CAP 4453
Robot Vision
Dr. Gonzalo Vaca-Castaño
Gonzalo.vacacastano@ucf.edu
Administrative details

• Homework 1 issues?
Questions?
Robot Vision

3. Image Filtering
Credits

• Some slides comes directly from:
  • Yogesh S Rawat (UCF)
  • Noah Snavely (Cornell)
  • Ioannis (Yannis) Gkioulekas (CMU)
  • Mubarak Shah (UCF)
  • S. Seitz
  • James Tompkin
  • Ulas Bagci
Outline

• Image as a function
  • Extracting useful information from Images
    • Histogram
    • Filtering (linear)
    • Smoothing/Removing noise
    • Convolution/Correlation
    • Image Derivatives/Gradient
    • Edges

• Colab Notes/ homeworks
• Read Szeliski, Chapter 3.
• Read/Program CV with Python, Chapter 1.
What is an image?

- We can think of a (grayscale) image as a function, $f$, from $\mathbb{R}^2$ to $\mathbb{R}$:
  - $f(x, y)$ gives the intensity at position $(x, y)$

- A digital image is a discrete (sampled, quantized) version of this function
Image transformations

• As with any function, we can apply operators to an image

\[ g(x, y) = f(x, y) + 20 \]

\[ g(x, y) = f(-x, y) \]

• Today we’ll talk about a special kind of operator, \textit{convolution} (linear filtering)
Filters

• Filtering
  – Form a new image whose pixels are a combination of the original pixels

• Why?
  – To get useful information from images
    • E.g., extract edges or contours (to understand shape)
  – To enhance the image
    • E.g., to remove noise
    • E.g., to sharpen and “enhance image” a la CSI
  – A key operator in Convolutional Neural Networks
Question: Noise reduction

• Given a camera and a still scene, how can you reduce noise?

Take lots of images and average them!

Source: S. Seitz
Question: Noise reduction

- Given a camera and a still scene, how can you reduce noise?

Take lots of images and average them!

Can we something else?

Source: S. Seitz
Thresholding!

\[ g(m, n) = \begin{cases} 
255, & f(m, n) > A \\
0, & \text{otherwise}
\end{cases} \]
Question: Noise reduction

• This is not a gray scale image

```python
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt

folder = 'C:/Users/gonza/OneDrive/Teaching/CAP4453/class3/'
list_dir = [fil for fil in os.listdir(folder) if fil[-3:]=='.JPG']

for iFile, fname in enumerate(list_dir):
    if iFile == 0:
        sumFile = cv2.imread(folder + fname)
        sumFile = sumFile.astype(np.float)
    else:
        sumFile = sumFile + cv2.imread(folder + fname).astype(np.float)

sumFile = sumFile/len(list_dir)
sumFile[sumFile>255] = 255
sumFile[sumFile<=90] = 0

plt.imshow(sumFile.astype(np.uint8))
```

Source: S. Seitz
Image noise

• Light Variations
• Camera Electronics
• Surface Reflectance
• Lens

• Noise is random,
  • it occurs with some probability
  • It has a distribution
Additive Noise

\[ I_{observed}(x, y) = I_{original}(x, y) + n(x, y) \]

True pixel value at \( x, y \)

Noise at \( x, y \)
Multiplicative Noise

\[ I_{\text{observed}} (x, y) = I_{\text{original}} (x, y) \times n(x, y) \]
Gaussian Noise

\[ n(x, y) \approx g(n) = e^{-n^2/2\sigma^2} \]

Probability Distribution

\( n \) is a random variable
Gaussian function

\[ g(x) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left( -\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2} \right). \]
Salt and pepper noise

