Analog Clock Reader

Computer Vision - COMP 4102

Final Project

Layne Koftinow-Mikan (101013563)
Majd Al Khany (100977501)
April 18, 2020

Abstract

The analog clock reader consists of a set of algorithms, the input to which is an image of an analog clock and the output of which is simply a string representation of the time on the clock. The code for this project was written in Python and makes use of the OpenCV and Tesseract computer vision libraries. The detectClock function in detectClock.py calls a series of functions in the following order: alignClock, orientClock, isolateClock, detectClockHands, and calculateTime. One of the goals of this project was to create a robust algorithm that provides a simple interface. While the input is a single image and the output is a single string, the program is designed to account for many different cases of an analog clock image. This includes the ideal case wherein a clock has an hour, minute, and second hand, numbers, and is directly facing the camera. In addition to this, the algorithm also successfully supports non-ideal cases wherein the clock is misaligned, misoriented, or missing potential data points such as hands and numbers. A test framework was also developed which determines the accuracy of the algorithm by comparing the expected results to the actual results and computing the difference.

Introduction

One of the goals of this project was to abstract away a robust algorithm and provide a simple interface through which a user can interact with the application. The input is an image of an analog clock and the output is a string representation of the time on the clock. There are multiple steps to this algorithm, each with their own internal steps. The general approach is as follows: aligning the clock if it is misaligned, rotating the clock if it is misoriented, isolating the clock from the rest of the image, detecting the hands on the clock, and finally calculating the time using the detected hands. This program is a good candidate to be placed into a vision application because of its practical use and robustness. It could theoretically be part of a computer vision mobile app that allows users to take photos of clocks or watches and detects the time on them. As previously mentioned, the main challenging aspect of this project lies in its versatility. The ideal case for detecting a clock is relatively straightforward and has been programmed many times; this project successfully implements the ideal case and expands upon it by supporting the following cases: a clock without a seconds hand, a clock without numbers, a clock that is misaligned (ie. the photo was taken from an angle and the clock does not directly face the camera), and a clock that is misoriented (ie. the clock is rotated by some unknown degree). Note that this algorithm supports both clocks and watches but the word “clock” is used to refer to both throughout this report.

Background

Canny edge detector

OpenCV function: Canny
The Canny edge detector is used to detect edges in an image (areas of significant local change of intensity). Canny edge detection involves blurring the image, computing gradient components, estimating edge strength and orientation, applying non-maxima suppression, and then hysteresis thresholding [1].

Canny edge detection is to aid in the detection of the outer perimeter of the clock as well as the clock's hands.

Suzuki contour detection

OpenCV function: findContours
The Suzuki contour detection algorithm determines the surroundedness relations among the borders of a binary image, allowing the extraction of image features without reconstructing the image [2]. This algorithm is used to find the perimeter of a skewed clock and help extract features from the image which are used to realign it (ie. warp the clock so it is circular).

Image blurring

OpenCV function: medianBlur
Images are blurred by applying low-pass kernel convolution to an image. This is done for a variety of reasons using many different filters. Median blur, which uses a non-linear filter, was chosen because it is particularly good at removing noise while preserving edges which is important for this algorithm [3]. In this algorithm, images are blurred before Hough transforms are applied to them to optimize feature detection.

Hough transform

OpenCV functions: HoughCircles, HoughLinesP
The Hough transform is a method of isolating features of a particular shape in an image, such as lines and circles. This technique works well because it is tolerant of gaps and is relatively unaffected by image noise [4]. HoughCircles is used to detect the outer circumference of the clock and HoughLinesP, the probabilistic variant of the Hough transform, is used to detect the lines representing the clock's hands.

Text Detection

OpenCV function: cv2.dnn.readNet
Library: EAST DNN Text Detector
The EAST deep neural network is a text detection library loaded into OpenCV, which is used to detect text in an image. This is the first step in determining the orientation of the clock in the input image. In order to apply OCR on the image, EAST detects any text and places a bounding box around it, which aids the OCR engine in finding the text and interpreting it [5].

OCR (Optical Text Recognition)

OCR Engine: PyTesseract
PyTesseract is an OCR engine based on Tesseract, adapted for use with Python. PyTesseract does not come included with Python or OpenCV and needs to be installed separately. After detecting any text in the image using EAST, PyTesseract is then used to apply OCR and return the text in the image. PyTesseract works by taking the input image and producing a binary image through adaptive thresholding, then produces character outlines through connected component analysis, then performs

various calculations to recognize the words and characters in the image. Afterwards, PyTesseract passes the produced words/characters through a word list before giving a final output [6]. This is the second step in detecting the orientation of the clock in the image; if PyTesseract successfully detects the numbers on the clock, that means that the image is in the correct orientation. If it does not, then that means that the image is in the incorrect orientation, as PyTesseract cannot read characters that are not horizontal (ie. not tilted sideways or upside down).

