CS 117: Project in Computer Vision

Spring 2013

Lecture 1: Introduction
Professor Deva Ramanan

Department of Computer Science
dramanan@ics.uci.edu
Outline

• Today’s lecture:
  ○ Visual interfaces
  ○ Course logistics (assignments, goals, grading)
  ○ Introduction to MATLAB

• Project / Homework 1
Introductions

Research

Our group works on computer vision, machine learning, and computer graphics, with a focus on statistical methods for analyzing images and video. Our research tends to explore theoretical issues (such as knowledge representation and large-scale learning) that are firmly grounded in concrete applications (such as visual search and video surveillance). Historically, it has been difficult to transfer algorithms that work in controlled lab settings to unconstrained "in-the-wild" footage. Our current work attempts to bridge this divide with a focus on machine learning and geometric models. Machine learning allows one to learn models that reflect the subtle statistical regularities of the visual world, leveraging large-scale visual datasets that are now readily available. Computer graphics provides the basis for structured, geometric representations, important for capturing the complexity of visual phenomena present in real-world data.
And you?

- Years?
- Majors?
- Matlab?
- Have taken 116 / 112 / 171 / 178?

(Having at least one of these will really help!)
What’s this course?

This project-based class will focus on the algorithmic design and implementation of vision-based interfaces. Lectures will cover the background for feature extraction from video frames, motion-processing, face and hand tracking, and gesture recognition. Students will be expected to implement algorithms through a series of 5 homework assignment. Students will be required to work in MATLAB for the homework assignments.
Projects in Computer Vision: Visual Interfaces

human-computer interaction
gesture recognition

Xbox Kinect

Nintendo Wii

We’ll study the technical algorithms behind the above
The future....

http://www.bruceongames.com/2007/10/02/the-future-it-is-all-a-gesture/
Kinect holds the Guinness World Record of being the "fastest selling consumer electronics device". It sold an average of 133,333 units per day with a total of 8 million units in its first 60 days.\[[18][19][20]\] 10 million units of the Kinect sensor have been shipped as of March 9, 2011.\[1\]
Some demos...

Real-Time Hand-Tracking with a Color Glove

Robert Y. Wang¹  Jovan Popovic¹,²,³

¹Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology

²Advanced Technology Labs, Adobe Systems Incorporated

³University of Washington
Some demos...

Head Tracking for Desktop Virtual Reality Displays using the Wii Remote

Johnny Chung Lee
Human-Computer Interaction Institute
Carnegie Mellon University
Microsoft Gesture Recognition Challenge
http://gesture.chalearn.org/
Intel Perceptual SDK

Kung Pow Kevin
by
Fingertapps

October 2012

Ver. 0.3.8.2088 (08/31/12)
High profit!

The Intel® Perceptual Computing Challenge is a contest to create innovative applications using natural human interfaces such as gestures, voice, and facial tracking. Use the Intel® Perceptual Computing SDK and the Creative Interactive Gesture Camera Kit to fast track your development of application prototypes. Compete against other developers, use your imagination, and show us your vision for the future of computing!

A total of USD $1 Million in prizes are in store for the winners of this Challenge, which will be done in two phases. Phase 1 of the contest has a total of USD $185K in cash prizes and is now open for entries. Phase 2 of the Challenge follows in March 2013 and will have more than USD $800K in prizes. Developers can quickly learn this exciting technology by downloading and using the SDK and joining both phases of the Challenge.

Explore the Challenge website to learn more about the contest or REGISTER NOW.
We will build up pieces for...
This course

• We will learn the “science” behind tracking and gesture classification, and focus less on the (important!) human interface issues
• Fun application of probability and linear algebra
• MATLAB will become your friend
  – pre-loaded image libraries
  – powerful tool for prototyping
• You will learn/implement
  – Color detection, Bayesian Probability Models, Multivariate Gaussians, 2D Tracking, Dynamic Programming, Nearest Neighbor Classification, Optical-flow
More motivation

- Our vision lab has lots of projects involving tracking... looking for (additional) undergraduate involvement

Biomechanical analysis for sports
More motivation (open projects)

Tracking people in Bren Hall
More motivation (open projects)

Facial analysis
More motivation (ongoing projects)

Activity analysis (led by undergrad!)
Outline

• Today’s lecture:
  ○ Visual interfaces
  ○ Course logistics (assignments, goals, grading)
  ○ Introduction to MATLAB

• Assignment 1
CS 117
Projects in Computer Vision: Visual Interfaces
Spring 2013

[Lectures] [Matlab] [Project Submission] [Project Videos]
[Project1] [Project2] [Project3] [Project4] [Project5]

Administrivia

Instructor: Deva Ramanan (dramanan@ics.uci.edu)
Mailing list: cs117-S13@classes.uci.edu
Lectures: T, R 11:00-12:20pm DBH 1300
Office hours: W 12:30-2:00pm DBH 4072 (or by appointment)

News:

- No class on Tuesday, April 9.

