convert cosines to angles using math.acos function, then convert the angle from radians to degrees using math.degrees function. Finally, calculate the directional product of the two vectors. If the directional product is greater than 0, it means vector u is to the left of vector v. Conversely, if the directional product is less than 0, it means the clock hand is to the right of the reference line.

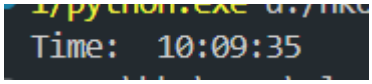


- Step 10: Calculates the time

  Hour is calculated by dividing the angle of the hour hand by 30 (each hour corresponds to 30 degrees). Minute is calculated by dividing the angle of the minute hand by 6 (each minute corresponds to 6 degrees). Second is calculated by dividing the angle of the second hand by 6 (each second corresponds to 6 degrees).

  If the angle of the hour hand is close to an integer multiplied by 30 (i.e. close to a specific hour) and the angle of the minute hand is approximately between 0 and 6 (i.e. 1 round of 60 minutes has passed), set hour to 0.
  If the angle of the hour hand is close to a specific hour and the angle of the minute hand is close to 360 (ie close to 12 o'clock), set minute to 0.

  If the angle of the minute hand is close to a specific minute and the angle of the second hand is close to 360 (ie close to 60 seconds), set second to 0.
  Convert the results to integers and format the time as HH:MM:SS.

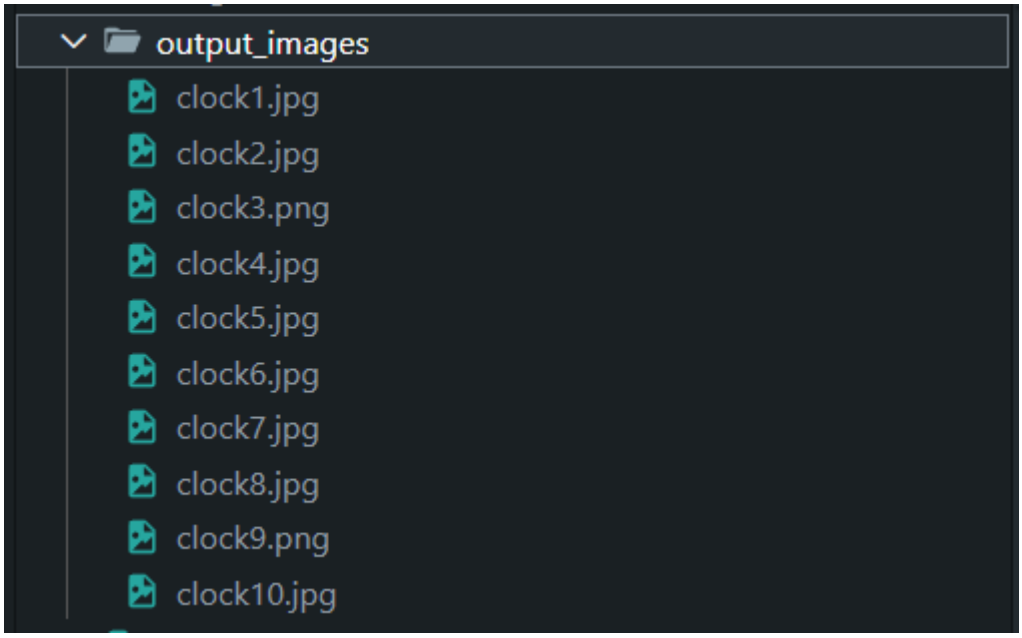


- Step 11: Display time

  Display the time on the image by using ***cv2.putText.***

- Step 12: Save the result

  First, read the image file with the ***cv2.imread*** function, then call the image processing function to perform the image processing steps. After getting the results save to the output directory using ***os.path.join*** and ***cv2.imwrite***, also displays images using the ***cv2.imshow*** function.