Image Rotation

Function: imutils.rotate
The imutils.rotate function is an image rotation function from the imutils library. This function takes an image as input and the number of degrees to rotate the image, then returns the rotated version of that image. This is applied to images in which OCR was unsuccessful and no hours were detected, in order to attempt to re-orient the image.

Approach

Algorithm Steps

Launch command: python detectClock.py <image.jpg>
Test framework launch command: python testDetectClock.py

Input: clock image

1. Align clock if it is sufficiently skewed, otherwise proceed to the next step (alignClock.py)
    a. Detect edges using Canny edge detector
    b. Use edges to detect clock contour using Suzuki contour detection
    c. Calculate the boundingRect and minEnclosingCircle of the contour
    d. Calculate the transformation matrix using boundingRect and minEnclosingCircle values
    e. Warp perspective using the transformation matrix
2. Orient clock if numbers cannot be detected, otherwise proceed to the next step (orientClock.py)
    a. Detect text (numbers on the clock) in the input image using the EAST DNN text detector
    b. Place bounding boxes around the detected text in the image (for the OCR engine to be able to find the text and interpret it)
    c. Pass the image and data computed in the previous step by the EAST text detector to the PyTesseract OCR engine (in order to read the text on the image)
    d. If PyTesseract was able to successfully read the numbers on the clock image, then the image is in the correct orientation, return the image in its current state; otherwise,
        i. Use imutils.rotate to rotate the image by an interval of 10 degrees, and repeat the steps above until the image is in the correct orientation, at which point the image will be returned in the corrected orientation
        ii. If the image gets rotated 360 degrees or more and PyTesseract still cannot read the text/numbers correctly, then all rotations are discarded and the image is returned in the same orientation that it was originally passed

3. Isolate clock by cropping the image so the clock's center is the image center (isolateClock.py)
    a. Convert image to greyscale and apply medianBlur
    b. Apply Hough transform to detect the clock's outer circumference
    c. Crop the image around this circumference
4. Detect the clocks hands (detectClockHands.py)
    a. Detect edges using Canny edge detector
    b. Detect lines using probabilistic Hough transform on these edges
    c. Remove lines that cannot represent clock hands (ie. do not pass near center)
    d. Merge similar lines together into one line (since most hands will have more than one line corresponding to them)
    e. Estimate which lines represent hour, minute, and second hands using their length and thickness, resulting in a list of 2-3 lines (seconds hand is optional)
5. Calculate time using clock hands (calculateTime.py)
    a. Calculate angle of each hand (eg. 12 o'clock is 0 degrees, 3 o'clock is 90 degrees, etc.)
    b. Use angles of each hand to calculate time value
    c. Format time values and return a single string

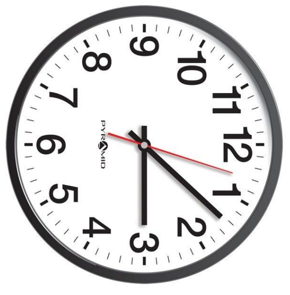
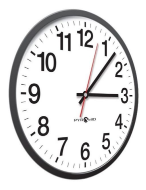
Output: string representing the time on the clock (hh:mm:ss)

Comparison

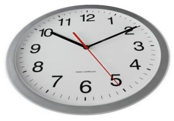
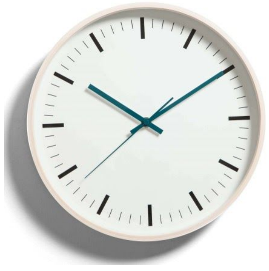
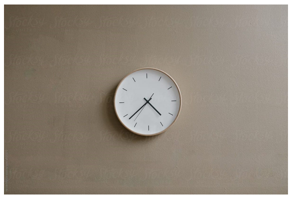
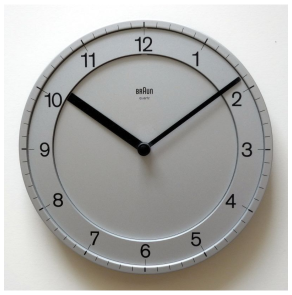
When compared to similar implementations of analog clock time detection, our implementation has an advantage when it comes to robustness. In similar implementations, the image needs to show the clock in the most ideal position possible. The clock needs to be facing straight forward in the correct orientation, should not be skewed, and should always have three hands [7]. We designed our algorithm to account for instances in which the clock in the image might not be in the most ideal position. As mentioned above, our implementation begins by first detecting if the clock image is skewed then adjusts the image to account for it, in order to correctly calculate the time. Our algorithm then proceeds to detect the orientation of the clock in the image, and applies sufficient rotation to re-orient it if necessary. The code is also robust enough to account for clocks which may not have a seconds hand.