Course overview

This project-based class will focus on the algorithmic design and implementation of vision-based interfaces. Lectures will cover the background for feature extraction from video frames, motion-processing, face and hand tracking, and gesture recognition. Students will be expected to implement algorithms through a series of 5 homework assignment. Students will be required to work in MATLAB for the homework assignments.

Prerequisites

CS 116 and the following undergraduate courses or their equivalent: ICS 6D/Mathematics 6D, Mathematics 6G or 3A, Mathematics 2A-B, ICS 23, or permission of the instructor.

The required course materials are available online

- Learning OpenCV by Bradski. Free online access from on-campus. Access from off-campus via the web-based VPN.
Pre-requisites

CS 116 and the following undergraduate courses or their equivalent: ICS 6D/Mathematics 6D, Mathematics 6G or 3A, Mathematics 2A-B, ICS 23, or permission of the instructor.

Tools:

Linear Algebra
Multivariate Calculus
Probability
Discrete Algorithms
Textbooks

The required course materials are available online

- [Computer Vision: Algorithms and Applications](#) by Szeliski. Textbook available online.
- [Learning OpenCV](#) by Bradski. Free online access from on-campus. Access from off-campus via the [web-based VPN](#).

Recommended course materials (all in course reserves at Library)

*Computer Vision, A Modern Approach* by Forsyth and Ponce. Modern text on computer vision. 
*Robot Vision* by Horn. Classic text on machine vision.
Grading

5 projects worth 20% each. Projects handed in 24 hours early will get 5% extra credit on the assignment (e.g. if the assignment is worth 100 points you will get 5 pts extra credit). No late assignments will be accepted. Each project will require writing code in MATLAB, which is available on about 34 machines in the CS 364 lab - the machines are in 3 rows front of the lab assistant's desk and to the left of this desk as you face away from it.

(If you are interested in this area, it may be worth it to get student copy of matlab from bookstore ~99$)
Classroom policy

You are asked to be respectful of your student colleagues and instructor in class, not being disruptive or otherwise distracting others in the classroom. This includes turning off cell-phones and not using your laptops during class.

I will try to make lectures interactive using a mixture of boardwork and slides.

You will not complete the projects unless you come to lecture and take notes!
Academic honesty

Homeworks can be discussed, but each student must independently write up their own solutions. In particular, no sharing of code. Please see the university policy on academic honesty. It is fine to use reference materials found online, but do not search for homework solutions. Rather, students are strongly encouraged to ask questions at both office hours and on the class discussion group.

cs117-S13@classes.uci.edu
Organizational Issues

• Questions
  ○ Email class distribution list
    ▪ cs117-S13@classes.uci.edu
  ○ Everyone benefits from questions/answers!
    ▪ I will e-mail responses within a day
  ○ Office hours: 12:30 to 2 on Wed

• If you can’t make office hours, request an appointment by email
An introduction to MATLAB and “background subtraction”

A large part of your success in this course depends on your ability to pick up MATLAB in the first assignment!
Background subtraction
Project/homework submissions

http://phoenix.ics.uci.edu/teaching/cs117_spring13/submission.html

Place all code and writeup (must be a pdf) for a project (e.g., hw1) in a zipfile and upload to “hw1” folder
Outline

• Today’s lecture:
  ◦ Visual interfaces
  ◦ Course logistics (assignments, goals, grading)
  ◦ Introduction to MATLAB

• Assignment 1
Outline of Matlab Tutorial

• Data Types
  – arrays: char, numeric, struct, logical, cell, others

• Operators
  – arithmetic, relational, logical

• Flow Control
  – conditionals, case, while, etc.

