METR4202 – Robotics & Automation

Lab/Prac 5: Image Manipulation & Basic Processing

(by: .A Snoswell & S.Singh)

Reading
Please read/review Lecture 6: Robot Sensing: Multiple View Geometry.
(§10.2 (Color) and §13.2 (Line Features) in RVC)

This tutorial will introduce the basics of image manipulation and processing using Matlab. We will cover a number of new Matlab commands. Don't forget to use `help`, `doc` and `lookfor` if you get stuck!

0 Loading Images

Matlab includes a number of built-in images for demonstration purposes, such as the Cameraman (left). We can find the location of this and other built in images with the command `which('cameraman.tif')`.

Let's take a quick look at some of these. To view the catalog, open Explorer (or Finder) and go the folder given from this command; that is something like `%MATLABROOT%\toolbox\images\imdata` (e.g. `C:\Program Files\MATLAB 2017\toolbox\images\imdata`).

Now back to Matlab, let's try to load `cameraman.tif`.

```
I_cm=imread('cameraman.tif');
```

� **Thought Questions** �
- How many channels does it have?
- What are the shape, datatype and max/min values of this image data structure?

1 Edging towards the point

Edges in an image are an important building block for many applications. Intuitively, this makes sense, as edges may correspond to (potentially) important real-world properties; such as:
- Changes in depth
- Changes in the orientation of a surface
- Changes in material properties
- Variations in illumination

Sometimes, these real-world characteristics will be relevant to a robotic task. Other times, they may distract, occlude or interfere with the goal of the robot. For this reason robust and reliable detection of edges is an important element of many computer vision systems. It is also difficult to do well!

2 Image gradients

Key commands: `imread`, `imshow`, `im2double`, `conv2`

Note that in Matlab (and many other programming languages), images are indexed with the vertical (***y-axis***) first, and this axis is “inverted.” If we consider the image as a 2D signal, then the command `I(1:10, 1:10)` will show the a 10x10 patch of the signal function in the bottom-left of the image.

Sometimes, not always, it is convenient to process images as double precision floating point in Matlab.

```
I = im2double(I);
```

With our image in double format, we can use simple matrix algebra to take the horizontal difference of the image.

```
diff = I(:, 2:end) - I(:, 1:end-1);
```

Or, to balance the derivative, we could use the first central difference. This is known as the numerical gradient of the signal.

```
diff = -0.5*I(:, 1:end-2) + 0*I(:, 2:end-1) + 0.5*I(:, 3:end);
```

We can do the same thing with a convolution and a small filter. This has the benefit that convolution is a simple element wise multiplication in the frequency domain – this means we can process large images (or larger filter kernels) efficiently.

```
diff = conv2(I, [0.5, 0, -0.5], 'same');
```

� Thought Questions �

- Experiment with different values in the filter kernel.
- Can you find a filter kernel that smooths the image?

3 Sobel Operator

We can combine convolution kernels to create more complex processing operations. The below kernel is known as the Sobel kernel for horizontal image gradients, named after the American scientist Irwin Sobel (1940 -) who first proposed this kernel in 1968 [1].

```
imshow(conv2(I, ([1;2;1]/4) * ([1 0 -1]/2), 'same'));
```

� Thought Questions �

- What would the Sobel kernel be for vertical gradients?
- How would you combine the horizontal and vertical edge images?
- How might you find the orientation of a detected edge?

4 Canny Edge Detection

Often, sensing conditions will lead to noise in the image.

```
I = I + (rand(256, 256) - 0.5) * 0.5;
```

If the image is noisy, we will see this reflected in the gradient **because a gradient amplifies noise**!

These ‘fake edges’ can interfere. As such we would like a more sophisticated way to detect edges. The Canny edge detection algorithm is one approach. It is named after John Canny, an Australian scientist who developed it in in 1986 [2]. It works as follows:

1. Smooth the image using a Gaussian filter kernel
2. Find horizontal, vertical and diagonal gradients using differencing kernels
3. Supress all but the local maxima around each edge – this counters the blurring that took place in step 1.
4. Categorize remaining pixels as ‘not edge’, ‘weak edge’, ‘strong edge’ using two empirical thresholds
5. Suppress weak edges that are not connected to strong edges

Thankfully, we don’t have to implement all of this from first principles. We can use the Canny edge detector code built-in to Matlab.

```
edges = edge(I, 'canny');
```

� Thought Questions �

- Look at the documentation for `edge`.
  Try adjusting the threshold and sigma parameters to a nicer result.
- Why does the Canny method use a Gaussian smoothing kernel?
  [**Hint**: See p. 13 of Canny’s original paper]
- Imagine you are a controls engineer recently assigned to a new project. Your goal is to develop a computer vision system to detect defects in products on a factory production line. A colleague suggests using Canny edge features.
  - What are the advantages of this approach compared to other edge detectors?
  - What other external factors would you have to consider when designing the computer vision system?
    [**Hint**: Think about where you can add structure in the hardware, the software AND the environment]

5 References

[1] Sobel, Irwin, and Gary Feldman. "A 3x3 isotropic gradient operator for image processing." *a talk at the Stanford Artificial Project in* (1968): 271-272.

[2] Canny, John. "A computational approach to edge detection." *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986): 679-698. [Cached PDF]

[3] Dollár, Piotr, and C. Lawrence Zitnick. "Structured forests for fast edge detection." *Proceedings of the IEEE International Conference on Computer Vision*. 2013.