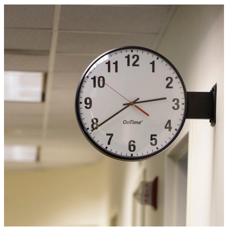
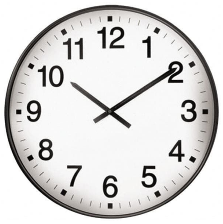
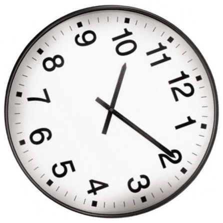
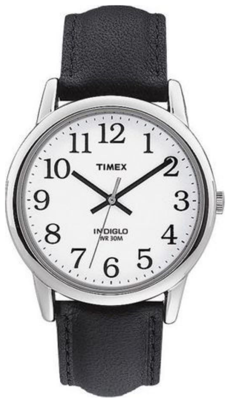
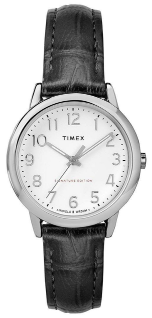
Results

The table below shows the clock images and their corresponding names, which we used as our test cases, with a screenshot of the testing results below.

clock1.jpg	clock1_rotated.jpg	clock1_skew.jpg	clock2.jpg

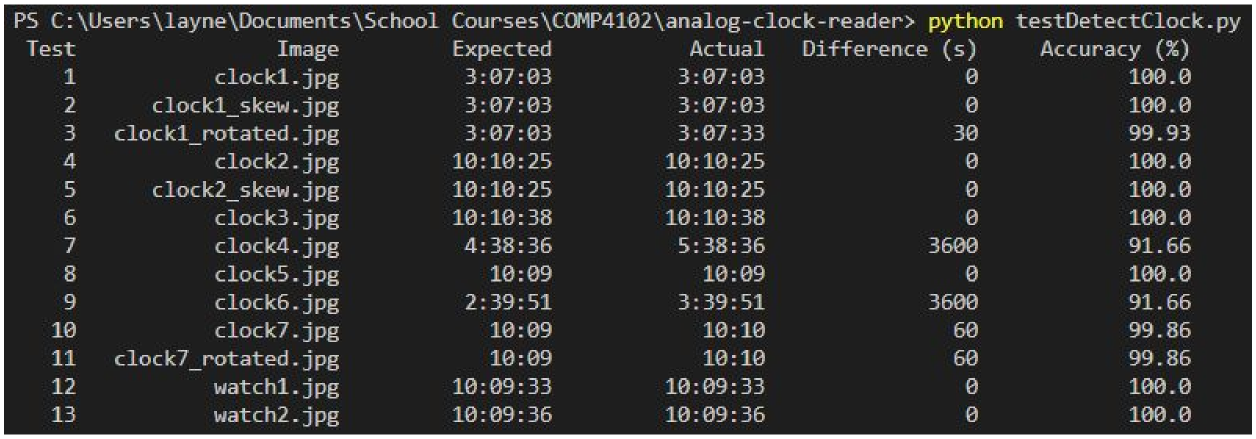
			
clock2_skew.jpg	clock3.jpg	clock4.jpg	clock5.jpg

				
clock6.jpg	clock7.jpg	clock7_rotated.jpg	watch1.jpg	watch2.jpg

These images cover all the cases which we had initially sought to cover:

- The “ideal” case (clock1, clock2, clock7, watch1, watch2)
- Clocks without seconds hands (clock4, clock5)
- Clocks without numbers (clock3, clock4)
- Misoriented clocks (clock1_rotated, clock7_rotated)
- Misaligned clocks (clock1_skew, clock2_skew).
- Clocks photographed in a natural environment (clock4, clock5, clock6)

