From Last Class

Controller operates on the ERROR between reference & output

Proportional
- Reduces disturbance error
- Non-zero steady-state error

Integral
- Ensures zero steady-state error
- Can be destabilizing

Derivative
- Adds artificial damping
- Slows response

Convolution & Filtering

- Image operations as convolution, given $I$, the signal/image,
  \[ f[m, n] = I \otimes g = \sum_{k,l} I[m-k, n-l] g[k, l] \]
- $g[k, l]$ – is the convolution kernel
- Smoothing

Convolution & Filtering

- Sharpening
- Differentiation

Properties of the Convolution

- Convolution is commutative $I \otimes g = g \otimes I$
- Convolutions is associative $(I \otimes g) \otimes f = I \otimes (g \otimes f)$
- Convolution is linear
  \[ I \otimes (g + f) = (I \otimes g) + (I \otimes f) \]
  \[ \alpha(I \otimes g) = (\alpha I) \otimes g = I \otimes (\alpha g) \]
**Edge Detection**

- An *Edge* is where change occurs
- Change in 1D – measured by derivative
- Max change corresponds to where derivative has max magnitude

**Computing Gradients: 1st Order Derivatives**

1. Take the image intensity difference in the X-direction
2. Average the difference in the Y-direction (smoothing)

\[
\frac{\delta I}{\delta x}(i,j) = \frac{1}{2} \left( (I(i,j+1) - I(i,j)) + (I(i+1,j+1) - I(i+1,j)) \right)
\]

**Some Matlab Code**

```
[nr,nc] = size(I);
Ix = zeros(nr,nc); % generate an empty matrix of size nr by nc
for i=1:nr-1,
    for j=1:nc-1,
        Ix(i,j) = 0.5 * (I(i+1,j+1) - I(i+1,j)) + (I(i,j+1) - I(i,j));
    end
end
```

**An Example**

\[ I_s = imcrop(I_g); \]
\[ Imagesc(I_s); colormap(gray) \]
Recall

We can compute gradient as:
1. Convolution Operation
2. Filtering Operation

First Order Derivatives

\[
\frac{\delta}{\delta x} \frac{\delta I}{\delta x}(i,j) = (I(i, j + 1) - I(i, j))
\]
Computing Gradients

\[
\frac{\delta I}{\delta x}(i,j) = (I(i,j+1) - I(i,j)); \\
= I \otimes \left( \frac{\delta}{\delta x} \right)
\]

Derivative in X-Direction

\[
\frac{\delta}{\delta x} = \begin{pmatrix} 1 & -1 \end{pmatrix} \\
S = \begin{pmatrix} 1 \\ 1 \end{pmatrix}
\]

Computing Gradients: Y-Direction

\[
\frac{\delta I}{\delta y}(i,j) = \frac{1}{2}(I(i+1,j) - I(i,j)) + (I(i+1,j+1) - I(i,j+1)); \\
I_y = (I \otimes \frac{\delta}{\delta y}) \otimes S'
\]

Matlab's Conv2 Function

\[
s = [1; 1]; \\
dx = [1, -1]; \\
gx = conv2(conv2(I, dx', 'same'), s, 'same'); \\
I_x = (I \otimes \frac{\delta}{\delta x}) \otimes S
\]
**Smoothing before Differentiating**

- Smoothing – Remove noise
- Does the following make a difference:
  - Image -> Smooth -> Differentiate OR
  - Image -> Differentiate -> Smooth
- Answer: NO!

**Furthermore**

We can simplify this even more …

**Recall the Gaussian**

\[
E \left( \frac{x^2 + y^2}{2\sigma^2} \right)
\]
**Smoothed Derivative Filter**

Rewrite:

$$\frac{\delta}{\delta x} \otimes G = \frac{\delta G}{\delta x} \rightarrow \frac{\delta G}{\delta x} = -\frac{2x}{\sigma_x^2} G(x, y)$$

**In Summary**

Filter out noise and compute gradient.

In Matlab:

```matlab
>> [dx,dy] = gradient(G); % G is a 2D gaussian
>> Ix = conv2(I,dx,'same'); Iy = conv2(I,dy,'same');
```

**A Word of Caution**