Computer Vision 1: Assignment 3
(Due date: 12.12.2022)

Task 1: Circle detection with Hough transform (programming)

Figure 1 shows a selection of the last Finnish pre-euro coins. The diameters of the coins measured in millimetres are listed in Table 1. We use the Hough transform to detect the coins.

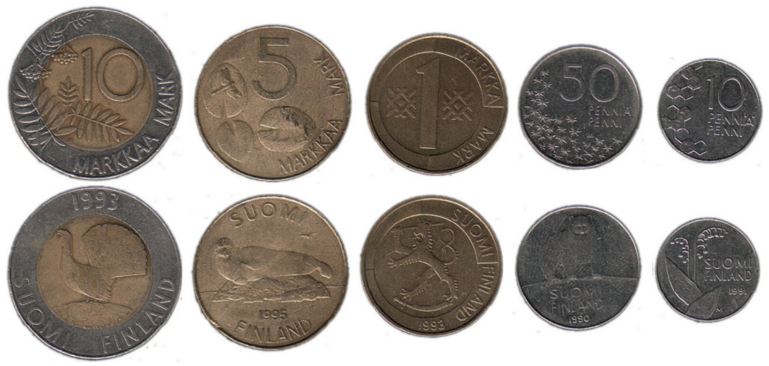

Figure 1: A selection of Finnish coins. From left to right: 10, 5, and 1 marks; and 50 and 10 pennis.

Table 1: Coin diameters specified by the mint.

Coin	10 marks	5 marks	1 mark	50 penni	10 penni
Diameter (millimetre)	27.25	24.50	22.25	19.70	16.30

- Download the image `coins.jpg` from Moodle. Read it and convert to grayscale.
- Calculate the radius r of each coin measured in pixels, by using the data in Table 1 and the fact that the resolution of the image is known to be approximately 0.12 mm/pixel.
- Apply the Canny edge detector to find edges in the grayscale image. Use the built-in function `skimage.feature.canny`. Visualize the edges and check that the outlines of the coins are detected.
- Use `skimage.transform.hough_circle` to calculate the Hough transform of the edge detection result. Use the list of radii you calculated above as the input `radius`. Use default values for the other parameters. Draw the result for each radius. For example, for the radius correspoding to the 5 marks coin you should obtain a result similar to Figure 2 that peaks strongly around the center of the 5 mark coin.
- Use `skimage.transform.hough_circle_peaks` to extract peaks from the Hough transform. Select two peaks per Hough space (per radius), and a total of 10 peaks (that is, 2 for each coin). Set `normalize` to `True` to not give preference to larger circles – see the function documentation for details. Get all outputs from the function, i.e., your call should look like: `accums, cx, cy, radii = hough_circle_peaks(...)`.
- Apply `matplotlib.patches.Circle` to superimpose the 10 circles extracted on the original image. See `https://matplotlib.org/api/_as_gen/matplotlib.patches.Circle.html` for more help. Use `from matplotlib.patches import Circle` to import the circle tool to your code. The result should look similar to Figure 3.

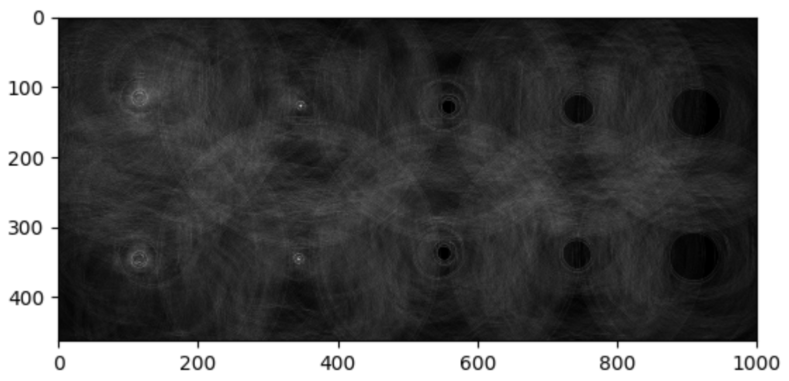

Figure 2: Hough transform for the radius of the 5 mark coin (compare to Figure 1).

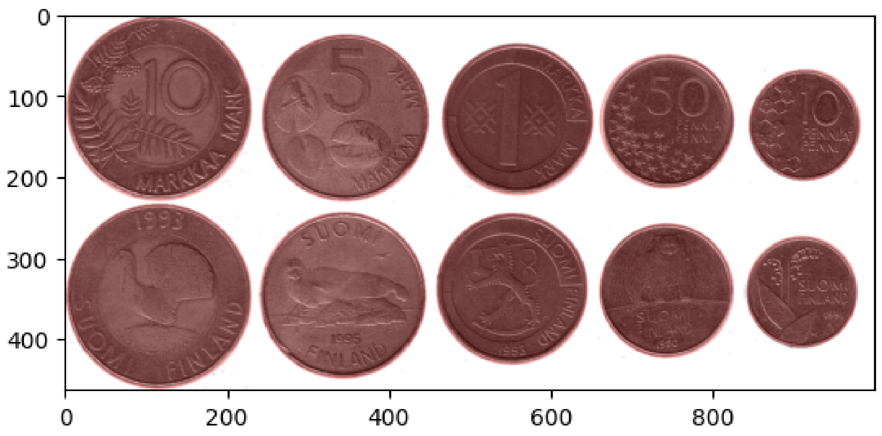

Figure 3: Coin detections shown with colored transparent circles.

