Analog clock and watch reader

Final project by
Chemi Shumacher
shumache@post.bgu.ac.il

Introduction

Telling time is a task no-one thinks much of. People learn it at a fairly early age, and then it becomes a second nature to us. You look at your $clock$[1], and within less than a second your brain processes the image and translates a circle with three hands into meaningful data. But have you ever thought about how it happens?

Think back to before you learned how the clock worked: It must have looked like gibberish to you. How did you learn how to read an analog clock? You must have been instructed to follow these simple steps:

1. Look for the shortest hand, check after what number it is. This is the hour.
2. Look for the longer thicker hand. Multiply the number it points to by five. That's the current number of minutes.
3. Look for the longest, thin hand. Multiply the number it points to by five. That's the current number of seconds.

When approaching this project, I wondered – how would a computer be able to decode the information that an analog clock contains? How can it tell which hands are which? And how can it know which *number* they point at?

I attempted to answer all these questions by dividing the identification of the provided image into a number of stages. In order to do this, I had to make a number of assumptions:

- The clock is relatively large in the image. Meaning, the focus of the image is on the clock, and it is not somewhere small in the background.
- The clock background is relatively uniform and uncluttered.
- Clock hands are uniform in shape and color.

My main goal for this project was to develop a model for analog clock identification, and to implement it with a program that would be able to receive an image of a clock, and return the time displayed after its analysis. I strived to write a program that would reach an acceptable success rate, and would work on a broad range of images.

1 In this report I shall refer to both "clocks" and "watches" as "clock".

Approach and Method

The approach taken in this project was to isolate the actual clock from the rest of the image, identify the clock hands, and calculate the time according to their angles.

The method for doing this consists of six steps:

I. Isolate the clock from the rest of the image
II. Apply edge detection to the image
III. Use Hough transform to isolate watch hands
IV. Calculate line angles
V. Filter out irrelevant lines
VI. Compute the time

I. Isolate the clock from the rest of the image

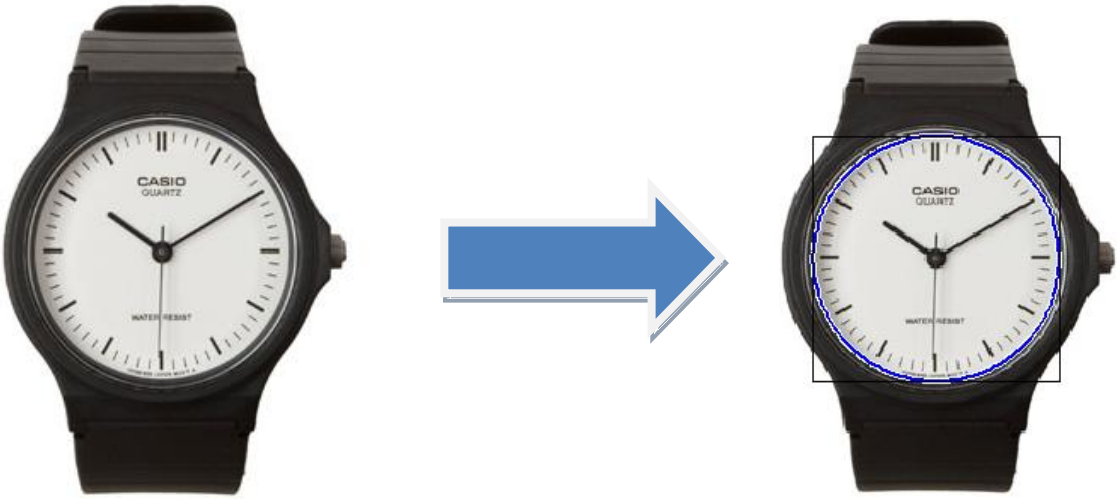
Given an image of a clock, we must first isolate the part of the image we are interested in. For this, we use the Hough Transform, a feature extraction technique. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. In this procedure, we apply shapes in various orientations and positions. Each pixel who takes part of the shape will "vote" for it. The shape with the largest amount of votes consists of the largest amount of pixels, and is therefore most likely to be an actual shape in the image. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

Specifically, we use the $circle\ Hough\ Transform$[1], a specialization of Hough Transform.