- Each pixel is randomly made black or white with a uniform probability distribution
Uniform distribution

\[ f(x) = \frac{1}{b-a} \] for \( a \leq x \leq b \]
Noise implementation

```python
import numpy as np
import os
import cv2

def noisy(noise_typ,image):
    if noise_typ == "gauss":
        row,col,chn = image.shape
        mean = 0
        var = 1
        sigma = var**0.5
        gauss = np.random.normal(mean,sigma,(row,col,chn))
        gauss = gauss.reshape(row,col,chn)
        noisy = image + gauss
        return noisy
    elif noise_typ == "speckle":
        row,col,chn = image.shape
        s_vs_p = 0.5
        amount = 0.004
        out = image
        # Salt mode
        num_salt = np.ceil(amount * image.size * s_vs_p)
        coords = [np.random.randint(0, i - 1, int(num_salt)) for i in image.shape]
        out[coords] = 1
        # Pepper mode
        num_pepper = np.ceil(amount * image.size * (1. - s_vs_p))
        coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in image.shape]
        out[coords] = 0
        return out
    elif noise_typ == "poisson":
        vals = len(np.unique(image))
        vals = 2 ** np.round(np.log(vals))
        noisy = np.random.poisson(image * vals) / float(vals)
        return noisy
    elif noise_typ == "speckle":
        row,col,chn = image.shape
        gauss = np.random.randn(row,col,chn)
        gauss = gauss.reshape(row,col,chn)
        noisy = image + image * gauss
        return noisy
```
Outline

- Image as a function
- Extracting useful information from Images
  - Histogram
  - Filtering (linear)
  - Smoothing/Removing noise
  - Convolution/Correlation
  - Image Derivatives/Gradient
  - Edges
- Colab Notes/ homeworks
- Read Szeliski, Chapter 3.
- Read/Program CV with Python, Chapter 1.
Linear shift-invariant image filtering

• Replace each pixel by a \textit{linear} combination of its neighbors (and possibly itself).

• The combination is determined by the filter’s \textit{kernel}.

• The same kernel is \textit{shifted} to all pixel locations so that all pixels use the same linear combination of their neighbors.
Filtering

• Modify pixels based on some function of neighborhood
Image filtering

- Image filtering: compute function of local neighborhood at each position

\[ h[m,n] = \sum_{k,l} f[k,l] I[m+k, n+l] \]

2d coords = k, l  2d coords = m, n

\[
\begin{bmatrix}
  [ ] & [ ]
\end{bmatrix}
\begin{bmatrix}
  [ ]
\end{bmatrix}
\]
Image filtering

- Image filtering: compute function of local neighborhood at each position

- Enhance images
  - Denoise, resize, increase contrast, etc.

- Extract information from images
  - Texture, edges, distinctive points, etc.

- Detect patterns
  - Template matching
Let’s run the box filter

$$h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]$$

note that we assume that the kernel coordinates are centered
Let’s run the box filter

\[ h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l] \]

output \hspace{1cm} filter \hspace{1cm} image (signal)
Let’s run the box filter

![Box filter diagram](image)

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

output \hspace{1cm} filter \hspace{1cm} image (signal)

shift-invariant: as the pixel shifts, so does the kernel
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

output  \hspace{1em} filter  \hspace{1em} image (signal)
Let’s run the box filter

Let's run the box filter with the given kernel:

\[
g[k, l] = \begin{cases} 
  1 & \text{if } (k, l) \text{ is in the center of the kernel}, \\
  0 & \text{otherwise}.
\end{cases}
\]

The output is calculated as:

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

where

- \(g[k, l]\) is the kernel
- \(f[m, n]\) is the input image
- \(h[m, n]\) is the output image
Let’s run the box filter

$$h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]$$

output \hspace{1cm} filter \hspace{1cm} image (signal)
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

output \hspace{1cm} filter \hspace{1cm} image (signal)
Let’s run the box filter

Let's run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

output \hspace{1cm} filter \hspace{1cm} image (signal)
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]
Let’s run the box filter

The box filter uses a simple rectangular kernel. For a $3 \times 3$ kernel, the filter operation is given by:

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

where $g[k, l]$ is the kernel, $f[m, n]$ is the image (signal), and $h[m, n]$ is the output.
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

output \hspace{1cm} filter \hspace{1cm} image (signal)
Let’s run the box filter

$$h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]$$

output \quad filter \quad image (signal)
Let’s run the box filter

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

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

output  \quad filter  \quad image (signal)
Let’s run the box filter

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

output \quad \text{filter} \quad \text{image (signal)}
Let’s run the box filter

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

output  \quad filter  \quad image (signal)
Let’s run the box filter

\[
h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]
\]

output filter image (signal)
Let’s run the box filter

$$h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]$$

output \quad filter \quad image (signal)
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]
Let’s run the box filter

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]
... and the result is

\[
h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]
\]

output \quad filter \quad image (signal)
Correlation (linear relationship)

\[ f \otimes h = \sum_k \sum_l f(k, l)h(k, l) \]

\( f = \text{Image} \)

\( h = \text{Kernel} \)

\[
\begin{array}{ccc}
  f_1 & f_2 & f_3 \\
  f_4 & f_5 & f_6 \\
  f_7 & f_8 & f_9 \\
\end{array}
\]

\[
\begin{array}{ccc}
  h_1 & h_2 & h_3 \\
  h_4 & h_5 & h_6 \\
  h_7 & h_8 & h_9 \\
\end{array}
\]

\[ f \otimes h = f_1 h_1 + f_2 h_2 + f_3 h_3 \\
+ f_4 h_4 + f_5 h_5 + f_6 h_6 \\
+ f_7 h_7 + f_8 h_8 + f_9 h_9 \]
**Convolution**

\[ f * h = \sum_{k} \sum_{l} f(k, l)h(-k, -l) \]

- \( f = \text{Image} \)
- \( h = \text{Kernel} \)

<table>
<thead>
<tr>
<th>( f )</th>
<th>( h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>( h_1 )</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>( h_2 )</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>( h_3 )</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>( h_4 )</td>
</tr>
<tr>
<td>( f_5 )</td>
<td>( h_5 )</td>
</tr>
<tr>
<td>( f_6 )</td>
<td>( h_6 )</td>
</tr>
<tr>
<td>( f_7 )</td>
<td>( h_7 )</td>
</tr>
<tr>
<td>( f_8 )</td>
<td>( h_8 )</td>
</tr>
<tr>
<td>( f_9 )</td>
<td>( h_9 )</td>
</tr>
</tbody>
</table>

- \( X - \text{flip} \)
- \( Y - \text{flip} \)

\[ f * h = f_1 h_9 + f_2 h_8 + f_3 h_7 \]
\[ + f_4 h_6 + f_5 h_5 + f_6 h_4 \]
\[ + f_7 h_3 + f_8 h_2 + f_9 h_1 \]
Correlation and Convolution

• **Convolution** is a filtering operation
  • expresses the amount of overlap of one function as it is shifted over another function

• **Correlation** compares the similarity of two sets of data
  • relatedness of the signals!
Key properties of linear filters

**Linearity:**
\[ \text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2) \]

**Shift invariance:** same behavior regardless of pixel location
\[ \text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f)) \]

Any linear, shift-invariant operator can be represented as a convolution

Source: S. Lazebnik
More properties

• **Commutative**: $a * b = b * a$
  – Conceptually no difference between filter and signal
  – But particular filtering implementations might break this equality

• **Associative**: $a * (b * c) = (a * b) * c$
  – Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  – This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$

• **Distributes over addition**: $a * (b + c) = (a * b) + (a * c)$

• ** Scalars factor out**: $k a * b = a * k b = k (a * b)$

• **Identity**: unit impulse $e = [0, 0, 1, 0, 0]$, $a * e = a$
Filtering Examples - 1

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\times
\begin{array}{c}
\end{array}
= 
\begin{array}{c}
\end{array}
\]
Filtering Examples - 2

\[
\begin{bmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\ast
\begin{bmatrix}
\end{bmatrix}
\]

=
Filtering Examples - 2

\[
\begin{array}{ccc}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

* 

\[
\begin{array}{ccc}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

= 

\[
\begin{array}{ccc}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]
Example: box filter

What does it do?
- Replaces each pixel with an average of its neighborhood

Average: mean
- Dividing the sum of N values by N
Filtering Examples - 3

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

\[ * \frac{1}{9} = \]

\[ \begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array} \]
Filtering Examples - 3

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

\[
\frac{1}{9}
\]

=
Example: box filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)
Filtering Examples - 4

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

* \frac{1}{25} =

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter

\[
\begin{pmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{pmatrix}
\quad = \quad
\begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix}
\quad \ast \quad
\begin{pmatrix}
1 & 1 & 1 \\
\end{pmatrix}
\]

column

row

What is the rank of this filter matrix?
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

Example: box filter

Why is this important?
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example: 
box filter

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

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

column

row

2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).
A 2D filter is separable if it can be written as the product of a “column” and a “row”.

Example: box filter

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

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

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

column

row

2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

If the image has M x M pixels and the filter kernel has size N x N:
• What is the cost of convolution with a non-separable filter?
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\quad = 
\begin{bmatrix}
1 \\
1 \\
1 \\
\end{bmatrix}
\ast
\begin{bmatrix}
1 & 1 & 1 \\
\end{bmatrix}
\]

column
row

example: 
box filter

2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

If the image has M x M pixels and the filter kernel has size N x N:

- What is the cost of convolution with a non-separable filter? \(M^2 \times N^2\)
- What is the cost of convolution with a separable filter?
Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

```
1 1 1
1 1 1
1 1 1
```

column

```
1
1
1
```

row

2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

If the image has M x M pixels and the filter kernel has size N x N:

- What is the cost of convolution with a non-separable filter? $M^2 \times N^2$
- What is the cost of convolution with a separable filter? $2 \times N \times M^2$
The Gaussian filter

• named (like many other things) after Carl Friedrich Gauss

• kernel values sampled from the 2D Gaussian function:

\[ f(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2 + j^2}{2\sigma^2}} \]

• weight falls off with distance from center pixel

• theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?
The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:
  \[ f(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \]
- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?
- usually at 2-3\(\sigma\)
The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:
  \[ f(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \]
- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?
- usually at 2-3\(\sigma\)

Is this a separable filter? Yes!

\[
\begin{array}{ccc}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{array}
\]
The Gaussian Filter

\[ g(x) = e^{-\frac{x^2}{2\sigma^2}} \]

\[ g(x, y) = e^{-\frac{(x^2+y^2)}{2\sigma^2}} \]

\[ g(x) = [0.11 \ 0.13 \ 0.6 \ 1 \ 0.6 \ 0.13 \ 0.011] \]

\[ \sigma = 1 \]
Gaussian filters

\( \sigma = 1 \text{ pixel} \) \quad \sigma = 5 \text{ pixels} \quad \sigma = 10 \text{ pixels} \quad \sigma = 30 \text{ pixels}
Filtering Examples - 5
Filtering Examples - 5

Gaussian Smoothing
Filtering Examples - 6

Gaussian Smoothing

Smoothing by Averaging
Filtering Examples - 7

After additive Gaussian Noise

After Averaging

After Gaussian Smoothing
Filtering Examples – 8
Sharpening

- do nothing for flat areas
- stress intensity peaks
Filtering Examples – 8
Sharpening

Accentuates differences with local average

- do nothing for flat areas
- stress intensity peaks
Sharpening

• What does blurring take away?

Let’s add it back:

(This “detail extraction” operation is also called a \textit{high-pass filter})

Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/
Sharpening

• What does blurring take away?

original

–

smoothed (5x5)

(This “detail extraction” operation is also called a *high-pass filter*)

original

+

detail

=

original

sharpened

Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/
Sharpening

• What does blurring take away?

2 times original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

Smoothed

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

\[\frac{1}{9}\]

(This “detail extraction” operation is also called a high-pass filter)

Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/
Sharpening

- What does blurring take away?

(This “detail extraction” operation is also called a high-pass filter)

Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/
Sharpening examples
Median Filter

- A **Median Filter** operates over a window by selecting the median intensity in the window.
Image filtering - median

\[ f[\ldots] \]

\[ h[\ldots] \]
Image filtering - median

Median of \{0,0,0,0, 90, 90,90,90,90\}
Median Filter

- A **Median Filter** operates over a window by selecting the median intensity in the window.

- Great to deal with salt and pepper noise!
Median Filter

Mean

Gaussian

Median

3x3

5x5

7x7
Practical matters

What about near the edge?

- The filter window falls off the edge of the image
- Need to extrapolate
- methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge

Source: S. Marschner
Questions?