2022-2023

**Communication and Mini Project**

LEBANESE UNIVERSITY

FACULTY OF ENGINEERING – BRANCH III

Specialty: Electrical and Electronics Engineering

Submitted by:

Ali Dbouk 6164

Mohammad Mahdi Kassem 6200

Ali Saleh 6097

Mostafa Kabalan 6183

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Under the Supervision of:

Dr. Houda Ghamlouch

Table Of Contents

[Table Of Contents 2](#_Toc137309244)

[General Introduction 3](#_Toc137309245)

[Introduction 4](#_Toc137309246)

[1.1. Objectives 4](#_Toc137309247)

[1.2. Code 5](#_Toc137309248)

[1.3. How the code works 7](#_Toc137309249)

[1.4. Error Analysis: 17](#_Toc137309250)

[1.5. Ideas For Project Development 19](#_Toc137309251)

[1.6. Conclusion 19](#_Toc137309252)

[References 20](#_Toc137309253)

General Introduction

The automated dimension measuring project aims to develop a system that automates the process of scanning and measuring the dimensions accurately. Traditional document scanning methods often require manual adjustments and measurements, which can be time-consuming and prone to errors. This project utilizes computer vision techniques and the OpenCV library to create a more efficient and reliable dimensions scanning solution.

Introduction

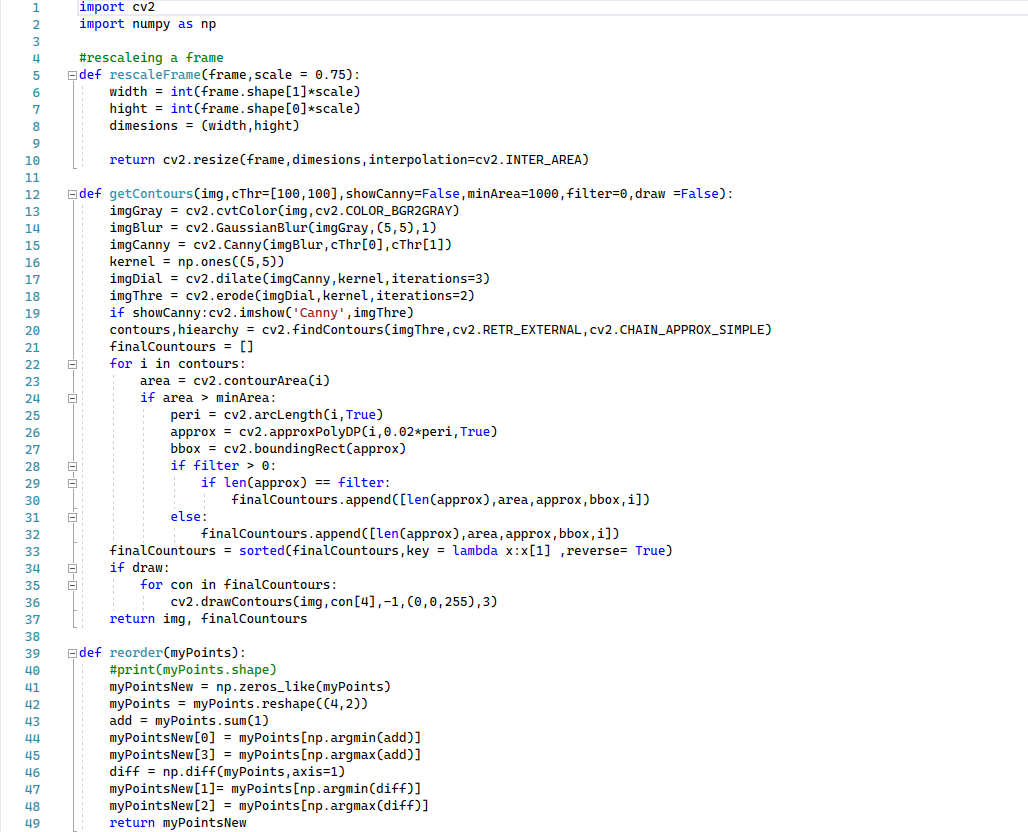
The purpose of this report is to provide an overview of the automated dimension measuring project. The project aims to develop a system that can scan documents, detect their contours, and calculate their dimensions accurately. This report will discuss the objectives of the project, explain the functions used in the code, and provide suggestions for further development.