• M-functions
  – syntax
  – examples of simple functions
  – writing and debugging a simple MATLAB function

• Graphics
Quick demo

Read image
Show it with ‘image’
‘whos’
Read directory of images
Function call
Data Types in MATLAB

- **Array**
  - Logical
  - Char
    - ‘a’
  - Numeric
    - Uint8
      - (8 bit, from 0 to 255, e.g., pixel values)
    - Double
      - e.g., 3.2567
        - (8 bytes)
  - Structure
    - image.width = 120
    - image.name = ‘face1’
  - Others
Matlab links to Kinect

http://acberg.com/kinect/
I prefer the non-fancy command-line interface
Images

>> im = imread(‘arnold.jpg’);

>> size(a)
ans =
450 313 3

>> imshow(im); [Requires JVM]

>> image(im); [command line]

Both imshow(im) and image(im) works for color images (3D arrays) and grayscale images (2D matrices)
Uint8 and Doubles

- Double
  - almost all MATLAB functions
    - expect doubles as arguments
    - return doubles
Uint8 and Doubles

• Double
  – almost all MATLAB functions
    • expect doubles as arguments
    • return doubles
    • e.g.,

```matlab
» a = 1:10
   a =
    1     2     3     4     5     6     7     8     9    10

» b = uint8(a)
   b =
    1     2     3     4     5     6     7     8     9    10

» whos
   Name      Size         Bytes  Class
   a         1x10            80  double array
   b         1x10            10  uint8 array

» b*500
   255  255  255  255  255  255  255  255  255  255

» double(b)*500
   ans =
    500  1000  1500  2000  2500   3000  3500  4000  4500  5000
```
Uint8 and Doubles

- **Double**
  - almost all MATLAB functions
    - expect doubles as arguments
    - return doubles
  - e.g.,

- need to convert uint8 to double before performing any math operations

```matlab
» a = 1:10
a =
1  2  3  4  5  6  7  8  9  10
» b = uint8(a)
b =
1  2  3  4  5  6  7  8  9  10
» whos
Name      Size         Bytes  Class
a         1x10            80  double array
b         1x10            10  uint8 array

» b*500
ans =
255  255  255  255  255  255  255  255  255  255

» double(b)*500
ans =
500  1000  1500  2000  2500  3000  3500  4000  4500  5000
```
Plotting an image

- Use imshow.m

>> help imshow

IMSHOW(I) displays the grayscale image I
IMSHOW(RGB)
  -if RGB is of type uint8, it assumes 0=dark and 255 = bright
  -if RGB is of type double, it assumes 0=dark and 1 = bright
IMSHOW(BW) displays the binary image BW. IMSHOW display pixels with the value 0 (zero) as black and pixels with the value 1 as white.
IMSHOW(X,MAP) displays the indexed image X with the colormap MAP. X takes on values from 1 to “k”, and colormap is a matrix of size “k by 3”

-Requires java virtual machine. I tend to use image.m and imagesc.m which do not.
Struct Data Type

\texttt{im.index = \begin{bmatrix} 4 & 10 & 3; \\ 12 & 12 & 2; \\ 10 & 4 & 3 \end{bmatrix};}
Struct Data Type

>> im.index = [8 10 2; 22 7 22; 2 4 7];
>> im.map = [0 0 0;0 0 .1; 0 .1 0;.....]

» whos
Name      Size      Bytes     Class

    im      1x1       590      struct

Grand total is 18 elements using 590 bytes

This is how gif images are stored internally
Arrays of Structures

```matlab
» im.index = [8 10 2; 22 7 22; 2 4 7];
» im.map = [0 0 0; 0 0 .1; 0 .1 0;.....]
» ims(1) = im;
» ims(2).index = [12 3 2; 23 3 3; 23 12 1];
» ims(2).map = ims(1).map;
» whos
    Name      Size     Bytes  Class            Attributes
    ims       1x2       894    struct array

Grand total is 28 elements using 894 bytes

» ims
```
Arrays of Structures

```matlab
» ims(1) = im;
» ims(2).index = [12 3 2; 23 3 3; 23 12 1];
» ims(2).cmap = ims(1).map;
» whos
Name        Size         Bytes  Class
image       1x2            894  struct array

Grand total is 28 elements using 894 bytes

» ims
image =

1x2 struct array with fields:
   index
   cmap

» ims(2)
ans =
   index: [3x3 double]
   map: [27x3 double]

» ims(1)
```
Array of structures (example)

```matlab
>> filelist = dir('*.jpg');
>> filelist
>> filelist(1)
>> imread(filelist(1).name);
```

Useful for enumerating all frames in a directory (eg, homeworks)
Arithmetic Operators

- Transpose, $a'$

- Power, $a^2$

- Addition, multiplication, division
  - $a(1) \times b(2)$
  - $a \times b$
    - works if $a$ and $b$ are matrices with appropriate dimensions
      (columns($a$) = rows($b$))
  - $a.*b$ (element by element)

- except for matrix operations, most operands must be of the same size, unless one is a scalar
Which will work?

» a = [2 3];
» b = [4 5];
» a(1)*b(2)

ans =
10

» a*b

??? Error using ==> *
Inner matrix dimensions must agree.

» a*b'

ans =
23

» a.*b

ans =
8 15

» b/2

ans =
2.0000 2.5000
Relational Operators

• \(<, \leq, >, \geq, ==, \sim=

• compare corresponding elements of arrays with same dimensions

• if one is scalar, one is not, the scalar is compared with each element

• result is of type Logical
  – element by element 1 or 0
» a = [2 3]

» b = [4 5]

» a > b

» b > a

» a > 2
Vectorization of Computation

tic
for i=1:100000
    y(i) = log(i);
end
toc
Vectorization of Computation

tic  
for i=1:100000  
    y(i) = log(i);
end  
toc  

elapsed_time = 168.78 seconds
Vectorization of Computation

tic
for i=1:100000
    y(i) = log(i);
end
toc

tic
i=1:100000;
z = log(i);
toc
Vectorization of Computation

First method calls the log function 100,000 times,
Second method only calls it once (much faster)
Concatenation of matrices

```matlab
» a = 1:4;
» b = [5 6 7 8];
» c = [a ; b]
» size(c)
» c'
» d = [a b]
```
Subsets of matrices

```matlab
» b = [5 6 7 8];
» a = [1:4];
» c = [a;b];
» c(1,1)

» c(1,:) 

» c(1:2,1:2)
```

How would you show the top half of an image?
Special Matrices

» ones(2,3)
ans =
1 1 1
1 1 1

» zeros(1,4)
ans =
0 0 0 0

» rand(3,3)
ans =
0.2176  0.4909  0.8985
0.4054  0.1294  0.5943
0.5699  0.5909  0.3020
Saving your work

>> ls
>> whos
>> save work
>> clear
>> whos
>> load work
>> whos

Note: You can save and load variables in the workspace.

Note: You can save the interactive commands in a file (e.g., script.m) and run the commands by typing “>> script”.
% script randscript
% A simple script to generate a vector of n random numbers. We calculate the numbers using (a) for loops, and (b) a direct function call.
%

Professor Smyth, Oct 2007

Comment text: it is always important to put comments in your code. Comments at the top should clearly explain what the script or function does.
% script randsum
% A simple script to generate a vector of n
% random numbers. We calculate the numbers
% using (a) for loops, and (b) a direct function call.
%

Professor Smyth, Oct 2007

n = 100000;       % the number of points for the "for loop"
y = zeros(n,1);   % preallocate memory for y
fprintf('Simulating %d random numbers.....\n\n',n);

Initialize various variables

Print out some information to the screen
% script randsum
% A simple script to generate a vector of n
% random numbers. We calculate the numbers
% using (a) for loops, and (b) a direct function call.
% %
% Professor Smyth, Oct 2007

n = 100000; % the number of points for the "for loop"
y = zeros(n,1); % preallocate memory for y
fprintf('Simulating %d random numbers.....\n\n',n);

% first do the calculation using a "for loop"
fprintf('For loop calculations.....\n');
tic % set the timer
for i=1:n
    y(i) = rand(1);
end
total = sum(y);
fprintf('Sum of %d random numbers = %f\n',n,total);
t1 = toc; % read the time elapsed since "tic" (in seconds)
fprintf('Time taken, using for loop = %6.5f microseconds\n\n', (t1)*1000);

(1) Calculate the n random numbers and their sum using a for loop, (2) record the time taken, and (3) print to the screen
Can we perform this faster?
Example of a MATLAB script

%%%%
% now do the calculation using vectorization
fprintf('Vectorization calculations.....\n');

tic \% reset the timer
z = rand(n,1);
total = sum(z);

fprintf('Sum of %d random numbers = %f\n',n,total);
t2 = toc; \% read the time elapsed since "tic" (in seconds)
fprintf('Time taken, using vectorization = %6.5f microseconds\n', (t2)*1000);

(1) Now calculate n random numbers
and their sum using a direct function call
(2) record the time taken,
and (3) print to the screen
Example of a MATLAB function

```matlab
function [sum, difference] = sumdiff(a, b);
```

- **Tells MATLAB this is a function**
- **List of output values returned (can be any form of array data type)**
- **Name of the function**
- **List of input argument values, comma delimited (any form of array data type)**
Example of a MATLAB function

```matlab
function [sum, difference] = sumdiff(a, b);
% A simple function to compute the sum and difference
% of two input arguments a and b
% Professor Smyth, Oct 2007

% INPUTS:
%   a: array of size r x c
%   b: array of size r x c

% OUTPUTS:
%   sum: a + b
%   difference: a - b
```

Clear comments in function headers are very useful

Note the explicit statement explaining what the inputs and outputs are (including their dimensionality)
Example of a MATLAB function

```matlab
function [sum, difference] = sumdiff(a, b);
% function [sum, difference] = sumdiff(a, b);
%
% A simple function to compute the sum and difference
% of two input arguments a and b
%
% Professor Smyth, Oct 2007
%
% error checking
[rowsa, colsa] = size(a);
[rowsb, colsb] = size(b);
if (rowsa ~= rowsb) | (colsa ~= colsb)
    error('sizes of a and b do not match');
end
```

Error checking is always a good idea!
Example of a MATLAB function

```matlab
function [sum, difference] = sumdiff(a, b);
% A simple function to compute the sum and difference
% of two input arguments a and b
% Professor Smyth, Oct 2007
%
% error checking
[rowsa, colsa] = size(a);
[rowsb, colsb] = size(b);
if (rowsa ~= rowsb) || (colsa ~= colsb)
    error('sizes of a and b do not match');
end

sum = a + b;
difference = a - b;
```

Finally, the actual computational part of the function
MATLAB graphics

The plot function is very powerful for plotting all sorts of variables. See “help plot” for more details and see the examples in the MATLAB online tutorial

```matlab
» a = 1:100;
» b = 100:0.01:101;
» c = 101:-1:1;
» d = [a b c];
» e = [d d d d d];
» plot(e)
```

What will this look like?
The plot function is very powerful for plotting all sorts of variables. See “help plot” for more details and see the examples in the MATLAB online tutorial.

```
» a = 1:100;
» b = 100:0.01:101;
» c = 101:-1:1;
» d = [a b c];
» e = [d d d d d];
» plot(e)
```
MATLAB graphics

» a = 0:0.1:10;

» subplot(3,1,1); plot(sin(a))

» r1 = rand(1,length(a))*0.2;
» subplot(3,1,2); plot(sin(a)+r1)

» r2 = rand(1,length(a))*0.8;
» subplot(3,1,3); plot(sin(a)+r2)
MATLAB images

```matlab
im = imread('face.jpg');
whos im
Name     Size    Bytes  Class   Attributes
image(im);
hold on;
plot(400,20,'r*');
```
MATLAB images

```matlab
» im = imread('face.jpg');
» whos im
Name    Size    Bytes  Class      Attributes
image(im);
» hold on;
» plot(400,20,'r*');
```
Project 1

- Due next Thursday as 12:30 pm
  - Use EEE dropbox, HW1 directory
  - No late assignments accepted – submit what you have by 12:30

- 2 parts:
  1. Online MATLAB tutorials (important!)
  2. Write “background subtraction” code

  - Compute average image in a video sequence
  - Compute a “background subtracted image” using:
    - Squared euclidean distance between 2 RGB vectors \( \mathbf{x} \) and \( \mathbf{y} \)
      - \( \text{distance}(\mathbf{x},\mathbf{y}) = \left( \sum_i (x_i - y_i)^2 \right) \)
      - sum is over the components of the vectors,
        - e.g., \( \mathbf{x} = [1 \ 2 \ 3], \mathbf{y} = [4 \ 5 \ 6] \)
        - \( \text{distance}(\mathbf{x},\mathbf{y}) = (1-4)^2 + (2-5)^2 + (3-6)^2 ) = ... \)
Summary

• Today’s lecture
  ◦ Class overview
  ◦ A broad overview of MATLAB
  ◦ Start reading tutorial material in Assignment 1 to get more details

• Next Lecture
  ◦ Fancy background subtraction