CHAPTER 2: EXPERIMENTAL STEPS AND RESULTS

2.1 The experimental steps

Below are the experimental steps of the previously stated process

- Step 1: Import necessary lỉbararies

```
# Import the necessary libraries
import os
import cv2
import math
import numpy as np
```

Figure 2. 1 Step 1

- Step 2: Image processing

```
#This function is used to pre-process the image
def pre_process(img):
    # Get the height and width of the image
    height, width, _ = img.shape
    # Calculate the scale factor to resize the image to a maximum of 1000 pixels
    scale = 1000 / max(height, width)
    # Resize the image using the calculated scale factor
    img = cv2.resize(img, (int(width * scale), int(height * scale)))
    return img

# First step: pre-process the image to improve the quality of the image
img = pre_process(img)
# Convert image from BGR color space to HSV
img_hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
# Invert color values in HSV space
img_hsv = cv2.bitwise_not(img_hsv)
# Balance the brightness of the image using the CLAHE method
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
# Use the CLAHE method to adjust the brightness of the V channel in the HSV image
img_hsv[:, :, 2] = clahe.apply(img_hsv[:, :, 2])
# Use thresholding to segment the image
_, thresh = cv2.threshold(img_hsv[:, :, 2], 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
# Use Gaussian blur to smooth the image
blurred = cv2.GaussianBlur(thresh, (5, 5), 0)
```

Figure 2. 2 Step 2

- Step 3: Clock detection

```python
#This function detects the clock by circle size or rectangle size.
def clock_detector(img, blurred):
    # Initialize variables to store the center and radius of the circle
    radius = 0
    cent_x, cent_y = 0, 0
    # Use the Hough method to find circles in the image
    circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, 1, 400, param1=50, param2=100, minRadius=100, maxRadius=500)
    # Initialize variable to store the largest circle
    bigest_circle = None
    # Find the largest circle in the image
    if circles is not None:
        # Browse through the circles found in the image to find the largest circle
        for circle in circles[0, :]:
            # Get the radius of the circle
            x, y, r = circle
            # Update the bigger circle if needed
            if r > radius:
                bigest_circle = circle
        # Get the coordinates and radius of the largest circle
        x, y, r = bigest_circle
        cent_x = int(x)
        cent_y = int(y)
        radius = int(r)
```

```python
    # Incase the radius is not found a circle, let find by rectangle.
    else:
        # Find objects in the image using the findContours method
        contours, _ = cv2.findContours(blurred, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        # Initialize variables to store the area and largest rectangle
        max_area = 0
        biggest_rectangle = None
        for contour in contours:
            # Calculate the area of the contour
            area = cv2.contourArea(contour)
            # Update the bigger rectangle if needed
            if area > max_area:
                max_area = area
                biggest_rectangle = contour
        if biggest_rectangle is not None:
            # Get the coordinates of the largest rectangle
            (x, y, w, h) = cv2.boundingRect(biggest_rectangle)

            # Calculate the coordinates of the center of the rectangle
            cent_x = x + w // 2
            cent_y = y + h // 2
            # Calculate the radius of the circle inscribed in the rectangle
            radius = min(w, h) // 2
    return cent_x, cent_y, radius
```

Figure 2. 3 Step 3

- Step 4: Line detection

```python
#This function detects the lines in the clock image
def line_detector(img, blurred):
    # Use Canny filter to find edges in image
    edges = cv2.Canny(blurred, 50, 150)
    # Use the Hough method to find lines in the edge image
    lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=90, minLineLength=30, maxLineGap=5)
    return lines
```

Figure 2. 4 Step 4

- Step 5: Group lines

```python
# This function groups lines that are close together and nearly parallel to each other
def group_lines(lines, center_x, center_y, radius):
    # Initialize a list to store groups of lines
    groups =[]
    # Iterate through the lines
    for line in lines:
        # Get the coordinates of the two endpoints of the line
        x1, y1, x2, y2 = line[0]

        # Find the length of the line from the center of the clock
        length1 = np.sqrt((x1 - center_x)**2 + (y1 - center_y)**2)
        length2 = np.sqrt((x2 - center_x)**2 + (y2 - center_y)**2)

        #Find the farthest and closest points from the center of the clock
        farthest_length = np.max([length1, length2])
        closest_length = np.min ([length1, length2])

        # The farthest point must be within the radius of the clock and the nearest point must only
        if ((farthest_length < radius) and (closest_length < radius*50/100)):
            # Calculate the angle of the line in degrees
            angle = math.atan2(y2 - y1, x2 - x1)
            angle = math.degrees(angle)
```

```python
            # Initialize flag variable to check whether the line belongs to any group or not
            grouped = False
            for group in groups:
                # Get the average angle of the group
                mean_angle = group['mean_angle']
                # Incase the angle of the line is close to the average angle of the group or the average angle of the group plus
                if abs(angle - mean_angle) < 12 or abs(angle - mean_angle - 180) < 12 or abs(angle - mean_angle + 180) < 12:
                    # Add lines to the group
                    group['lines'].append(line)
                    # Set the flag variable to True to signal that the group has been found
                    grouped = True
                    break
            # If you cannot find a suitable group
            if not grouped:
                # Create a new group with its lines and angles
                groups.append({'lines': [line], 'mean_angle': angle})
    return groups
```

Figure 2. 5 Step 5

- Step 6: Detect hand line

```python
# This function to calculate the distance between two parallel lines.
def distance_lines(line_1, line_2):
    # Get the coordinates
    x1_1, y1_1, x2_1, y2_1 = line_1[0]
    x1_2, y1_2, _ , _ = line_2[0]
    # Create vector line
    vector = np.array([x2_1 - x1_1, y2_1 - y1_1])
    # Create a vector connecting a point on the other line
    vector_connect = np.array([x1_2 - x1_1, y1_2 - y1_1])
    # Distance between the two lines.
    distance_line = np.abs(np.cross(vector, vector_connect)) / np.linalg.norm(vector)
    return distance_line
```

```python
# This function has the purpose of finding the farthest endpoint from the clock center of a line segment among line segments
def hands_detector(groups, center_x, center_y):
    # Initialize a list clock hands
    hands_list = []
    # Interate through the groups
    for group in groups:
        # Get the list of lines in the group
        lines = group['lines']
        total_line = len(lines)
        max_t = 0
        farthest = 0
        # Initialize the maximum length of the clock hand
        for i in range(total_line):
            x1, y1, x2, y2 = lines[i][0]
            length1 = np.sqrt((x1 - center_x)**2 + (y1 - center_y)**2)
            length2 = np.sqrt((x2 - center_x)**2 + (y2 - center_y)**2)
            length = np.max([length1, length2])
            # Find the farthest point from the center of the clock
            if length > farthest:
                farthest = length
                if length == length1:
                    max_line = x1, y1, center_x, center_y
                else:
                    max_line = x2, y2, center_x, center_y
            # Interate through the other lines and calculate the distance between them
```

```
            j = i+1
            while(j<total_line):
                # Find the distance between two lines
                thickness = distance_lines(lines[i], lines[j])
                # Update maximum thickness
                if (thickness > max_t):
                    max_t = thickness
                j += 1
        # Create line with maximum thickness and the farthest point from the center of the clock
        line = max_line, max_t, farthest
        # If the thickness is greater than 0, it means there are at least two parallel lines
        if max_t > 0:
            # Add this set to the clock hands list
            hands_list.append(line)
    # Sort descending the list of clock hands by thickness
    hands_list.sort(key=lambda z: z[2], reverse=True)
    # The three lines with the largest thickness are hours, minutes, and seconds
    hands_list = hands_list[:3]
    return hands_list
```

Figure 2. 6 Step 6

- Step 7: Determine hands

```
#This function determines the hour hand, minute hand, and second hand
def determine_hands(hands_list):
    # Determine the hour hand, minute hand, and second hand based on the thickness and length of the clock hands
    hands_bythickness = sorted(hands_list, key=lambda hands_list: hands_list[1])
    # Get the second hand from the list containing 3 clock hands
    second_hand = hands_bythickness[0]       #The second hand is the thinnest hand
    hands_list.remove(second_hand)           # Remove the second hand from the list of clock hands
    # Arrange the remaining 2 clock hands by length
    hands_bylength = sorted(hands_list, key=lambda hands_list: hands_list[2])
    # Get the hour hand and minute hand from the list containing 2 clock hands
    hour_hand = hands_bylength[0]            # The hour hand is the shorter hand
    minute_hand = hands_bylength[1]          # The minute hand is the longer hand
    return hour_hand, minute_hand, second_hand
```