Task 2: Line fitting with RANSAC (programming)

Unmanned surface vehicles can detect the direction of the horizon for navigation and localization purposes. Here, we apply RANSAC to detect the horizon line in an image. We will detect the minima of the horizontal gradient along vertical slices of the input image – these most likely correspond to the constant horizon line – and use RANSAC to fit a line to these points. See Figure 4 for samples of intermediate results.

- Download the image `yellow_horizon.jpg` from Moodle. Read it and convert to grayscale. Plot the result.
- Apply a horizontal Sobel filter to detect the horizontal edges in the image. Use the function `sobel_h` from `skimage`. Remember to smooth the image first to avoid false detection due to noise. Plot the result.
- Along 20 equidistant vertical lines across the image, find the coordinates of the minimum

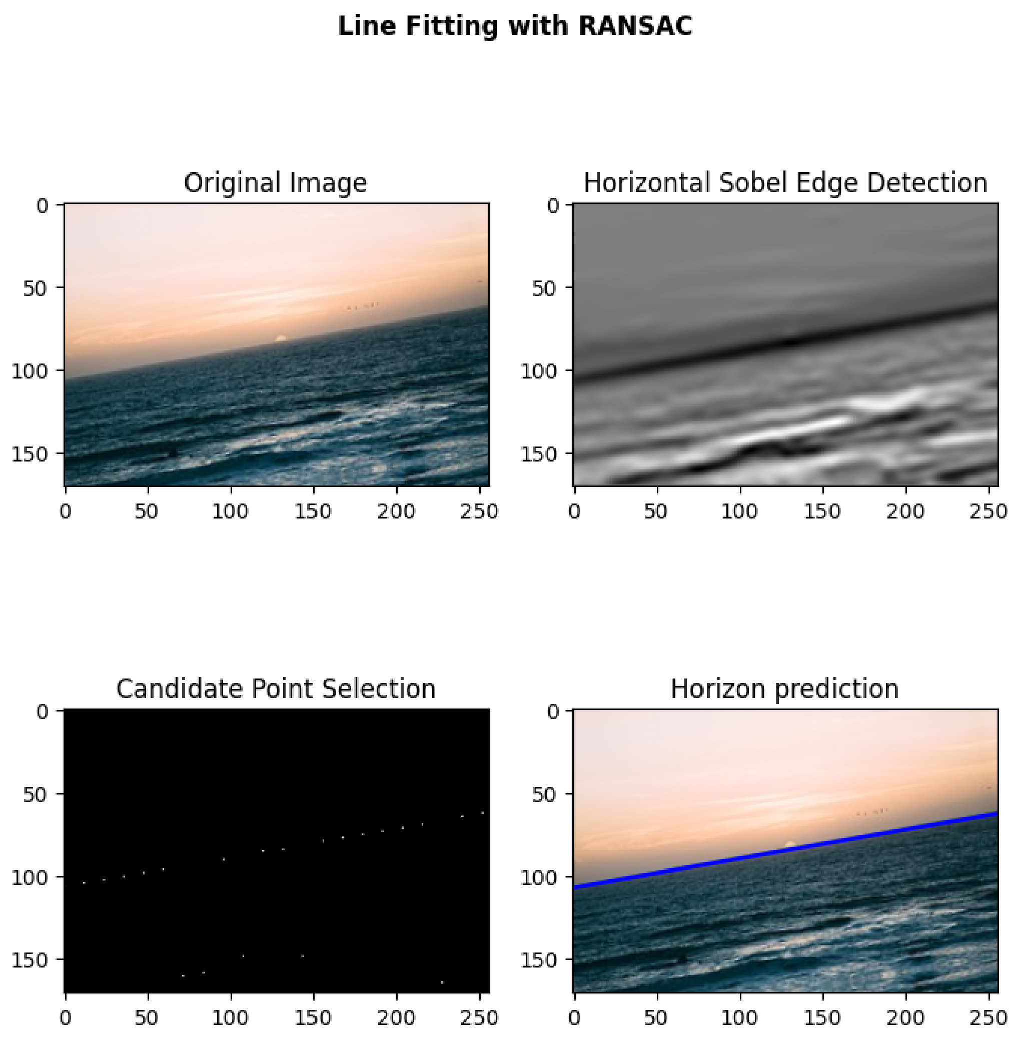

Figure 4: Fitting a line to detect the horizon on a rotated image. Note that the candidate points (bottom left) are found using the minimum of the gradient.

of the horizontal gradient. The (x, y) coordinates found will be the dataset of candidate points used for RANSAC.

- Find the best fitting line using RANSAC. You can either use the built-in function from scikit-learn: `https://scikit-model.RANSACRegressor.html`, or, implement your own version of RANSAC. Tune the parameters to find the line reliably. Verify that the horizon is correctly detected by plotting the line you found superimposed on the original image.
- Rotate the original image by a random amount clockwise or counter-clockwise. Use `skimage.transform.rotate` with `mode="reflect"`. Does your horizon detection method still work? Plot the results.

Task 3: RANSAC properties (pen & paper)

The fraction of inliers in the data is ϵ, and m points are required to define a single model hypothesis. Prove that

$$k \geq \frac{\log(1-p)}{\log(1-\epsilon^m)}$$

model hypothesis iterations are required for RANSAC to succeed with probability at least p (success: at least one model hypothesis has only inlier points).