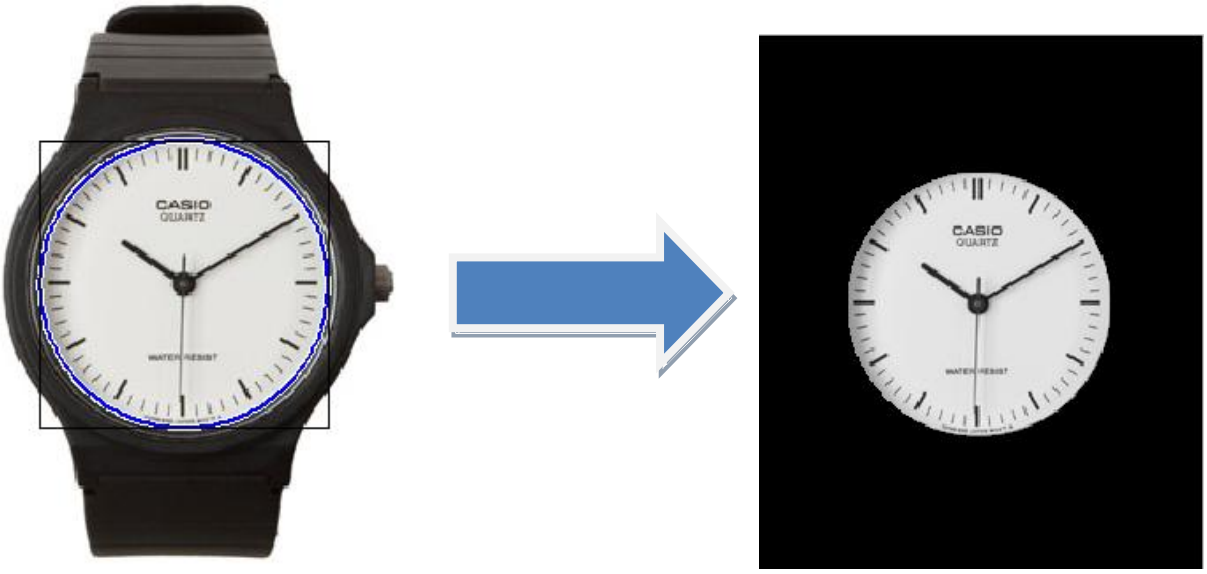
The process of identifying possible circular objects in Hough space is as follows,

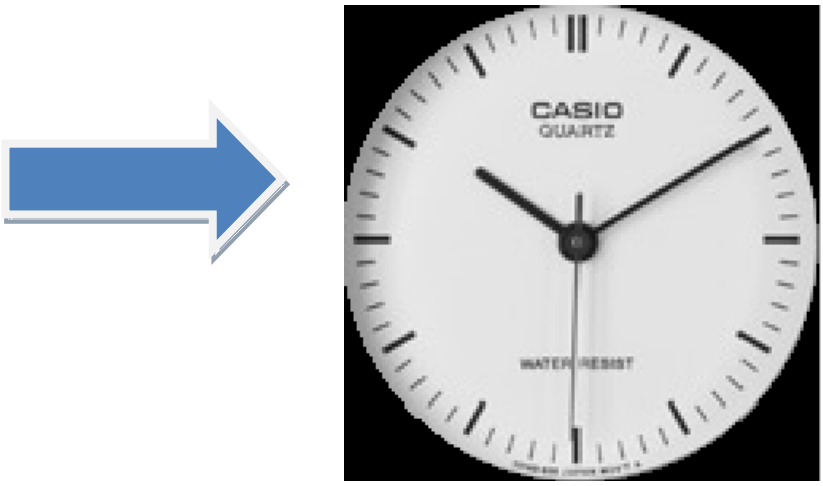
- First we create our accumulator space which is made up of a cell for each pixel, initially each of these will be set to 0.
- For each (edge point in image(i, j)): Increment all cells which according to the equation of a circle($(i-a)^2 + (j-b)^2 = r^2$) could be the center of a circle, these cells are represented by the letter 'a' in the equation.
- For all possible value of a found in the previous step, find all possible values of b which satisfy the equation.
- Search for the local maxima cells, these are any cells whose value is greater than every other cell in its neighborhood. These cells are the one with the highest probability of being the location of the circle(s) we are trying to locate.