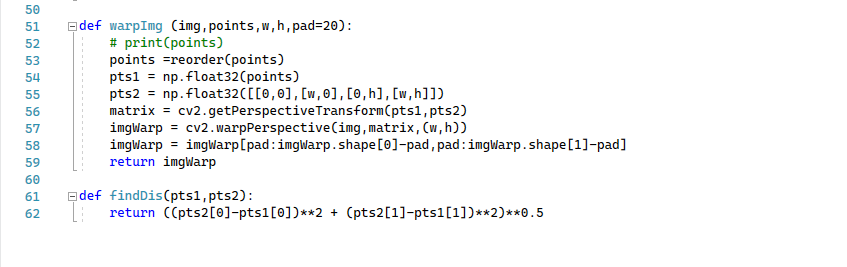
* 1. Objectives

The main objectives of the automated dimension measuring project are as follows:

* Develop a system to automatically scan object using either a webcam or an input image.
* Detect the contours of the A4 paper in the captured image
* Calculate the 2D dimensions of the object by using a known reference dimension (in our case the A4 paper [29.7 cm , 21 cm ].
* Display the scanned object with annotated dimensions.
* Enable users to accurately measure the length and width of the object using the displayed dimensions.
  1. Code

The code is divided into two files: utlis which contains the functions used in the program, and file objectMeasurment which contains the main program of our project

* Utlis file:



* objectMeasurment File



* 1. How the code works

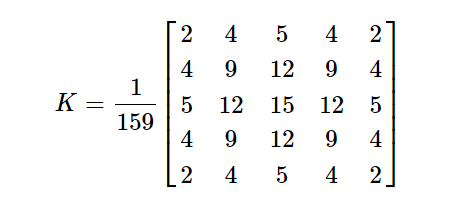
The provided code is an implementation of an automated dimension measuring using OpenCV. First, we will explain each function and how it works in part A, and second we will go through the main program and explain how it works step by step in part B.

1. **Functions Definitions**
2. **Get Contours:**
3. **Function Definition:**

* The Get Contours function takes several parameters, including the input image (img), threshold values for Canny edge detection (cThr), a flag to show the Canny output (showCanny), a minimum area threshold (minArea), a filter value (filter), and a flag to draw contours on the image (draw).
* The function returns the image with drawn contours and a list of final contours.

1. **Preprocessing:**

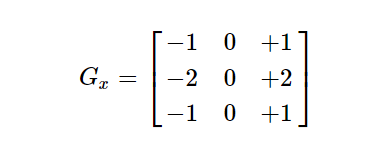
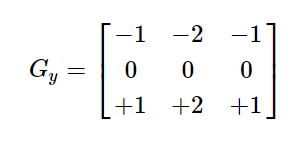
* The input image is converted to grayscale using cv2.cvtColor() to prepare it for further processing.
* Gaussian blur is applied to the grayscale image using cv2.GaussianBlur() to reduce noise. An example of a Gaussian kernel of size=5 that might be used is shown below:



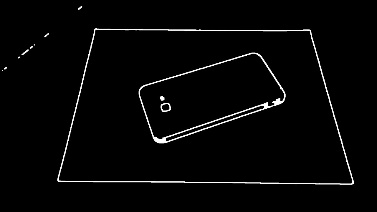
* Canny edge detection is performed on the blurred image using cv2.Canny() with the specified threshold values.

Find the intensity gradient of the image. For this, we follow this procedure:

* Apply a pair of convolution masks (in x and y directions):



* Find the gradient strength:
* Dilations and erosions are applied to the Canny output using cv2.dilate() and cv2.erode() respectively to enhance the contour shapes and fill any gaps.



1. **Contour Detection:**

* The contours are detected using cv2.findContours() on the preprocessed image.
* The function returns a list of contours and the contour hierarchy.

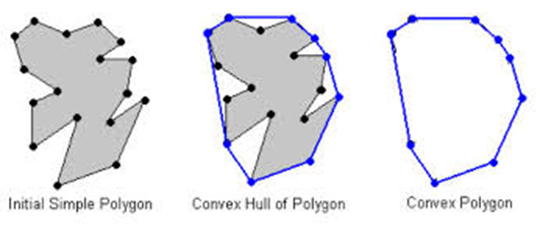
**NOTE:** In cv2.findContours(), finding contours is like finding a white object from a black background. So remember, the object to be found should be white and the background should be black.

