Now Entering

The Third Dimension!
3D Signal Processing

\[ f(x, y, z) \]  

No, not this one!

\[ f(x, y, t) \]  

This one!
Video Processing

- Not volumetric 3D signal processing, but processing of video streams
  - Set of 2D image frames
- Typical algorithms operate on a frame-by-frame basis with some state carried among frames
- Many video processing algorithms (some of these apply to still images as well):
  - Detection / recognition
  - Tracking
  - Compression
  - 3D reconstruction
Algorithm Performance

• Based on processing time per frame, we can express the performance of the algorithm in terms of frames per second
  • Very common metric in computer gaming / display systems
• Human visual system can perceive up to 1000 fps under certain circumstances
  • 13 - 20 fps: video motion becomes fairly fluid
  • 24 fps: broadcast TV / motion picture standard
  • 30 - 60 fps: gaming
  • 120+ fps: TV [with interpolation]
Algorithm Performance

- Insufficient FPS?
  - Live with it
  - Drop frames
  - Drop pixels
  - Drop frames and/or pixels and interpolate result
- Decreasing frame rate or resolution can potentially make things harder due to
  - lower temporal correlation
  - lower resolution
- Target FPS can put a significant limit on how much computation your algorithm can perform on each frame
2D DFT

\[ X[k, \ell] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-j2\pi \left( \frac{km}{M} + \frac{\ell n}{N} \right)} \]

Direct implementation: \( O(N^4) \) [ouch!]

\[ X[k, \ell] = \sum_{m=0}^{M-1} e^{-j2\pi \frac{km}{M}} \sum_{n=0}^{N-1} x[m, n] e^{-j2\pi \frac{\ell n}{N}} \]

Separable implementation: \( O(N^3) \) [better!]

Replace direct sums with FFT

\[ y[m, \ell] = F_n\{x[m, n]\} \]
\[ X[k, \ell] = F_m\{y[m, \ell]\} \]

2D FFT: \( O(N^2 \log N) \) [best!]
2D DFT

• 2D DFT samples span \([0, 2\pi)\) in each dimension
  • Samples are conjugate-symmetric about the origin
  • `fftshift()` moves the DC component to the image center for easier visualization
• Also images tend to have a **VERY** strong DC component, so some manipulation of magnitude values is necessary for visualization
  • `log`, `sqrt`, etc.
  • If your DFT looks empty, check the DC pixel!
2D Convolution with DFT

- Multidimensional extension of the convolution theorem
  - $y[m, n] = x[m, n] \ast \ast h[m, n] = F_2^{-1}\{F_2\{x\}F_2\{h\}\}$
- When using the 2D DFT, we get 2D circular convolution
2D Convolution with DFT

- We want to apply a (mostly) zero phase filter $h[m, n]$
- The ‘center’ of $h$ needs to be at the [0,0] location
- Other patches of $h$ wrap around
  - $h$ is non-causal, which results in circular wrapping

- Zero padding the image prior to DFT yields linear convolution
  - Still need to rearrange $h$ as above, or accommodate pixel shift
  - Can also leverage `ifftshift()` to restructure $h$ appropriately
Brief Review of Matrix Operations

• An $m$ by $n$ matrix has $m$ rows and $n$ columns

• Elements indexed as $a_{ij}$ for element in row $i$ and column $j$

\[
\begin{bmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]

• Input data (samples, state, etc.) represented as a column vector ($m$ by 1 matrix)

\[
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6
\end{bmatrix}
\]

• Higher dimensional input data (e.g. images) ‘stacked’ to form a 1D vector

• A matrix variable is usually written in bold, using lowercase ($\mathbf{x}$) for a column matrix and uppercase for a ‘full’ matrix operator ($\mathbf{A}$)
**Brief Review of Matrix Operations**

- Addition/subtraction is element wise application of operation

\[
A + B = \begin{bmatrix}
a_{11} + b_{11} & a_{12} + b_{12} \\
a_{21} + b_{21} & a_{22} + b_{22} \\
a_{31} + b_{31} & a_{32} + b_{32}
\end{bmatrix}
\]

- Multiplication is inner products between rows and columns of respective matrices

\[
C = AB, \quad c_{ij} = \sum_k a_{ik} b_{kj}
\]

- Instead of ‘division’ we talk about matrix inverse \( A^{-1} \)

\[
A^{-1}A = I
\]
Brief Review of Matrix Operations

- Identity matrix \( I \) is 1 on the diagonal and 0 everywhere else

\[
I = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

- A matrix is *diagonal* if its non-zero elements are on the diagonal only

\[
A = \begin{bmatrix}
a_{11} & 0 & 0 \\
0 & a_{22} & 0 \\
0 & 0 & a_{33} \\
\end{bmatrix}
\]

- The inverse of a diagonal matrix is easily calculated

\[
A^{-1} = \begin{bmatrix}
1/a_{11} & 0 & 0 \\
0 & 1/a_{22} & 0 \\
0 & 0 & 1/a_{33} \\
\end{bmatrix}
\]
Brief Review of Matrix Operations

• Matrix transpose flips elements about the diagonal

\[ A^T = \begin{bmatrix}
  a_{11} & a_{21} & a_{31} \\
  a_{12} & a_{22} & a_{32}
\end{bmatrix} \]

• Hermitian \( A^H \) is a matrix transpose with conjugation of each element

• Norm is defined as \( \|A\| = \sqrt{\sum_{i,j} a_{ij}^2} \)

• An operator is defined as linear if

\[ Ax + Ay = A(x + y), \quad A\alpha x = \alpha Ax \]

• All linear operators can be written as a matrix!
Detection vs. Tracking

• Detection
  • Usually posed as a single-frame / image problem
  • Is there a particular object present?
    • Where is it?
    • What is it?

• Tracking
  • Given a starting location/description (seed)
  • Follow object as it traverses scene
  • May also want to estimate/report changes in “pose”
    • How is it oriented / configured?
  • Tracking will typically involve some detection
Challenges in Tracking

- Motion
- Size/Orientation
- Occlusion
Kalman Filter

• General problem statement:
  • Given a model of the system state evolution, estimate progression of system state over time, given system measurements

• State update equation
  • \( x_t = F_t x_{t-1} + B_t u_t + w_t \)
  • \( x_t \) - system state vector
  • \( F_t \) - state transition matrix
  • \( u_t \) - system control vector
  • \( B_t \) - control input matrix
  • \( w_t \) - process noise (with covariance \( Q_t \))
Kalman Filter

- **State measurement**
  - \( z_t = H_t x_t + v_t \)
  - \( z_t \) - measured data
  - \( H_t \) - measurement matrix
  - \( v_t \) - measurement noise (covariance \( R_t \))

- **Kalman filter algorithm has two parts**
  - Prediction step
  - Measurement update step

- For notational simplicity, let \( H_t = I \)
Kalman Filter - Prediction

- Given past state estimate, calculate new state estimate
  - \( \hat{x}_{t|t-1} = F_t \hat{x}_{t-1|t-1} + B_t u_t \)
- Notation \( \hat{x}_{a|b} \)
  - Estimate of \( x \) at time \( t = a \) given measurements up to time \( t = b \)
- This update propagates the estimated state forward
- Key to the Kalman filter is keeping track of the *certainty* of our estimates
  - \( P_{t|t-1} = Var[x_t - \hat{x}_{t|t-1}] = F_t P_{t-1|t-1} F_t^T + Q_t \)
- Note that at this point we have updated the state without any feedback from the system
Kalman Filter - Prediction

\[ \hat{x}_{t-1|t-1} \]

\[ \hat{x}_t|t-1 \]
Kalman Filter - Measurement update

• Given noisy measurements, update the state estimation
  
  • \( \hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t (z_t - \hat{x}_{t|t-1}) \)
  
  • \( K_t = P_{t|t-1} (P_{t|t-1} + R_t)^{-1} \)
  
• Note that at no point in time do we assume a perfect state value
  
  • Every vector has an associated uncertainty with it
  
• Updated certainty of estimate
  
  • \( P_{t|t} = Var[x_t - \hat{x}_{t|t}] = P_{t|t-1} - K_t P_{t|t-1} \)
  
• How did these updates come about?
Kalman Filter - Measurement Update

\[ \hat{x}_{t|t-1, Z_t} \]

\[ \hat{x}_{t|t} \]
Fusing Measurements

• Consider two noisy measurements $x_1, x_2$ with different variances $\sigma_1^2, \sigma_2^2$
  • How should these be ‘optimally’ combined?
• Consider a linear combination of the two measurements that minimizes the variance of the combined estimate
  • $\hat{x}_{opt} = \min_{\alpha} Var[(1 - \alpha)x_1 + \alpha x_2]$
• This is achieved by ‘Kalman Gain’ $K$
  • $\alpha = K = \sigma_1^2 / (\sigma_1^2 + \sigma_2^2)$
• Yielding
  • $\hat{x}_{opt} = x_1 + K(x_2 - x_1)$
  • $Var[\hat{x}_{opt}] = (1 - K)\sigma_1^2$
Fusing Measurements - Kalman Filter