Figure 2. 7 Step 7

- Step 8: Draw frames

```python
# This function draws a frame around and labels the clock hands back on the image
def draw_hands(img, hour_hand, minute_hand, second_hand):
    # Draw line and add label for hour hand
    x1, y1, x2, y2 = hour_hand[0]
    cv2.line(img, (x1, y1), (x2, y2), (127, 0, 255), 3)
    cv2.putText(img, 'Hour', (int(x1), int(y1)), cv2.FONT_HERSHEY_TRIPLEX  , 1, (127,0,255), 2)

    # Draw line and add label for minute hand
    x1, y1, x2, y2 = minute_hand[0]
    cv2.line(img, (x1, y1), (x2, y2), (0, 102, 0), 3)
    cv2.putText(img, 'Minute', (int(x1), int(y1)), cv2.FONT_HERSHEY_TRIPLEX  , 1, (0,102,0), 2)

    # Draw line and add label for second hand
    x1, y1, x2, y2 = second_hand[0]
    cv2.line(img,(x1,y1),(x2,y2), (0,102,102), 3)
    cv2.putText(img, 'Second', (int(x1), int(y1)), cv2.FONT_HERSHEY_TRIPLEX  , 1, (0,102,102), 2)
```

Figure 2. 8 Step 8

This is the output of this step:

Figure 2. 9 Output of step 8

- Step 9: Calculates the angle of the clock hands

```python
# This function calculates the angle of the clock hands
def get_angle(hand, center_x, center_y):
    # Define the direction of the clock hands as a vector
    x1, y1, x2, y2 = hand[0]
    u = [x2 - x1, y2 - y1]
    # Define the horizontal direction as a vector
    v = [center_x - center_x, center_y - (center_y-100)]
    # Calculate the dot product of two vectors
    dot_uv = u[0] * v[0] + u[1] * v[1]
    # Calculate the length of the two vectors
    length_u = math.sqrt(u[0]**2 + u[1]**2)
    length_v = math.sqrt(v[0]**2 + v[1]**2)

    # Apply cosine formula to calculate the angle between two vectors
    cosine = dot_uv / (length_u * length_v)
    # Ensure that the cosine value is within the range [-1, 1]
    cosine = max(min(cosine, 1.0), -1.0)
    # Calculate the angle by applying the arc cosine function
    thet = math.acos(cosine)
    # Convert the angle from radians to degrees
    degrees = math.degrees(thet)
    # Calculate the cross product of two vectors
    cross_uv = u[0] * v[1] - u[1] * v[0]
    # If the directional product is greater than 0, that means vector u is to the left of vector v
    if cross_uv > 0:
        return 360 - degrees
    # If the directional product is less than 0, that means vector u is to the right of vector v
    else:
        return degrees
```

Figure 2. 10 Step 9

- Step 10: Calculates the time

```python
# This function calculates the time based on the angle of the clock hands
def get_time(hour_angle, minute_angle, second_angle):
    # Hour is calculated by dividing the angle of the hour hand by 30 (each hour corresponds to 30 degrees)
    hour = hour_angle / 30
    # Minute is calculated by dividing the angle of the minute hand by 6 (each minute corresponds to 6 degrees)
    minute = minute_angle / 6
    # Second is calculated by dividing the angle of the second hand by 6 (each second corresponds to 6 degrees)
    second = second_angle / 6
    # If the angle of the hour hand is close to an integer multiplied with 30 (i.e. close to a specific hour) and the angl
    if (round(hour)*30 - hour_angle <= 6) and ((355 < minute_angle and minute_angle < 360) or (minute_angle < 90)):
        hour = round(hour)
        # If the hour is 12, set it to 0
        if hour == 12:
            hour = 0
```

```
    # If the angle of the hour hand is close to a specific hour and the angle of the minute hand is close to 360 (ie
    if (hour_angle - hour*30 <= 6) and (355 < minute_angle and minute_angle < 360):
        # Set minute to 0
        minute = 0

    # If the angle of the minute hand is close to an integer multiplied by 6 (ie close to a specific minute) and the
    if (round(minute)*6 - minute_angle <= 6) and (second_angle < 6):
        #Set second to 0
        minute = round(minute)
        if minute == 60:
            minute = 0

    # If the angle of the minute hand is close to a specific minute and the angle of the second hand is close to 360
    if (minute_angle - minute*30 <= 6) and (354 < second_angle and second_angle < 360):
        # Set second to 0
        second = 0
    # Convert the hour, minute, and second to integers
    hour = int(hour)
    minute = int(minute)
    second = int(second)

    # Format the time in the form of HH:MM:SS
    time = f"{hour:02d}:{minute:02d}:{second:02d}"\

    return time
```

Figure 2. 11 Step 10

- Step 11: Display time

```
# This function display time on the image
def display_time(img, time):
    time = "Time:" + time
    # Font type, size, and thickness
    font = cv2.FONT_HERSHEY_TRIPLEX
    scale = 2
    thickness = 3
    # Where to put the text on the image
    position = (50, 100)
    # Color of the text
    color = (255, 0, 255)
    # Draw the text on the image
    cv2.putText(img, time, position, font, scale, color, thickness)
```

Figure 2. 12 Step 11

Here is the output of this step:

Figure 2. 13 Output for step 11

- The final step is read the image, display and save the results:

```
#Read the image file
img = cv2.imread(img_path)

#Handle the image with the handle_step function
img = handle_step(img)

#Save the image to the output directory
result_path = os.path.join(output_dir, f"{filename}")
cv2.imwrite(result_path, img)

#Display the image
cv2.imshow(filename, img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 2. 14 Step 12

2.2 The result

Here are 10 outputs for this topic:

Figure 2. 15 Output clock 1

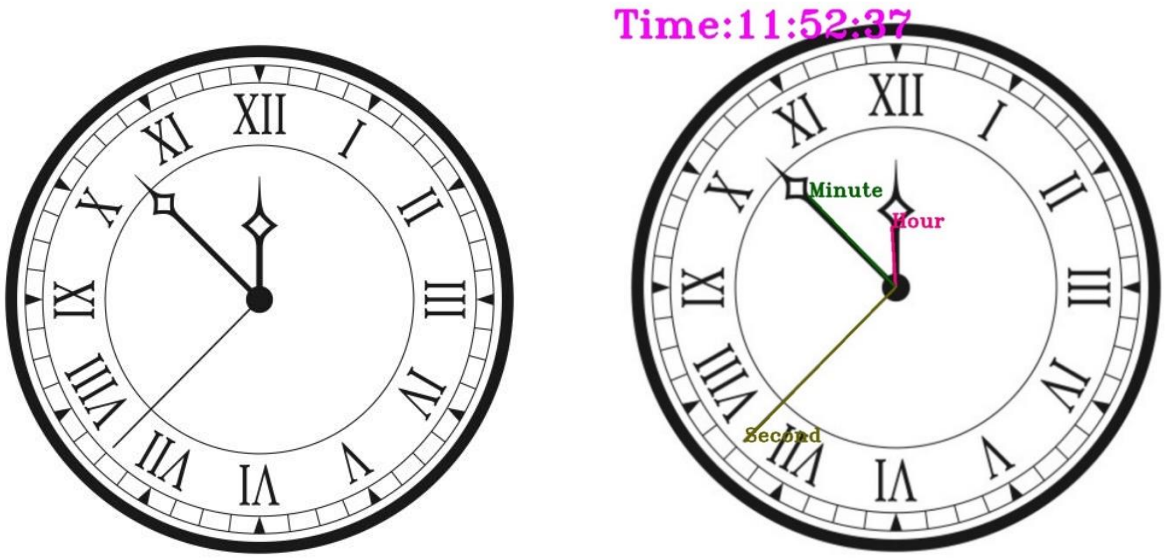


Figure 2. 16 Output clock 2

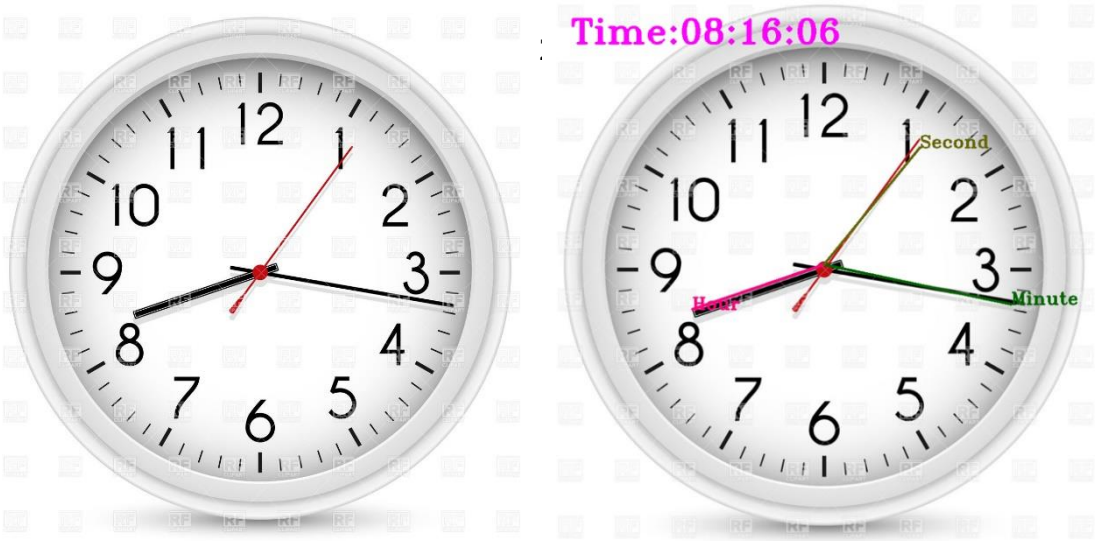


Figure 2. 17 Output clock 3

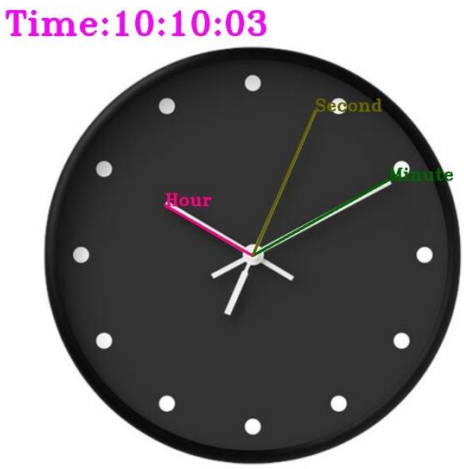


Figure 2. 18 Output clock 4

Figure 2. 19 Output clock 5

Figure 2. 20 Output clock 6

Figure 2. 21 Output clock 7

Figure 2. 22 Output clock 8

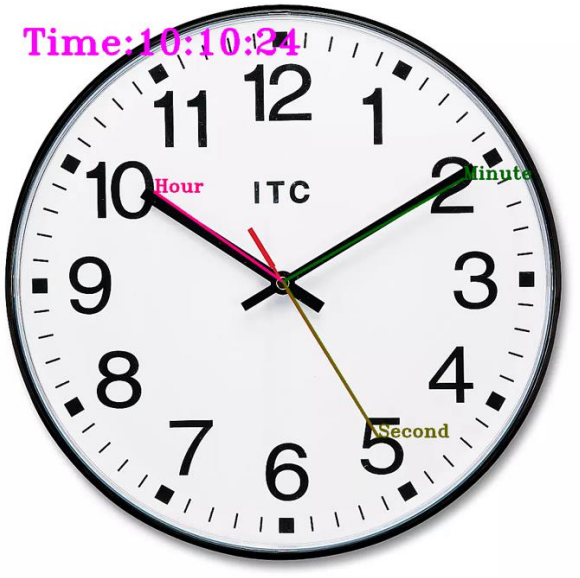


Figure 2. 23 Output clock 9

Figure 2. 24 Output clock 10

REFERENCE

W3School, “NumPy Introduction”.

W3school, “Python os Module”.

Geeksforgeeks, “Find and Draw Contours using OpenCV”, 04 Jan, 2023.

Geeksforgeeks, “Line dectection in python with OpenCV”, 03 Aug, 2023.