1. **Contour Filtering:**

* The code iterates through the detected contours and filters them based on their area.
* Contours with an area smaller than the specified minArea are ignored.
* If a filter value is provided, contours with a different number of approximated points are also filtered.
* Approximating is done using

cv2.approxPolyDP (0.02\*perimeter(contour)) which uses Ramer-Douglas-Peucker algorithm.

* The purpose of the algorithm is, given a curve composed of line segments (which is also called a Polyline in some contexts), to find a similar curve with fewer points. The algorithm defines 'dissimilar' based on the maximum distance between the original curve and the simplified curve (i.e., the Hausdorff distance between the curves). The simplified curve consists of a subset of the points that defined the original curve.



1. **Sorting Contours:**

* The final filtered contours are sorted based on their area in descending order using the sorted () function.

1. **Drawing Contours (Optional):**

* If the draw flag is set to True, the function draws the final contours on the input image using cv2.drawContours().

1. **Return:**

* The function returns the input image (possibly with drawn contours) and the final filtered contours.



1. **Reorder function:**
2. **Function Definition:**
   * The reorder function takes a single parameter, which represents the input set of points.

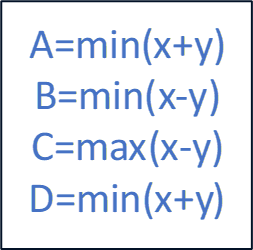
ex:[C,B,A,D]

1. The function returns a new array that contains the reordered points.

ex:[A,B,C,D]

1. **Reshaping the Input:**
   * The myPoints array is reshaped to have a shape of (4, 2). This assumes that myPoints contains four points, where each point has two coordinates (x, y).
2. **Computing Sums and Differences:**
   * The sums of the x and y coordinates for each point are computed using the sum() function along the second axis (axis=1). This results in an array add containing the sums.
   * The differences between the x and y coordinates for each point are computed using the diff() function along the first axis (axis=1). This results in an array diff containing the differences.
3. **Reordering Points:**
   * The first point in myPointsNew is set as the point with the smallest sum of coordinates (np.argmin(add)).
   * The last point in myPointsNew is set as the point with the largest sum of coordinates (np.argmax(add)).
   * The second point in myPointsNew is set as the point with the smallest difference between its coordinates (np.argmin(diff)).
   * The third point in myPointsNew is set as the point with the largest difference between its coordinates (np.argmax(diff)).





**Such as:**

1. **Return:**
   * The function returns the myPointsNew array containing the reordered points.



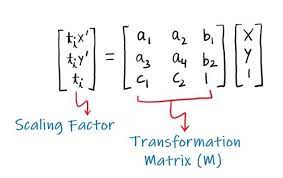
1. **Perspective Transformation (WrapImg function):**
2. **Function Definition:**

As clear from the name, the perspective transformation is associated with a change in the viewpoint. This type of transformation does not preserve parallelism, length, and angle. But they do preserve collinearity and incidence. This means that the straight lines will remain straight even after the transformation.

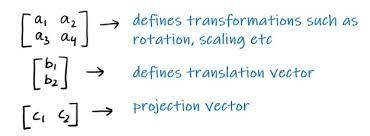
1. **Point Reordering:**

The points array is reordered using the reorder function to ensure the points are in the correct order for the perspective transformation.

1. **Perspective Transformation:**
   * The reordered points are converted to floating-point arrays (pts1 and pts2) to be used as inputs for cv2.getPerspectiveTransform().
   * cv2.getPerspectiveTransform() computes the perspective transformation matrix (matrix) based on the source (pts1) and destination (pts2) points.
   * Since the transformation matrix (M) is defined by 8 constants thus to find this matrix we first select 4 points in the input image and map these 4 points to the desired locations in the unknown output image according to the use-case (This way we will have 8 equations and 8 unknowns and that can be easily solved).



**The transformation matrix (M) can be seen as a combination of**



1. **Warping:**
   * The perspective transformation is applied to the input image (img) using cv2.warpPerspective(). The resulting warped image is stored in imgWarp.
   * To remove any padding that was added during the perspective transformation, a region of interest (ROI) is selected from the warped image by removing the padding on all sides.
   * The ROI is extracted from the warped image using array slicing and stored in imgWarp.
2. **Return:**
   * The function returns the resulting warped image (imgWarp).

