METR4202: Advanced Robotics and Control

Tutorial 5: Image Formation and Sensing

Many modern robotic systems use computer vision as a way of sensing the robot environment. This tutorial will introduce you to image manipulation and processing using Matlab. This tutorial references a number of new Matlab commands. Don't forget to use `help` or `doc` if you get confused!

1 Reading

Review Matlab's documentation on the image processing toolbox and check the Matlab documentation for any commands in this tutorial that you do not recognize.

Read sections 10.2 (Color) and 13.2 (Line Features) from Corke.

2 Loading and Displaying Images

Download 'chess_circles.jpg' to your computer and in Matlab, change to the directory where you saved it. Load the image using `imread`, and inspect it using `imshow`. Matlab stores images as matrices; you can use the `disp` command to show the raw matrix values.

The JPG image files store 'metadata' about how the photo was captured. Using your operating system's built in tools (Windows Explorer or Finder in Mac), try and discover this information. What camera took this photo? What was the focal length and F-number? What was the exposure time?

3 Detecting Lines

We are going to try and detect where straight lines are in this image using the Hough Transform.

Convert the image to grayscale using `rgb2gray`. The Hough transform operates on black and white images - before we can use it to detect lines, we need to highlight the edges in the image. The Canny edge detector is one way to do this.

Use `icanny` to find edges in the image (or for installations without Peter Corke's Robotics and Vision toolboxes – you can use `edge([image name],'canny')`). You may need to convert your matrix to a `double` or `single` data type before using `icanny`. Your image should now look something like this:

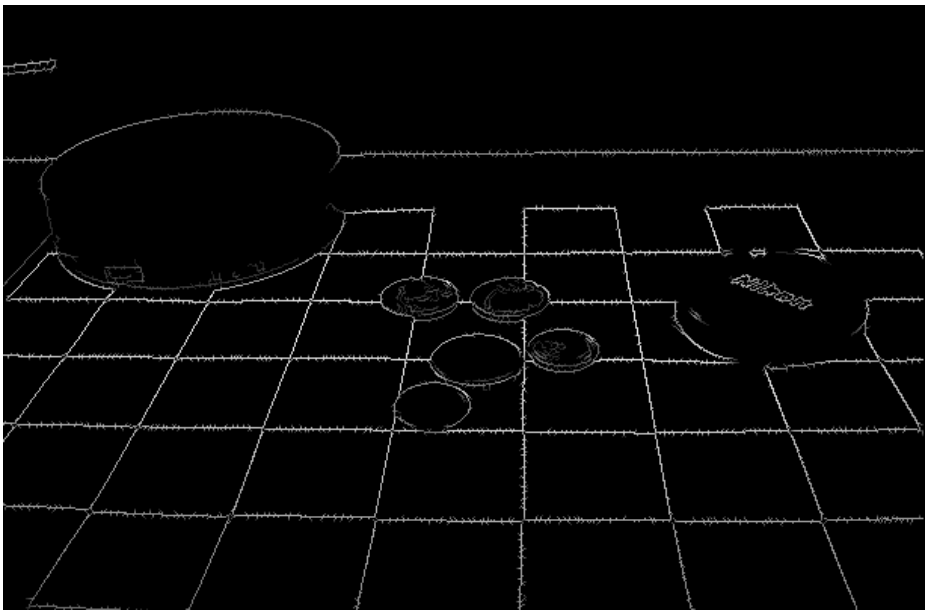

Figure 1: chess_circles.jpg after Canny edge detection

Use the `Hough` class (note the uppercase 'H') to apply the Hough line transform to your edge image. Use the `.show()` command to view the Hough Accumulator for this image. How many lines did the transform find? Plot the lines over the original image. Try adjsuting the smoothing parameter 'sd' for `icanny`. How does this affect the performance of the Hough transform?

4 Detecting Circles

The Hough transform can be used to detect arbitrary geometric shapes. We are now going to try and detect circles in the image.

Download 'houghcircles.m' from the Matlab File Exchange and save it to your working directory.

Use `houghcircles` to search for circles in the image. Note that `houghcircles` takes a grayscale image, not an edge image (it performs edge detection internally). If you call `houghcircles` without specifying the output argument, the image will be shown, and the circles overlayed for you. You can then retrieve the result using the built in 'ans' value in Matlab.