1 In a two dimensional space, a circle can by described by: $(\boldsymbol{x}-\boldsymbol{a})^2+(\boldsymbol{y}-\boldsymbol{b})^2=\boldsymbol{r}^2$, where (a,b) is the center of the circle, and r is the radius. If a 2D point (x,y) is fixed, then the parameters can be found according to (1). The parameter space would be three dimensional, (a, b, r). And all the parameters that satisfying (x, y) would lie on the surface of an inverted right-angled cone whose apex is at (x, y, 0). In the 3D space, the circle parameters can be identified by the intersection of many conic surfaces that are defined by points on the 2D circle. This process can be divided into two stages. The first stage is fixing radius then find the optimal center of circles in a 2D parameter space. The second stage is to find the optimal radius in a one dimensional parameter space.



Once the circle is obtained, the rest of the image is irrelevant. Therefore whatever is outside the circle is blacked out, and the image is cropped to contain only the clock.



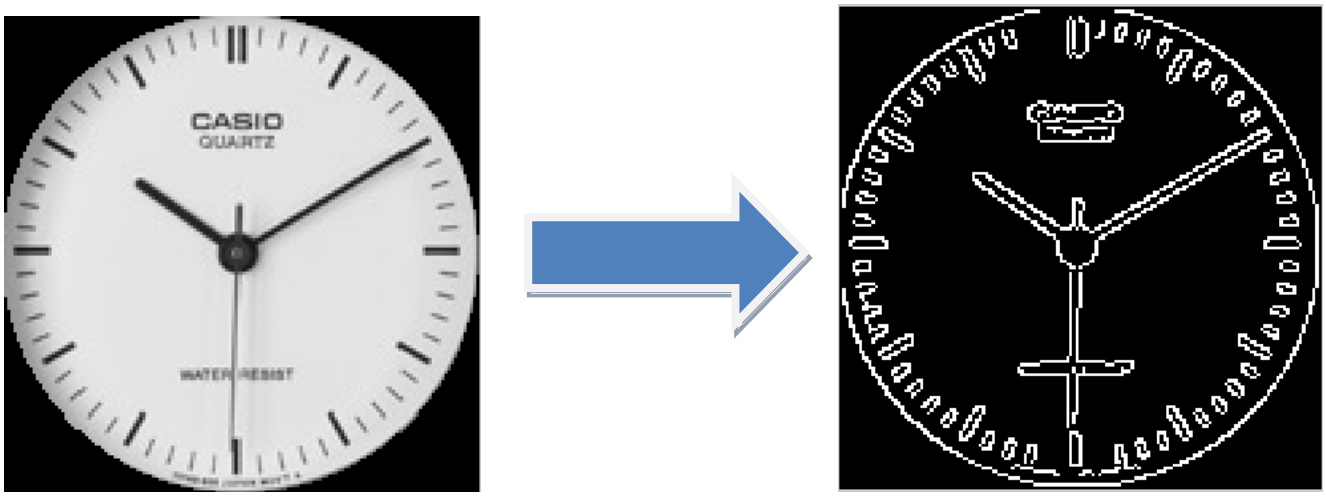


II. Apply edge detection on the image

Once we have our region of interest isolated, we begin the identification process of the clock. In order to do that, we need a binary edge map of the image. We apply the Canny edge detector operator to the image. This operator uses a multi-stage algorithm to detect a wide range of edges in images.

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise.
2. Find the intensity gradients of the image.
3. Apply non-maximum suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges.
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

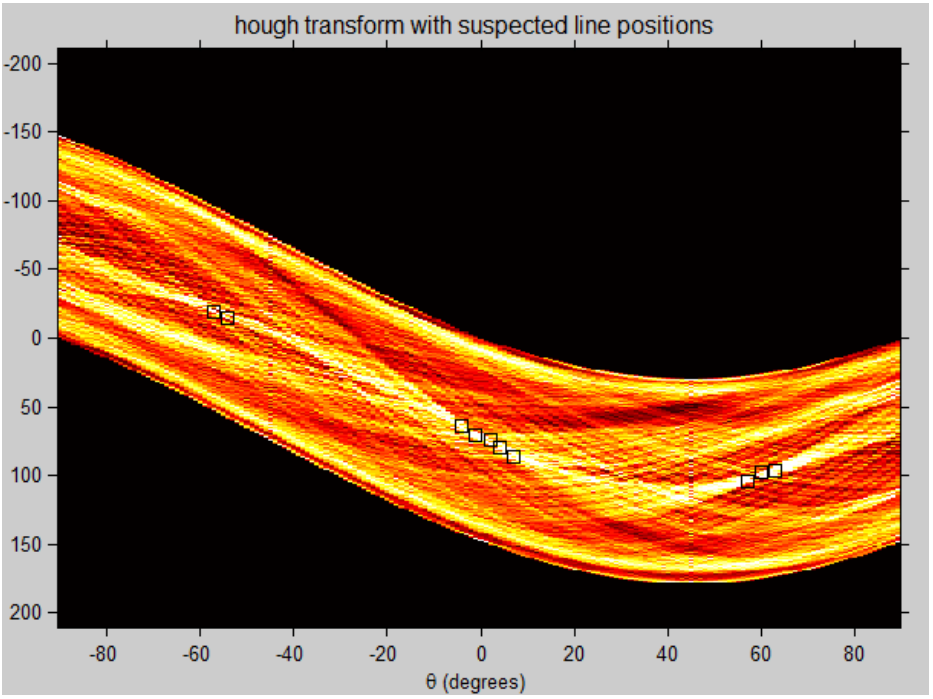


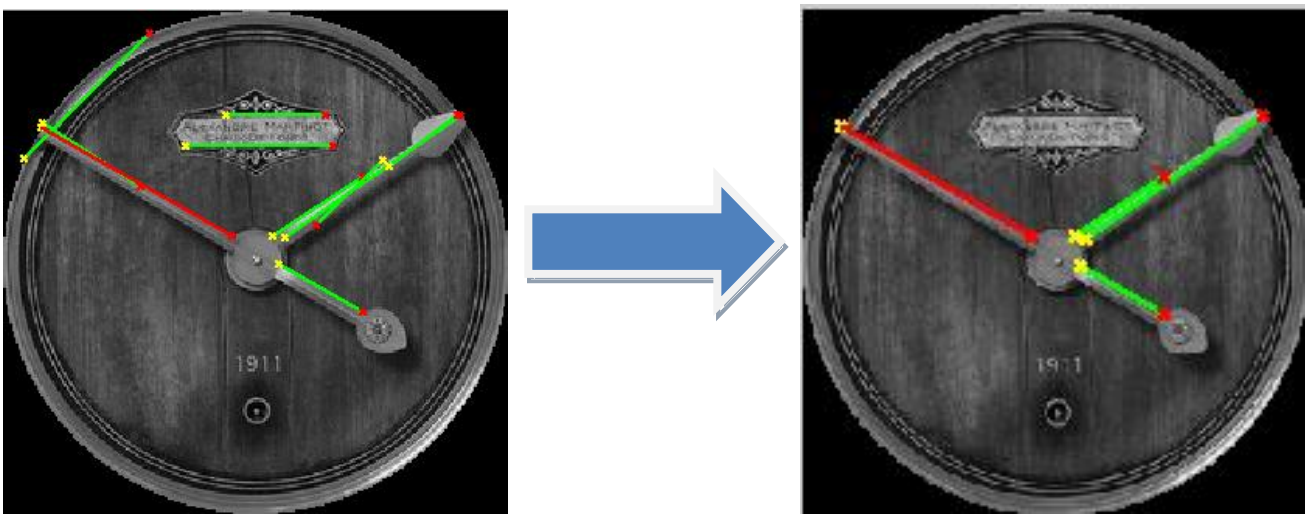
III. Use Hough transform to isolate watch hands

Once we've obtained the edge map, we use an implementation of the Standard Hough Transform designed to detect lines, using its parametric representation

