<u>Project report- Time Reader</u>

By: Alon Arbel and Gev Keren

This project implements an analog clock reader using computational vision techniques.

This app receives an image of an analog clock and return the time in hh/mm/ss format.

<u>Goal:</u> Extracting time from a various type of analog clocks.

To achieve this goal, we followed these steps:

1. **<u>Image processing:</u>** Reading an image and converting it to a grayscale representation.
2. **<u>Clock detection:</u>** Using HoughCircle method we detected the frame of the clock. HoughCircle uses canny's edge detection algorithm in its process.
3. **<u>Cropping the image:</u>** Using HoughCircle we found the center point and the radius of the clock and we cropped the image to a 2*radius X 2*radius sized image such that the center point of the circle is the center of the new image. (croppedImage = image[(x – r):(x + r) , (y – r):(y + r)])
4. **<u>Resizing the image:</u>** In order to create a uniform threshold, we resized the image to a uniform size (250X250).
5. **<u>Hands detection:</u>** Using cv2.threshold we converted the image to a binary image in order to find the contours using cv2.findContours. We screened those contours by area (which was possible thanks to the uniform size of the image) and extracted the hands contours.
   Using cv2.HoughLinesP we detected the hands lines in the image and drew them using cv2.line on the image.
6. **<u>Hands filter and classification</u>**: Each image produced several lines.
   - We centralized those lines to the center of the clock (the center of the image).
   - We filtered the lines by angle and size to avoid duplications.
   - Each image remained with 1, 2 or 3 lines (depends if the clock has seconds hand and If the hands are above one another).
   - Based on the number of lines we classified which line represents which hand:
     - Case 3: longest hand is the seconds hand, the second longest is the minutes hand and the last is the hours.
     - Case 2: longest hand is the minutes hand and the last is the hours hand.
     - Case 1: Both hands are pointing to the same direction, and therefore this line represents both the hours and minutes hands. (We assumed in this case that there is no second’s hand).

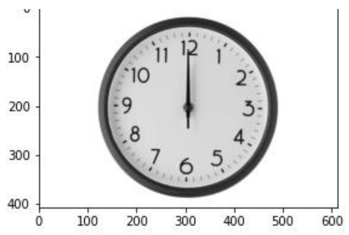
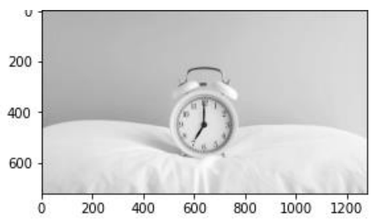
7. **Calculate the time:** Using the hands we found we calculated the angles of the hands.
   - Hour's hand – Detecting the interval in which the hand points and converting it to the right hour.
   - Minutes hand – Calculating the angle of the hand and converting it to the exact number of minutes.
   - Seconds hand – Same as minutes hand.

Examples:

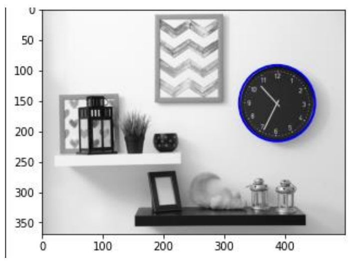
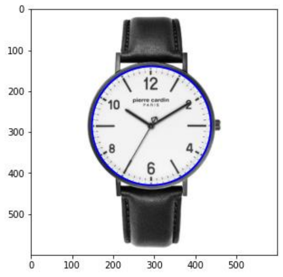
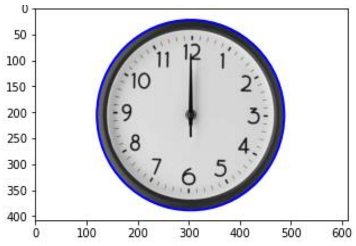
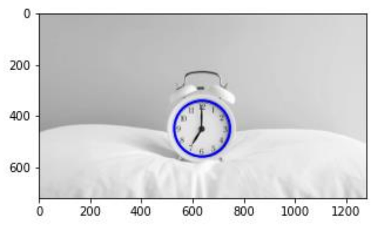
Here are some examples of a few diverse types of watches:

1. **The images:**

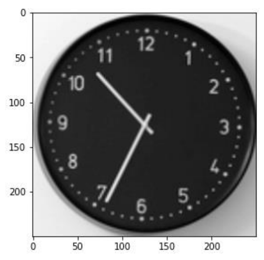
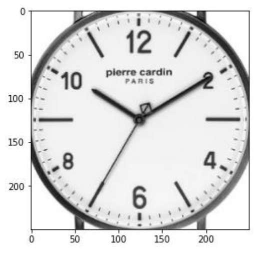
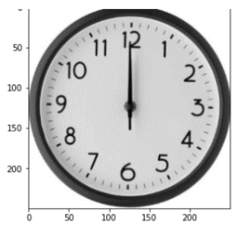
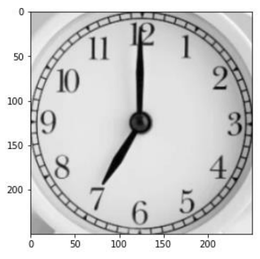
2. **Image processing**:

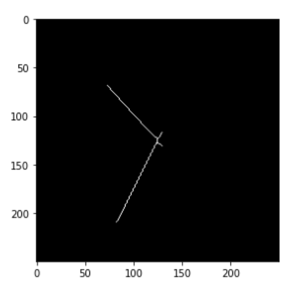
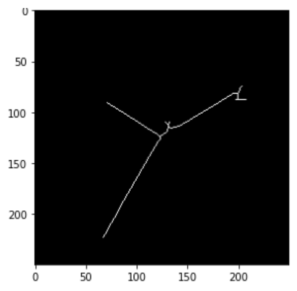
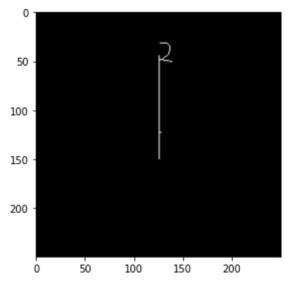
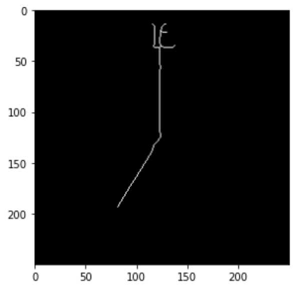
3. **Clock detection**:

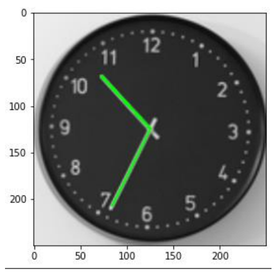
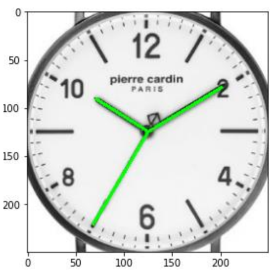
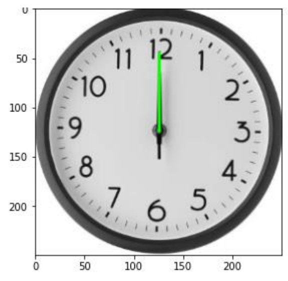
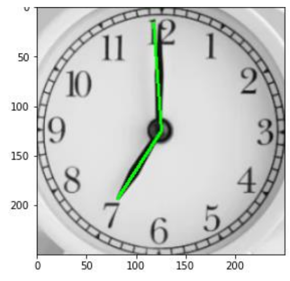
4. **<u>Cropping and resizing the image:</u>**

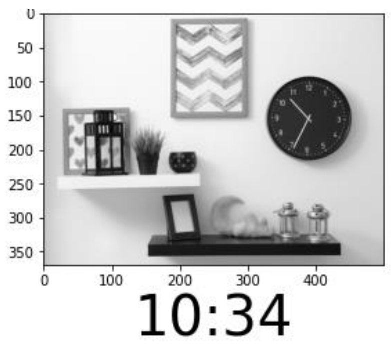
5. **<u>Hands detection:</u>**

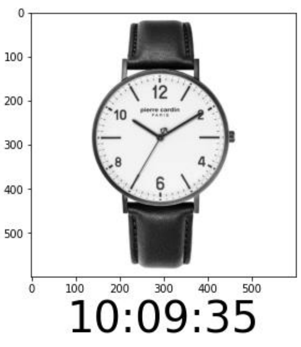





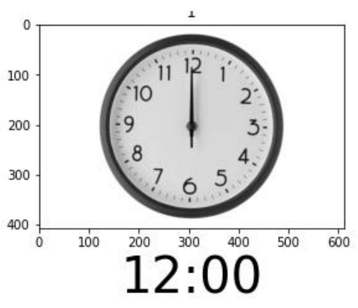
6. **<u>Hands filter and classifying</u>**:

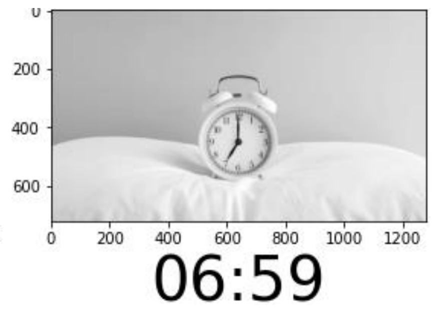





7. **<u>Calculate the time (and show it on the grayscale image):</u>**



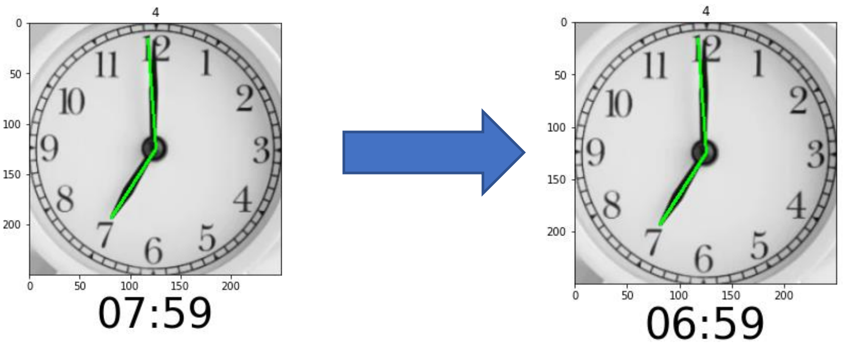






Difficulties and discussions:

1. Since the HoughLinesP might detect a line with a little deviation we might get the wrong time with a deviation of up to 1 hour if considering only the hours hand to calculate the hours.
   To fix it, we added a specification which also considers the minutes hand for cases when the hours hand points exactly on the border of a specific hour interval.
   Therefore, when the minutes hands point close to the '12' we can now calculate the hours hand properly.



2. At first, we thought to calculate the time using only the hours hand. This is theoretically possible because the angle of the hours hand is supposed to represent the exact time and not just the hour. It would have also been easier and faster to calculate.
   We decided on a different approach, which computes all the hands. We did it due to the deviation that might occur in the HoughLinesP method.
   If using the first approach, a 1° deviation in the hours hand causes a 2 minutes error. If using the second process, a 1° deviation in the minutes hand causes only 0.1667 minutes error.
   We have decided to pursue the second approach in order to achieve more accurate results.
3. Classifying the hands contours was not easy. Each image has different size and therefore the hands area on different clocks might be totally different. In order to use threshold classifying we had to resize all of the images to a uniform size for which our threshold will work.

Methods and Functions:

Our methods:

1. **plotCircles** – Plotting the circle we found on the photo.
2. **dist** – Calculates the distance between two points. Used as a part of the line centering.
3. **centerize_lines** – Center the line. Calculates which points of the two ends of the lines are closer to the center and then changes its location to the center. That way we achieve a more accurate line and angle.
4. **calculate_angle** – Using math.atan function, we calculates the angle of the line using the points that represents the line. Returns the angle in a 0-359 degrees format when 0° is "north".
5. **sort_lines** – Sorts the lines list by length using the dist function we wrote.
6. **filter_similar_lines** – Filters lines based on the angle.
7. **screen_lines** – Screens all but the first two lines of a list.
8. **calculate_time** – Calculates the time based on the angles and specification of the hands.
9. **specify_and_calculate** – Specify the case of the time calculations. Whether there are 3 hands or 2 hands, or whether the hands are on top of one another. Then calls the calculate_time function with the right arguments.

OpenCV:

1. **HoughCircles** - Finds circles in a grayscale image using the Hough transform.
2. **Threshold** - Applies a fixed-level threshold to each array element.
   Returns the threshold and a binary image.
3. **findContours** - Finds contours in a binary image.
   Used for shape analysis and object detection and recognition.
   Returns the contours array and its hierarchy.
4. **contourArea** - Calculates a contour area.
   Used to screen objects by size and therefore to detect which object is the hands.
5. **drawContours** - Draws contours outlines or filled contours.
6. **HoughLinesP** - Finds line segments in a binary image using the probabilistic Hough transform.

Results and limitations:

Our app succeeded to read diverse types of analog clocks:

1. Wall, hand and pocket clocks.
2. Clocks with 2 or 3 hands.
3. Clocks with one hand above another.
4. Clocks that are in the background of the image with other objects in it as well.
5. Different sizes of clocks.
6. Different sizes of images.
7. Clocks with or without numbers.

Limitations of the app:

1. Hand's format: In order to specify the hands, we must assume that the hours hand is the shorter than the minutes hand, and the seconds hand (if exists) is the longest.
2. The clock's shape is a circle.
3. The clock's hands are line shaped.
4. We assume that the clock's "12" is pointing north (up).
5. Not all angles of tilt will be able to be calculated.
6. Unclear image: Bad lighting, reflection, etc.