Below is a screenshot of the test framework output which shows the results of passing all of the above images into our program, showing the expected time compared to the time calculated from the images. Each row corresponds to the images above in the same order they are shown with the name under the “Image” column.



```
PS C:\Users\layne\Documents\School Courses\COMP4102\analog-clock-reader> python testDetectClock.py
 Test              Image         Expected          Actual   Difference (s)     Accuracy (%)
   1          clock1.jpg          3:07:03         3:07:03                0            100.0
   2     clock1_skew.jpg          3:07:03         3:07:03                0            100.0
   3  clock1_rotated.jpg          3:07:03         3:07:33               30            99.93
   4          clock2.jpg         10:10:25        10:10:25                0            100.0
   5     clock2_skew.jpg         10:10:25        10:10:25                0            100.0
   6          clock3.jpg         10:10:38        10:10:38                0            100.0
   7          clock4.jpg          4:38:36         5:38:36             3600            91.66
   8          clock5.jpg            10:09           10:09                0            100.0
   9          clock6.jpg          2:39:51         3:39:51             3600            91.66
  10          clock7.jpg            10:09           10:10               60            99.86
  11  clock7_rotated.jpg            10:09           10:10               60            99.86
  12          watch1.jpg         10:09:33        10:09:33                0            100.0
  13          watch2.jpg         10:09:36        10:09:36                0            100.0
```

Accuracy was determined by comparing the expected time to the actual time, taking into account that there are 43,200 possible times on a 12-hour clock (60 seconds * 60 minutes * 12 hours). For the most part, our algorithm was able to successfully calculate the time correctly, with the accuracy rate being 100% for 8 out of the 13 test cases. The other 5 test cases had an accuracy ranging from 91% to 99%. In most cases, this difference is due to what essentially amounts to rounding issues resulting in "off by one" errors. This discrepancy is more pronounced when the hour hand is off by one (3600 seconds) compared to the minute hand (60 seconds). For example, in clock7, our program calculated the time as 10:10 when in reality the time is actually 10:09.

Another issue is the miscalculation of the time for certain images, such as in clock1_rotated, in which the seconds were calculated to be 33 instead of 03. That is due to the line detection algorithm not detecting the clock hands properly. The line is detected to be closer to the opposite side so the algorithm thinks the hand is facing the opposite direction, resulting in a 30 second difference. This behaviour was only observed for the seconds hand because they have more of a tendency to pass through the center of the clock and onto the opposite side.

One trivial issue with numberless clocks is that the program cannot detect whether they are in the correct orientation or not. When an image like clock3 is passed into the orientClock function, the image is continually rotated and scanned for text using an OCR algorithm until it hits 360 degrees, at which point orientClock decides that it cannot read any text and returns the image in its original orientation. While this issue is essentially unavoidable (afterall, it is difficult even for humans to know whether a clock without numbers is oriented correctly), it is very inefficient to keep trying to detect numbers on the clock when there aren't any. It leads to a noticeably significant slowdown in the time calculation. For this reason, the test script skips this step for clocks which are already known to be oriented correctly. Note that this issue only impacts the algorithm's efficiency and not its accuracy.

List of Work

The following features were implemented by Layne:

- Clock alignment (alignClock.py)
- Hand detection (detectClockHands.py)

- Test framework (testDetectClock.py)

The following features were implemented by Majd:
- Clock isolation (isolateClock.py)
- Clock orientation (orientClock.py)
- Time calculation (calculateTime.py)

The following features were implemented by both:
- The main function that runs through the algorithm steps (detectClock.py)
- Other utility functions and globals (utils.py, globals.py)

GitHub

Link: https://github.com/majdalkhany/analog-clock-reader

References

[1] J. Canny, “A Computational Approach To Edge Detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6): 679–698, 1986.

[2] S. Suzuki and K. Abe, “Topological Structural Analysis of Digitized Binary Images by Border Following,” CVGIP 30(1): 32-46, 1985.

[3] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer: 109, 2010.

[4] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, “Hough Transform,” Image Processing Learning Resources, 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm. [Accessed: 15-Apr-2020].

[5] A. Rosebrock, “OpenCV OCR and text recognition with Tesseract,” *PyImageSearch*, 17-Sep-2018. [Online]. Availablehttps://www.pyimagesearch.com/2018/09/17/opencv-ocr-and-text-recognition-with-tesseract/?fbclid=IwAR1gD8J2PCHV1M3QTbVltQ2iJdPwkvy2KLQkRYuinFnP4YUtdUk02Z0yovU. [Accessed: 16-Apr-2020].

[6] M. Gjoreski, G. Madjarov, D. Gjorgjevikj, and H. Gjoreski, “Optical character recognition applied on receipts printed in Macedonian language,” International Conference on Informatics and Information Technology, pp. 1-4, Apr. 2014.

[7] C. Schumacher, “Analog clock and watch reader,” Ben-Gurion University of the Negev. [Online]. Available: https://www.cs.bgu.ac.il/~ben-shahar/Teaching/Computational-Vision/StudentProjects/ICBV151/ICBV-2015-1-ChemiShumacher/Report.pdf. [Accessed: 16-Apr-2020].