$x * cos(theta) + y * sin(theta)$

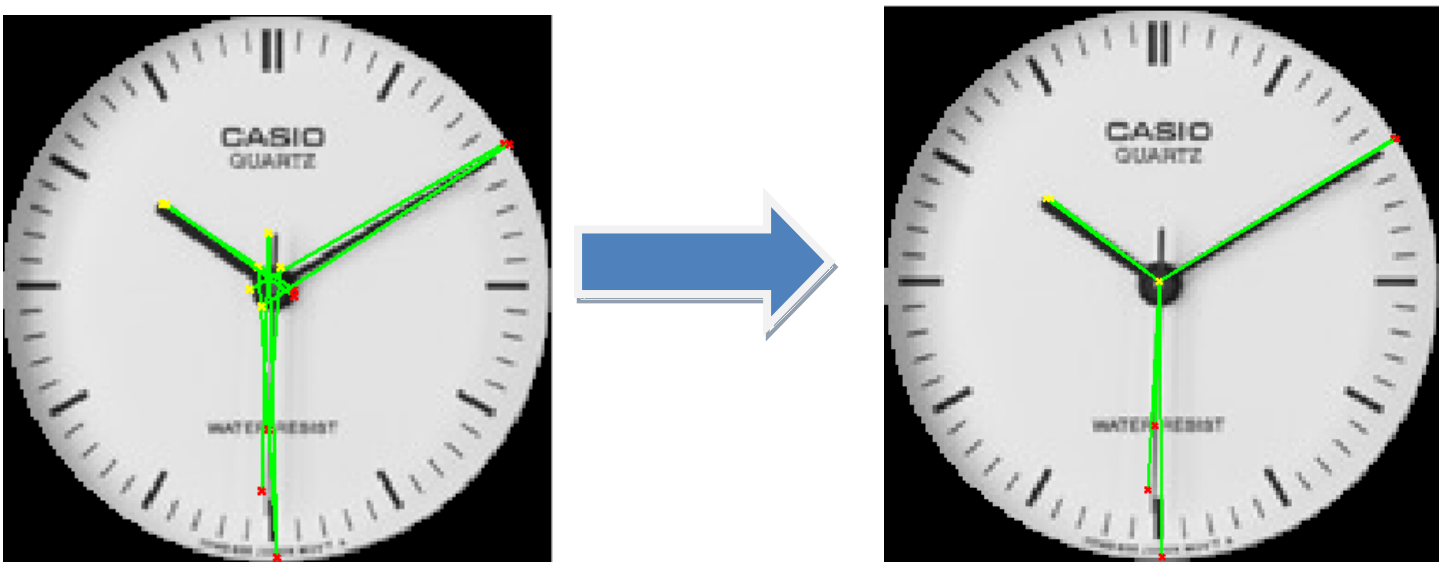
and receive an array of possible candidates. The lines that have no points close to the center of the clock are not watch hands, meaning they are of no interest and are filtered out.





Now we are left with an array of lines close to the center, making for a good list of candidates. To make future calculations easier, all point values are re-assigned relative to the center of the clock, which we will treat from now on as point (0, 0).

The next stage is unifying the lines (clock hand candidates) so that they will all originate from the same point. To do this, we calculate the closer point to the center for each line, and merge it with the clock center.

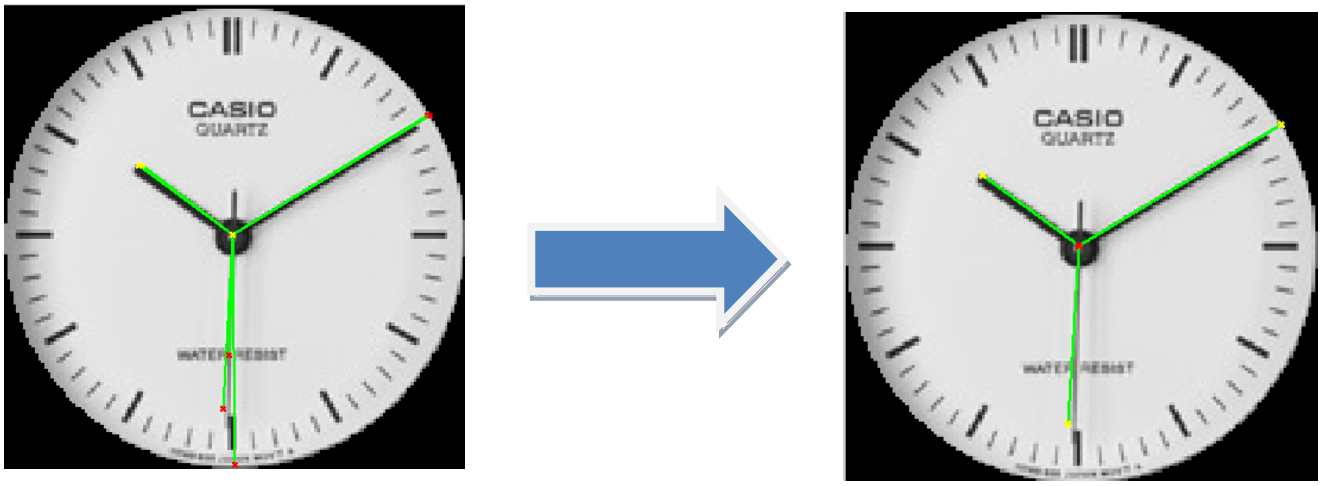


IV. Calculate line angles

Now we have managed to filter down our lines to a list of candidates, aligned to originate from the center of the clock. In order to tell what number each line is pointing at, we will calculate its angle relative to the clock. Since clock images can be received both with and without numbers, there is no possibility of aligning the clock in case it is rotated. Therefore, correct orientation must be assumed, and all calculation will treat north direction as 12 o'clock. In order to align the angles with the correct time, we define that the hours 12, 3, 6, and 9 o'clock are $0°$ $90°$ $180°$ and $270°$, respectively. Using the 'atan2' function on Δx and Δy of every line, we can calculate the angle in radians of the line relative to the x axis, between $\pm[0, \pi]$. Using each line's point's co-ordinates, we can determine which quarter it points to, turn the angle into degrees and correct it relative to our angle system.

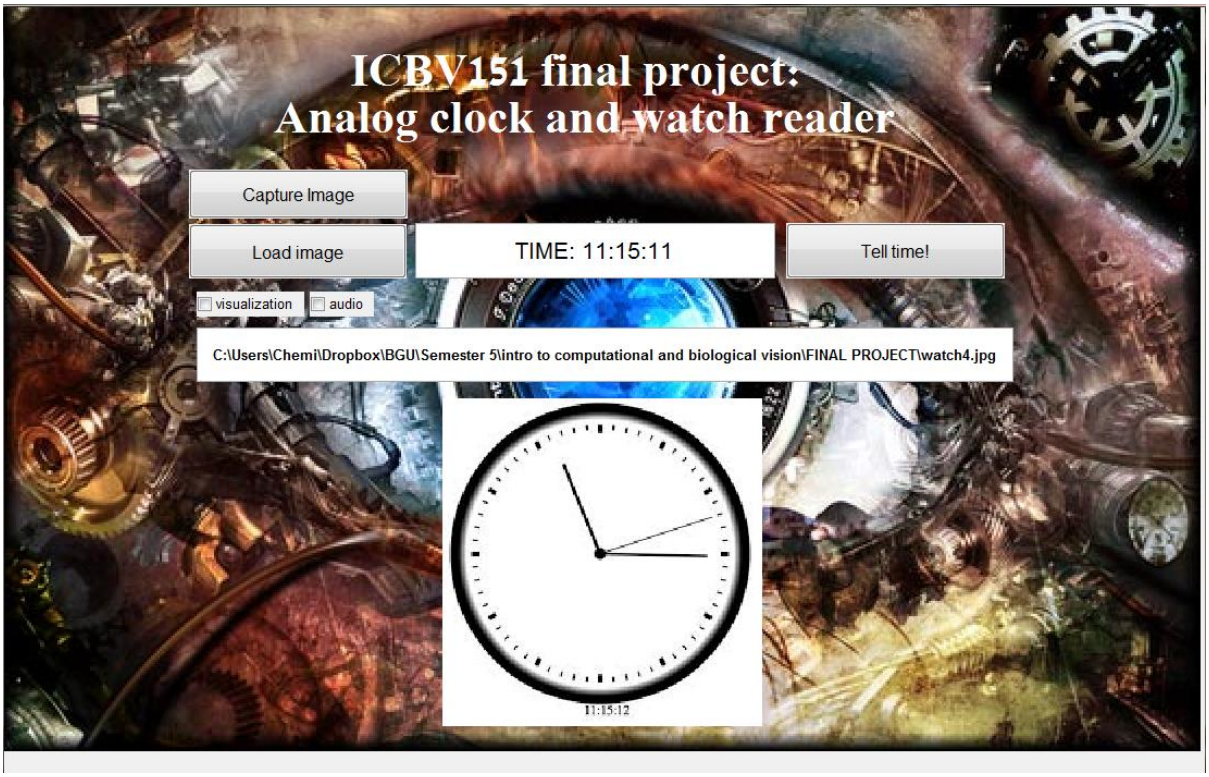
V. Filter out irrelevant lines

If the number of lines is larger than three, sorting the list of angles will enable us to catch suspicious duplicate lines and eliminate them. Comparing every two line angles, we traverse the list of lines and merge any two lines with less than a 5° difference between them, until we are left with three lines. In case only two lines are identified, since not all watches have a seconds hand, we treat them as hour and minute hands.



VI. Compute the time

Down to three lines and line angles, all that's left to do now is identify which line belongs to which hand, and calculate its value accordingly. Assuming the lines are accurately drawn, usually the shortest line will be the hour hand, middle line is the minute hand, and the longest line is the second hand. We can easily calculate the hour, minute, and second based on the angles, which are then displayed and read out loud.



Results

Although the approach to the problem was clear to me, I discovered that there were many tricky issues that needed to be overcome while working on the system. Including partial successes, after testing the system on around 25 images, I can estimate that my system has a success rate of 75-80%.

Tricky issues can occur from a variety of reasons, including:

- Bad lighting
- Non-uniform clock background
- Cluttered clock area (smaller circles and logos)
- Clock hands of the same length

Since I was unable to deal with such a broad range of images, I had to reduce the spectrum of images I could accept by making assumptions of 'valid' input, as I described in the introduction.

Regarding the last issue - since the differences in clock hand lengths are sometimes extremely small, incorrect labeling can occur due to similar hand length and incorrect identification of actual line length by the Hough line transform, resulting in a line that's too long or too short. While my program works correctly on these images for most stages, during the last stage these differences result in some hands detected as others and a different time is displayed.

There are a few approaches that could be taken to deal with the last problem. One of them was to fine-tune the actual line length by traversing each one and summing the pixel values along it. Doing so, one would expect that if the system couldn't decide between two lines, this method would "tip the scales" in one direction. However, when a clock hand is identified as too **long,** occasionally it will merge with the outer perimeter of the clock. In this case, summing pixel values would still result in an incorrect result, since irrelevant black pixel values from the outer clock perimeter would be summed as well.

Another option was to check line *width*. In most clocks, the seconds and minutes hands are very close in length, yet have different widths. Taking into account that the line identification may have overshot, we can calculate the middle pixel of each line (far away from bad pixels), create a local matrix of pixels, and sum their values. Assuming, in most cases, that the seconds hand is thinner than the minutes hand, we would be able to determine which is which. This is the approach I took, and it has managed to eliminate a number of incorrect decisions by the program.

Conclusions

My conclusions from this program, is that in computer vision, there is no easy solution for anything. Each problem must have a specifically-designed solution based on trial and error, using different approaches, and combining different methods.

In my program, I attempted to apply a multi-stage solution for analog clock identification. My goal was to correctly define all stages of identification, and reach a working implementation which has an acceptable success rate and can be applied to many image variations.

I feel that I managed to correctly identify the stages needed to solve the problem. Regarding the implementation, my program has had identical results with a wide variety of pictures, regardless of clock background, style, color and shape.

My success rate overall is acceptable. Most failures occur during the final stage, and I have yet to find a foolproof way of classifying clock hands. I feel that I have learned a lot and gained valuable experience during this project, and have managed to implement some of the computer vision theory studied in class.

Looking forward at future improving the project, I think that the following steps can greatly increase the success rate:

- Improve the line detection algorithm to detect lines *accurately*, under various backgrounds, and correctly classify them
- Improve the circle detection algorithm to work on non-uniform background and under different lightings by pre-processing the image to cancel them out, and possibly integrating a classifier using a training set of clock images.
- Further improve the GUI to allow webcam capture of images directly from the program (requires the image acquisition toolbox. The GUI currently supports Lenovo webcam capture software)
- Explore other alternative or additional methods of time recognition

References

[1] http://en.wikipedia.org/wiki/Hough_transform

[2] http://www.mathworks.com/help/images/examples/detect-and-measure- circular-objects-in-an-image.html

[3] http://homepages.inf.ed.ac.uk/rbf/HIPR2/linedet.htm

[4] http://en.wikipedia.org/wiki/Canny_edge_detector

[5] http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html