1. **Find Distance Function:**
2. **Function Definition:**
   * The find distance function takes two parameters: pts1 and pts2, which represent the coordinates of two points in a two-dimensional space.
   * It returns the Euclidean distance between the two points.
3. **Euclidean Distance Calculation:**
   * The Euclidean distance between two points (x1, y1) and (x2, y2) in a two-dimensional space is calculated using the formula:
4. **Return:**
   * The function returns the computed Euclidean distance between the two points.
5. **Main Program Code**
6. **Importing Libraries:**

The code starts by importing the necessary libraries, including 'cv2' for computer vision tasks and 'utlis' for utility functions specific to this project.

1. **Initializing Variables:**

The code sets up various variables to configure the scanner's behavior, such as the source (webcam or image file), path to the image file, video capture object, camera settings, scale for image processing, and desired width and height for the document.

1. **Main Loop:**

The code enters an infinite loop, where it continuously captures frames from the video source (webcam or image) and performs the scanning and measurement operations.

1. **Image Capture:**

Inside the loop, the code checks whether the source is a webcam. If it is, it uses the 'cap.read()' function to retrieve the next frame from the webcam. If the source is an image file, it uses the 'cv2.imread()' function to read the image from the specified path.

1. **Contour Detection:**

The code calls the 'utlis.getContours()' function, passing the captured image and minimum contour area as parameters. This function utilizes OpenCV's contour detection algorithms to find contours in the image. It returns the image with drawn contours and a list of contours.

1. **Document Extraction:**

The code checks if any contours were detected. If contours are present, it assumes the largest contour represents the document. It extracts the document by calling the 'utlis.warpImg()' function, passing the captured image, the largest contour, and the desired width and height for the document.

1. **Contour Detection on Extracted Document:**

The code calls the 'utlis.getContours()' function again, but this time on the extracted document image. It applies a different set of parameters for contour detection on the smaller region of interest. The purpose is to detect contours within the extracted document, such as edges or features.

1. **Dimension Calculation and Annotation:**

If contours are found within the extracted document, the code enters a loop to process each contour. It draws polylines around the contours and calculates the dimensions of the document by using reference points. The 'utlis.reorder()' function is called to ensure the corner points of the contour are in a consistent order. The dimensions are calculated using the 'utlis.findDis()' function, which measures the distances between reference points.

1. **Annotation and Display:**

The code uses various OpenCV functions, such as 'cv2.polylines()', 'cv2.arrowedLine()', and 'cv2.putText()', to annotate the extracted document image with arrows indicating the measured dimensions. The annotated image is then displayed using the 'cv2.imshow()' function.

1. **Displaying the Original Image:**

The original captured image is resized using the 'cv2.resize()' function and displayed in a separate window using 'cv2.imshow()' to provide a side-by-side view of the original and processed images.

1. **Exiting the Program:**

The code waits for a key press using 'cv2.waitKey()', and if the 'd' key is pressed, it breaks out of the infinite loop, and the program execution stops.

This code continuously captures frames, detects document contours, extracts the document, calculates its dimensions, and displays the annotated image in real time. Users can use this automated dimension measuring to measure the length and width of documents accurately.



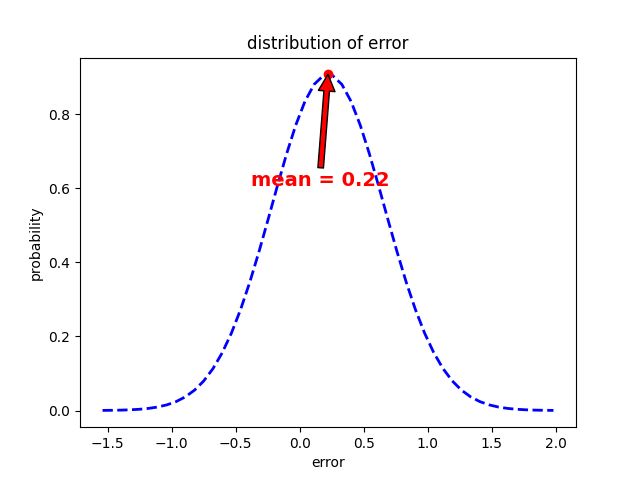


* 1. Error Analysis:

