Color Space 1: RGB Color Space

- RGB Cube
  - Easy for devices
  - But not perceptual
  - Where do the grays live?
  - Where is hue and saturation?

Color Space 2: HSV

- Hue, Saturation, Value (Intensity)
  - RGB cube on its vertex
  - Decouples the 3 components (a bit)
  - How do you convert from one to the other?
    - Wikipedia
    - Use rgb2hsv() and hsv2rgb() in Matlab

RGB ↔ XYZ

Color Similarity in Different Color Spaces

\[
\Delta E_{ab} = \sqrt{(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2(0.5)}
\]

CIE xy

CIE ab
Color Space 4: CIE LAB Space

• CIE XYZ
  \[(x, y) = (X/(X+Y+Z), Y/(X+Y+Z))\]

• CIE LAB
  \[
  \begin{align*}
  L &= 116((Y/Y_n)^{1/3} - 16; \\
  A &= 500((X/X_n)^{1/3} - (Y/Y_n)^{1/3}; \\
  B &= 200((Y/Y_n)^{1/3} - (Z/Z_n)^{1/3})
  \end{align*}
  \]

• Difference in this space is more close to human color space

Color Space 5: YUV

• transform RGB to YCrC, or to YUV using 3x3 matrix
  \[
  \begin{bmatrix}
  Y \\
  Cb \\
  Cr
  \end{bmatrix} = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
  \end{bmatrix}
  \begin{bmatrix}
  R \\
  G \\
  B
  \end{bmatrix}
  \]

• most image enhancement and compression are performed on luminance and chrominance values separately
  - eye is more sensitive to luminance than to chrominance
  - preserve color before and after processing

Color Calibration

• Histogram
• Averaging of region of interest
• Goal: Obtain a range of ||(u,v)|| to represent color

Binary Morphological Operations
  - Dilation, Erosion
  - Opening, Closing

Labeling Connected Components
Visual Servoing

- Navigation based on visual cues
- Visual Cue: Color Blob
- Navigation: Follow the Blob!
  - PURELY reactive

Open-Loop vs. Feedback Control

- Objective: Follow the color blob

Mechanical Speed Control

1788 Fly-ball Governor (James Watt)

Proportional Controller

Let \( e(t) = y(t) - r(t) \),

then, \( P = K_p e(t) \)

Prone to overshooting the reference
### Proportional Controller
- High $K_p$ results in large change in output
- Steady-state error depends on magnitude of $K_p$

### Integral Controller
\[ I = K_I \int_0^t e(\tau) d\tau \]
Note: Integral term makes adjustments based on BOTH magnitude and duration of the error

### Integral Controller
- $K_I$ (w/ $K_p$) speeds up movement towards reference
- Possibility of overshooting – Why?
- Can add instability to the system

### Derivative Controller
\[ D = K_d \frac{de(t)}{dt} \]
Note: Integral term makes adjustments based on BOTH magnitude and duration of the error
Derivative Controller

- $K_d$ slows down movement towards reference
- In conjunction with $K_i$ can limit overshoot caused by $K_i$
- Sensitive to noise in the signal

In Summary

- 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

How this relates to following color blobs?

- What is the input?
- What is the output?

Given an image …
### Convolution

- **Definition**
  - **Continuous**
    \[(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau\]
  - **Discrete**
    \[(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n - m]\]

### Applying Masks to Images

- **Convolution of the Image w/ another “Signal”**
- **Masks have origins**
  - Symmetric masks – origins are the center pixels
Linear Functions

- **Simplest: Linear filters**
  - Key Idea: Replace each pixel by a linear combination of its neighbors
- The prescription for the linear combination is called the convolution kernel

<table>
<thead>
<tr>
<th>Local image data</th>
<th>kernel</th>
<th>Modified image data</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Simple Neural Network

The Retina: Up-Close

- The retina is composed of photoreceptors, bipolar cells, ganglion cells, and nerve fiber layers.
- Light enters through the photoreceptors, which convert light into electrical signals.

Simple Neural Network

- **Input:** $I(x) = [0, 0, 1, 0, 0]$
- **Kernel:** $g(x) = [0.3, 1, 0.6]$
- **Output:** $f(x) = [0, 0.6]$
Linear Filtering: Warm – Up Exercise I

f(x) = [0, 0, 1, 0, 0]

h(x) = [0.3, 0.6, 0.3, 1, 0.6]

g(x) = [0, 0.6, 1]

Linear Filtering: Warm – Up Exercise II

f(x) = [0, 0, 1, 0, 0]

h(x) = [0.3, 1, 0.6, 0.3, 1, 0.6]

g(x) = [0, 0.6, 1, 0.3]

(Following examples taken from B. Freeman)
Linear Filtering: Warm-Up Exercise III

**Blur Examples**

**Linear Filtering: Warm-Up Exercise IV**

**Linear Filtering: Warm-Up Exercise V**
Remember Blurring

![Original image](image1)

![Blurred image](image2)

Coefficient: 0.3

Pixel offset: 0

Blurred (filter applied in both dimensions).

Linear Filtering: Warm-Up Exercise V

![Original image](image3)

![Sharpened image](image4)

Coefficient: 2.0

Coefficient: 0.33

Sharpened

original

original

Sharpening Examples

![Original image](image5)

![Sharpened image](image6)

Coefficient: 8

Coefficient: 1.7

Coefficient: 11.2

Sharpened
differences are accentuated; constant areas are left untouched.

before

after
Convolution

• Let \( I \) be a Signal (image), \( g \) be the convolution kernel

  then the convolution of \( I \) and \( g \) is given by

\[
 f[m,n] = I \ast g = \sum_{k,l} I[m-k,n-l]g[k,l]
\]

Note: Discrete!

• Can think of it as a form of running average.

Note the Following

\[
f[m,n] = I \ast g = \sum_{k,l} I[m-k,n-l]g[k,l] = \sum_{k,l} I[m+k,u+n]g[k,l]
\]

Image Operations as Convolution

• Average Filter

  • Masks w/ positive entries that sum to 1;
  • Replaces each pixel w/ an ave of its neighborhood;
  • If all weights are equal – called BOX filter

F

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

• (Campos)
**Smoothing by Averaging**

- **Gaussian Averaging**
  - Rotationally symmetric
  - Weights nearby pixels more than distant ones
  - This makes sense for probabilistic inference

**Other 2-D Filters**

- **An Isotropic Gaussian**
  - The picture shows a smoothing kernel proportional to
  \[
  \sigma^2 + \gamma^2
  \]
  - Reasonable model for a fuzzy blob

\[\nabla^2 \text{ is the Laplacian operator:} \]

\[
\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}
\]
Smoothing with a Gaussian

The Effects of Smoothing

• Each row shows smoothing with Gaussians of different width;
• Each column shows different realizations of an image Gaussian noise.
Image Smoothing Can Remove Noise

- And also ...

Computing Gradient as a Convolution Operation
Can also compute gradient as a filtering operation!

\[
I(x) = \begin{bmatrix} 1 & 2 & 5 & 5 & 10 \\ 0 & -1 & 1 \end{bmatrix}
\]

\[
f(x) = \begin{bmatrix} 0 & -1 & 1 \end{bmatrix}
\]

\[
g(x) = \begin{bmatrix} 1 & 3 & 0 \end{bmatrix}
\]

Edge is Where Change Occurs

- In 1-D, change is measured by the derivative
- Derivatives can have maxes (Biggest Change) and mins (No Change)

Computing Gradients: 1st Order Derivatives

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

$$\frac{\partial I}{\partial z} (i,j) = \frac{1}{2} ( (I(i, j + 1) - I(i, j)) + (I(i + 1, j + 1) - I(i + 1, j))$$

$$I(i,j)$$

$$I(i+1,j)$$

$$I(i,j+1)$$

$$I(i+1,j+1)$$

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

An Example

```
Iz = imerode(Ig); Imagesc(Iz); colormap(gray)
```
Recall
• We can compute gradient as:
  1. Convolution Operation
  2. Filtering Operation

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

\[
= I \otimes \left( \frac{\delta}{\delta x} \right)
\]

First Order Derivatives
\[
\frac{\delta}{\delta x} = \begin{bmatrix} 1 & -1 \end{bmatrix}
\]

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

\[
S = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}
\]

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

\[
= (I \otimes \frac{\delta}{\delta x}) \otimes S
\]
Computing Gradients: Y-Direction

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

\[
I_y = (I \otimes \frac{\delta}{\delta y}) \otimes S'
\]

Matlab's Conv2 Function

\[
s = [1; 1];
\]
\[
dx = [1; -1];
\]
\[
dy = [1; 1];
\]
\[
gx = conv2(conv2(I, dx', 'same'), dy', 'same');
gy = conv2(conv2(I, dy', 'same'), dx', 'same');
\]

\[
I_y = (I \otimes \frac{\delta}{\delta y}) \otimes S'
\]