The circular Hough transform struggles with this image due to the perspective transform.

Because you know the lines in the checkerboard pattern are parallel, it is possible to correct the transformation of the image so that the checkerboard squares are actually square (and thus the circles will actually be circular) You may use `checkerboard`, `cpselect`, `cp2tfrom` and `imtransform` for this.

Re-run the Hough circle detector on the transformed image. What are the x,y locations of the coins in your new image?

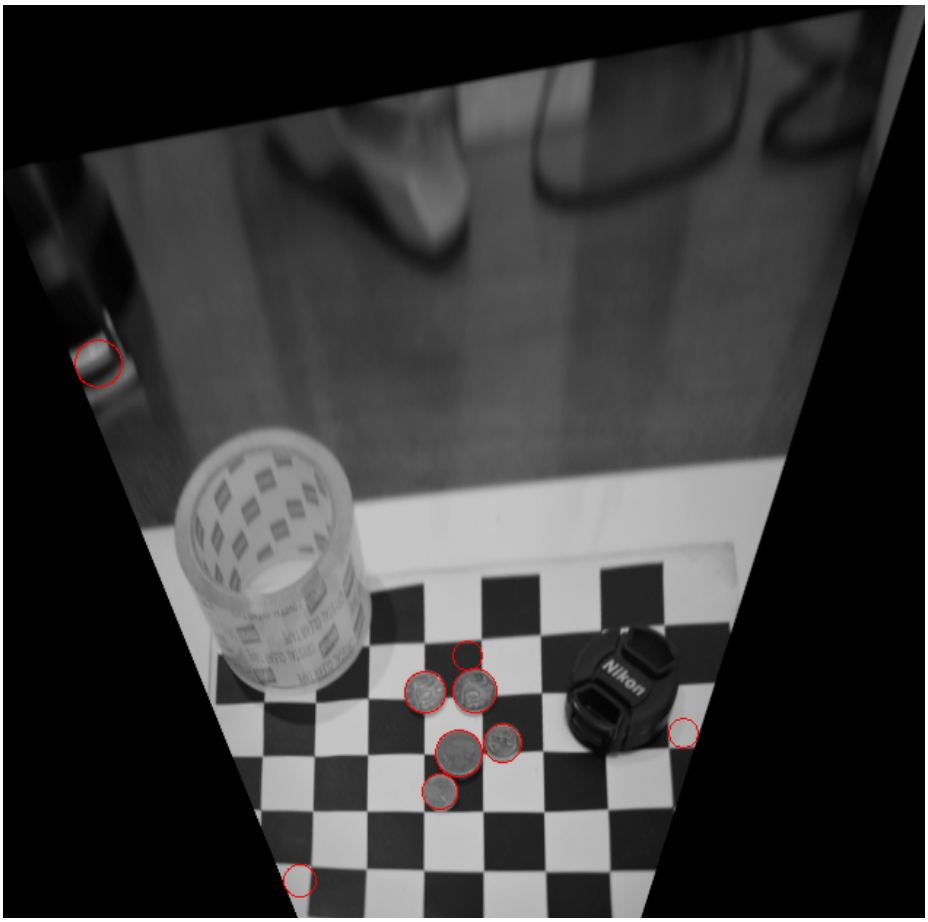

Figure 2: The Hough transform results after image rectification

5 Extra Credit: JPEG Compression

The JPEG compression algorithm is a lossy image encoder. JPEG pictures exploit the fact that the human eye is less sensitive to hue (color) and saturation (color intensity) variations than it is to brightness variations. This can be seen by comparing the hue, saturation and value channels of a highly compressed JPEG image. To try this, save 'chess_circles.jpg' to a new JPEG image using a very high compression setting (hint: look at `doc imsave`). Load this image back in, and convert it to the HSV color model using `rgb2hsv`. Plot the H, S and V channels side-by-side. What do you notice?

What are the benefits of processing images in HSV vs RGB?

One place observe the "lossy compression" in at the edges. To investigate this, compute the edges of both the original and highly compressed versions of `'chess_circles.jpg'`, for example via the `edge(image,'canny')` command.