- In the Kalman Filter derivation, we want to estimate $\hat{x}_{t|t}$ given
  - $\hat{x}_{t|t-1}$, which has variance $P_{t|t-1}$
  - $z_t$, which has variance $R_t$
- Applying the ‘optimal’ fusion of these two measurements from the scalar case

<table>
<thead>
<tr>
<th>Variable</th>
<th>Scalar Fusion</th>
<th>Kalman/Vector Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>$\sigma_1^2/ (\sigma_1^2 + \sigma_2^2)$</td>
<td>$P_{t</td>
</tr>
<tr>
<td>$\hat{x}_{t</td>
<td>t}$</td>
<td>$x_1 + K(x_2 - x_1)$</td>
</tr>
<tr>
<td>$P_{t</td>
<td>t}$</td>
<td>$(1 - K)\sigma_1^2$</td>
</tr>
</tbody>
</table>

- The attractive feature of Kalman filtering is its simple, recursive form
Example of Kalman Video Tracking

- Consider tracking a ball
- Provided an initial location
- Estimate new ball location
- Check for ball near new location, update based on discrepancy
- If no ball detected, continue propagating state without measurement reinforcement
Correlation Filter Tracking

- Not correlation of image patches with each other but rather correlation with a classifier filter
- In a *training phase* a target image/patch is provided which is used to construct the classifier filter
  - The filter is designed so that its response to the training image is similar to a predefined regression target image (e.g. a Gaussian)
- In the *tracking phase* applies the classifier filter to patches in the image
  - Large responses = high correlation = the object we are looking for!
Correlation Filter Tracking

• Selecting which sections of the image to test can be tricky
  • Correlation evaluation can be costly per patch
  • Insufficient patch coverage leads to loss of tracking performance

• Test all the patches using the DFT / convolution
  • Apply a window to attenuate circular wrapping effects

• Look for maximum response and update classifier filter

• FFT implementation allows for very efficient tracking algorithm

<table>
<thead>
<tr>
<th></th>
<th>Storage</th>
<th>Bottleneck</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Sampling</td>
<td>Features from p subwindows</td>
<td>Learning algorithm (Struct. SVM [4], Boost [3, 6]...)</td>
<td>10 - 25 FPS</td>
</tr>
<tr>
<td>(p random subwindows)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Sampling</td>
<td>Features from one image</td>
<td>Fast Fourier Transform</td>
<td>320 FPS</td>
</tr>
<tr>
<td>(all subwindows, proposed method)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OpenCV

• Open Source Computer Vision Library

• Implements main different computer vision algorithms with focus on real-time applications

  • Can leverage multiple cores, hardware accelerators

• Among other areas has support for facial and gesture recognition, object identification, segmentation, motion tracking, machine learning, image filtering and transforms, drawing

• C++, Python and Java Interfaces

• Active community with continual contributions

• Goal is not to reinvent the wheel
Lab 7

- Video Processing
- Utilize KCF to track an object of interest
  - Identified at start of algorithm’s execution by user
- Leverages OpenCV to do the heavy lifting
Assigned Project Lab Proposals

• Due March 4, 2PM
• Expectations for proposal:
  • Overview of the algorithm to be implemented, including citation of sources.
  • Plan for testing and validation of the algorithm's implementation.
  • Rough idea(s) for Final Project applications of the algorithm.
• Feedback to be provided prior to starting on Assigned Project Lab
  • The earlier the proposal is submitted, the sooner it can be returned and the more time you have to adjust based on feedback
Outline of Rest of Semester

• 3/17: Final Project Proposals Due

• Week of 3/25: Final Project Proposal Presentation + Assigned Lab Demo

• Week of 4/22: Final Project Demo

• 4/29: In-class Final Lecture Cumulative Quiz

• 5/3: Final Project Report and optional Video Due
This week

• Lab 6: Image Processor Quiz/Demo

• Lab 7: Video Tracker

• Assigned Project Lab Proposals Due March 4
  • Sign up groups as soon as you have them worked out
  • Submission of proposals on Compass (available soon)