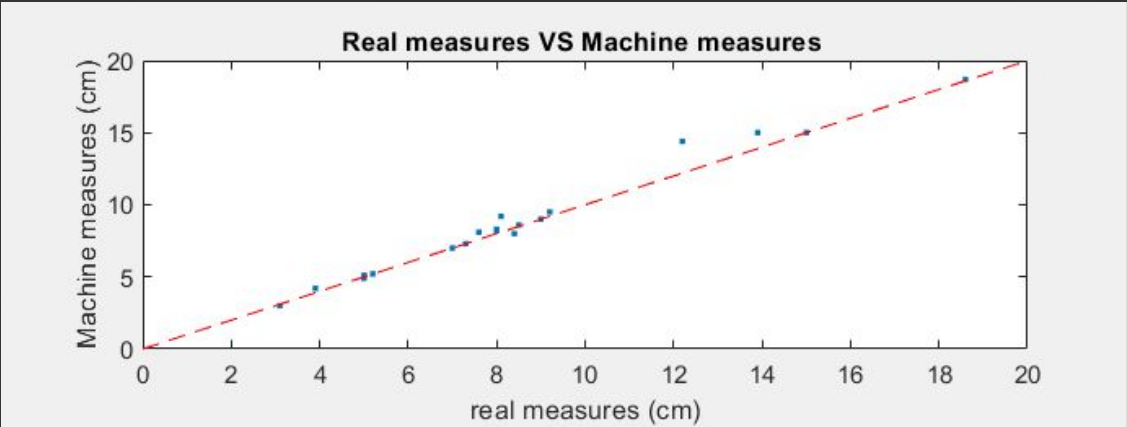
1. Introduction:

In this section, we will analyze the measurement errors. The aim of this analysis is to provide an understanding of the accuracy and precision of the measurements taken by the software. The data used for analysis includes the comparison between the true measurements and the machine measurements of approximately 24 sampled items. The recorded error values will be discussed in terms of their mean and standard deviation.

1. Error Measurement Results:

* Upon analyzing the collected data, it was found that the mean error between the true measurements and the machine measurements was approximately 0.22 cm. This implies that, on average, the machine measurements deviated from the true measurements by 0.22 cm.
* The standard deviation of the error was determined to be 0.44 cm. The standard deviation is a measure of the spread of the error values around the mean error. In this case, a standard deviation of 0.44 cm suggests that the majority of the errors fell within a range of ±0.44 cm from the mean error value.

Here is a graph done using Matlab shows the sample of measures measured using the program in comparison to its real measures



1. Interpreting the Results:

The mean error of 0.22 cm suggests a slight systematic bias in the measurements obtained from the object measurement software. This bias could be justified as follow:

* Merging the object shadow in the object’s contour
* Merging the object side thickness in the object’s contour
* Bended A4 edges
* Misleading objects or noisy objects (contain many details)

Based on the error analysis, it is evident that the object measurement software provides reasonably accurate measurements, as indicated by the low mean error value. However, there is room for improvement to reduce the bias observed.

* 1. Ideas For Project Development
* Automatic document classification: Extend the project to automatically classify different types of documents based on their shapes or content. This could be achieved by training a machine learning model on a dataset of various document types.
* Optical character recognition (OCR): Implement OCR techniques to extract text from the scanned documents, enabling users to search and analyze the text within the documents.
* Cloud integration: Allow users to store the scanned documents in cloud storage services such as Google Drive or Dropbox for easy access and sharing.
  1. Conclusion

The automated dimension measuring project offers a practical solution for scanning and measuring documents accurately. By leveraging OpenCV's computer vision capabilities, the system can detect document contours, calculate their dimensions, and display the results. The project has the potential for further development and integration with additional features like document classification, OCR, and cloud storage. Overall, this project showcases the capabilities of computer vision in automating dimension-measuring processes.

References

* <https://theailearner.com/2020/11/06/perspective-transformation/>
* <https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html>
* <https://docs.opencv.org/4.x/da/d5c/tutorial_canny_detector.html>
* <https://stackoverflow.com/questions/41402137/opencv-splitting-contours>
* Freecodecamp : <https://youtu.be/oXlwWbU8l2o>
* [https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker\_algorithm#](https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm)