CS 1114: Introduction to Computing Using MATLAB and Robotics

Prof. Graeme Bailey

http://cs1114.cs.cornell.edu

(notes modified from Noah Snavely, Spring 2009)

Overview

- What is CS 1114?
  - An honors-level intro to CS using camera-controlled robots (Rovio, Sony Aibo, iRobot Create)
  - An alternative to CS1112 or CS1132, to fulfill your Matlab computing requirement
  - Formerly known as CS100R

Goals of CS1114

- Give you an intuition about computation problem solving
- Teach you useful (and interesting) computer science
- Give you fluency in the Matlab programming environment
- Have fun with robots

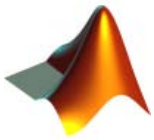
Requirements

- Exposure to programming (in any language)
- Some interest in math
  - Computer science is about much more than programming, and so is this course

Java or Matlab?

- Both CS1110 and CS111[2,4] teach fundamental problem solving skills and computer science techniques

- The destination is the same…



- … but the vehicle is different

(inspired by Charlie Van Loan)

Robots: 2029

Robots: cute but dumb

- What do they know about the world around them?
  - Without your help, very little
  - Can't even notice a bright red lightstick
- Your mission: make them smarter
- Lots of interesting math and computer science, some computer programming
  - Lots of experience with programming, even with robots, won't give you a leg up in 1114

CS1114 Logistics

- Lectures: Tue Thu 11:15–12:05, PHL 307
- Sections:
  - Wed 1:25 - 2:15, Upson 317
  - Wed 2:30 - 3:20, Upson 317
  - Wed 3:35 - 4:25, Upson 317
  - Please go to same section for the entire course
- Occasional evening lectures (optional)
  - First lecture will be in next week (TBA)

CS1114 Logistics

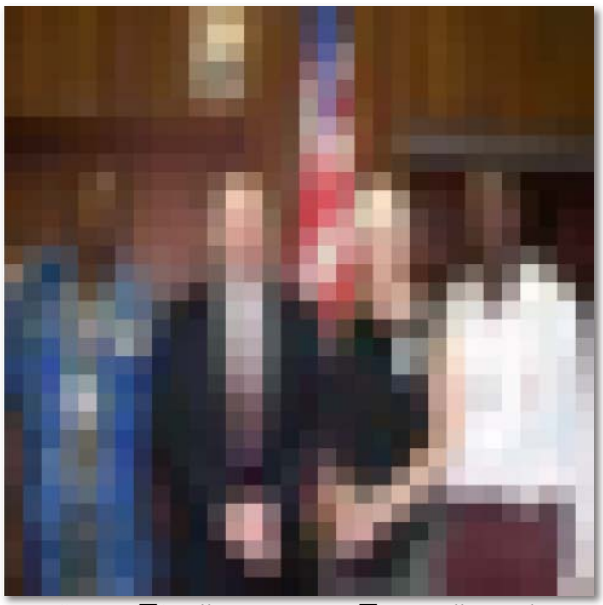
- CS1114 lab: Upson 317
- You will soon have access to the lab and passwords for the computers
- Office hours will generally be held in the lab (hours to be announced soon)

Assignments

- Several mini-quizes
  - In class, usually at start of lecture
    - Corollary: be on time, or write fast…
- 5-6 robot programming assignments with multiple parts
  - You will demo each part to the lab TA's
- 2-3 prelims
- Free-form final project (required)

What can we do with computer vision?

Human vision



Source: 80 million tiny images by Torralba, et al.

Question: How many people are in this image?

Interpreting images

Q: Can a computer (or robot) understand this image?

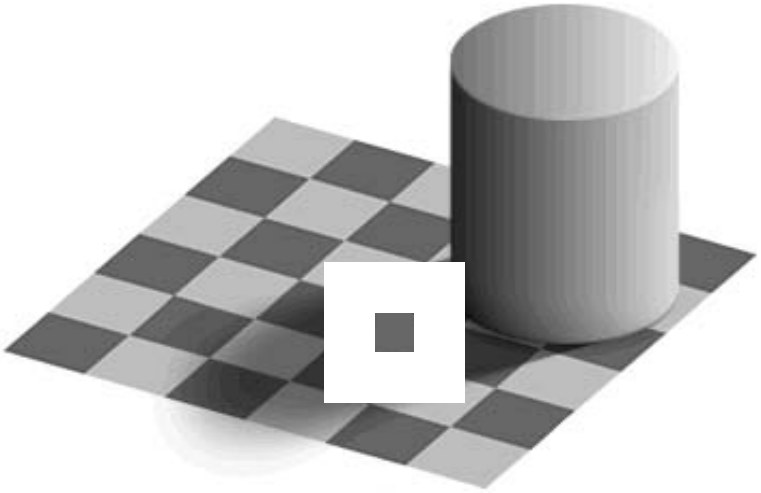
A: Yes and no (mostly no)

Human vision has its shortcomings…

Sinha and Poggio, *Nature*, 1996

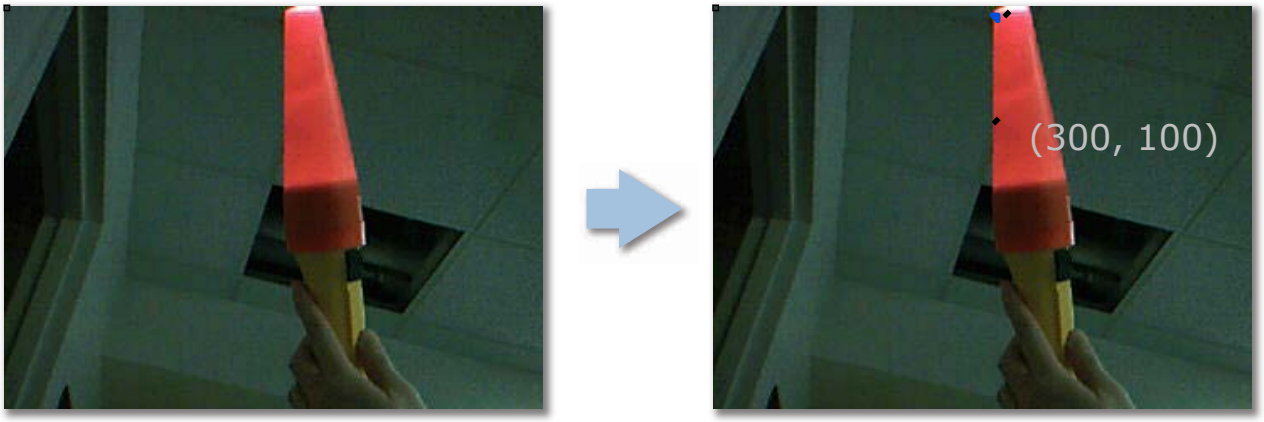
Credit: Steve Seitz

Human vision has its shortcomings…



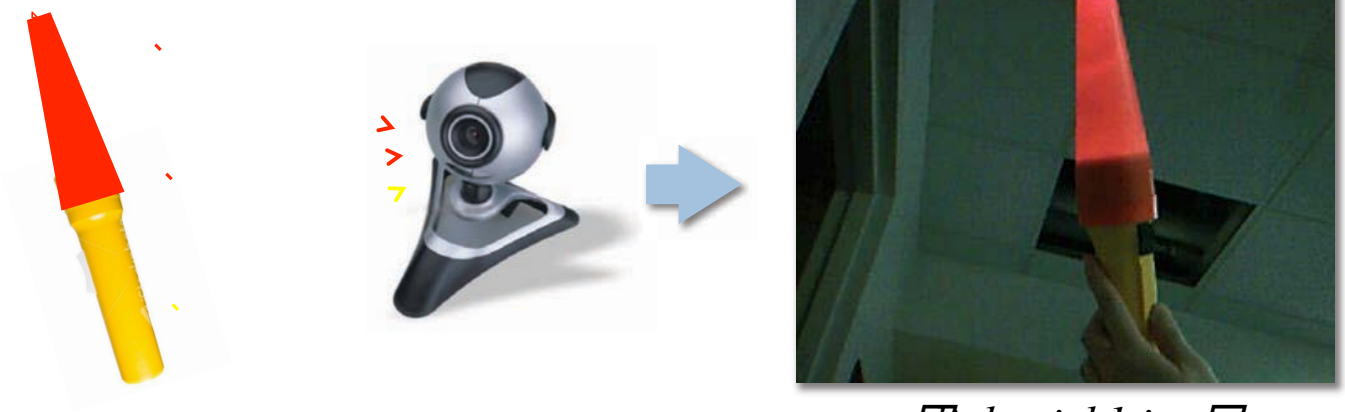
by Ted Adelson, slide credit Steve Seitz

Interpreting images



Q: Can a computer (or robot) find the lightstick?
A: With your help, yes!*

What is an image?

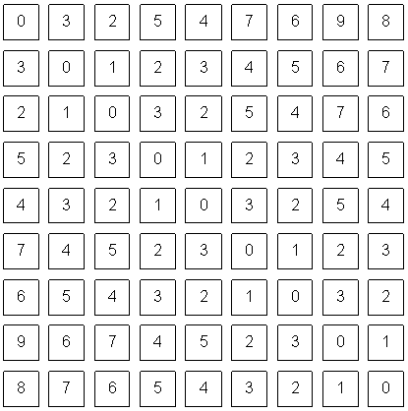


lightstick1.jpg

Major CS1114 Projects

- From a camera, figure out the position of a bright red lightstick
  - Use this to guide a robot around

What we see



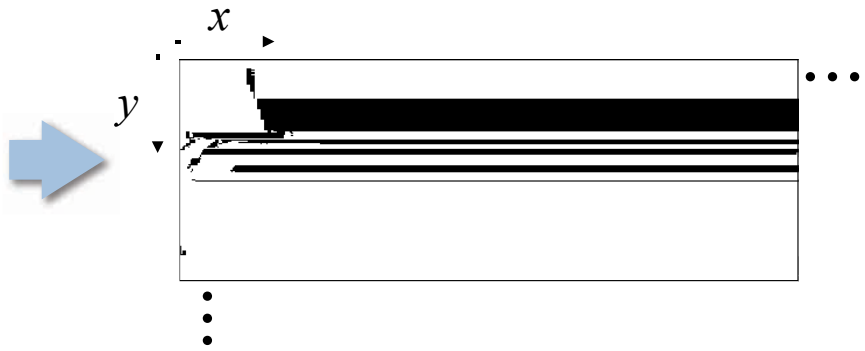
What the robot sees

What is an image?

- A grid of numbers (*intensity values*)

snoop

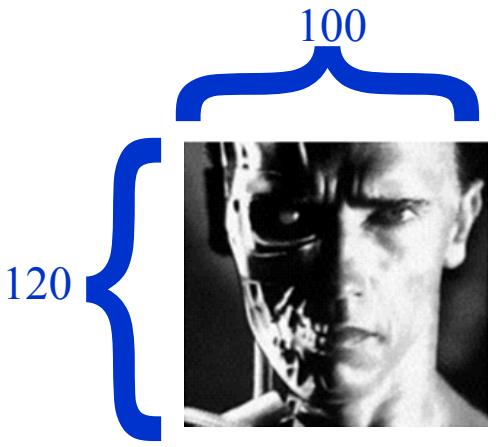
3D view



- Intensity values range between 0 (black) and 255 (white)

What is an image?

- A grid of numbers (*intensity values*)
- In Matlab, a *matrix*




```
[ 10  30  40  106  123  … ;
   8  49  58  112  145  … ;
  16  53  86  123  152  … ;
                …          ]
```

120 x 100 matrix

Matrices in Matlab

- 1D matrix is often called a vector
  - Similar to arrays in other languages

```
A = [ 10 30 40 106 123 ]
```

Row vector
(or 1 x 5 matrix)

```
A(1) ==  10
A(4) == 106
```

```
B = [  10 ;
       30 ;
       40 ;
      106 ;
      123 ]
```

Column vector (or 5 x 1 matrix)

Matrices in Matlab

```
C = [ 10  30  40 106 123 ;
       8  49  58 112 145 ;
      16  53  86 123 152  ]
```

3 x 5 matrix

```
C(1,1) == 10
```

```
C(2,4) == 112
```

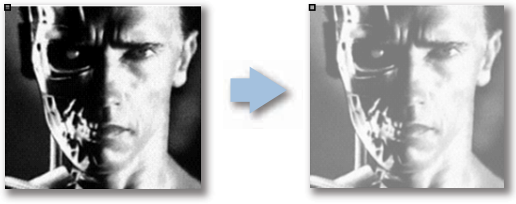
can also *assign* to a matrix entries

```
C(1,1) = C(1,1) + 1
```

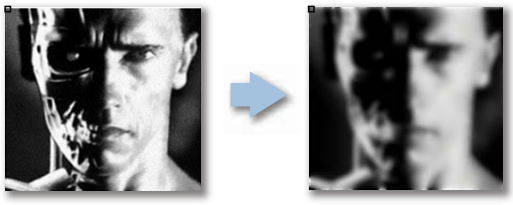

Image processing

- We often want to modify an image by □doing something□to each pixel:

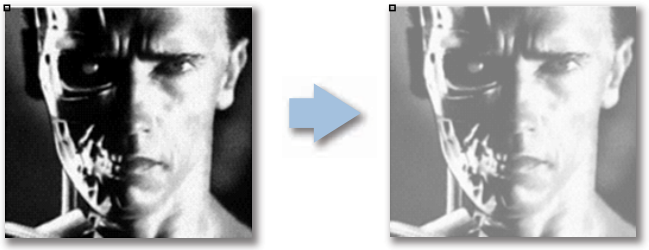
Brighten



Blur



Brightening an image



Q: What does this mean in terms of matrix entries?

```
[ 10 30 40 106 123 ;
   8 49 58 112 145 ;
  16 53 86 123 152  ]
```

A: Increase each element by some amount (say, 20)

Brightening an image (Take 1)

```
D = [ 10 30 40 106 123 8 49 58 112 145 16 53 ]
```

```
D(1)  = D(1) + 20;
      = D(2) + 2
      = D(3) +
D     = D(4)
D(5   D(5)    0;
D(6)  (       20;
D(7)          + 20;
D(8)  =       + 20;
D(9)          + 20;
D(10)         + 20;
D(1   D(1     20;
D     = D(12) 0;
      = D(13)
   4) = D(14) +
D(15) = D(15) + 20;
```

Avoiding duplicate code

- Programming languages are designed to make this easy
  - It's a huge theme in language design
  - Many new programming techniques are justified by this
    - Object-oriented programming, higher-order procedures, functional programming, etc.

Why is it a bad idea to duplicate code?

```
D(1)  = D(1) + 20;
D(2)  = D(2) + 20;
D(3)  = D(3) + 20;
D(4)  = D(4) + 20;
D(5)  = D(5) + 20;
D(6)  = D(6) + 20;
D(7)  = D(7) + 20;
D(8)  = D(8) + 20;
D(9)  = D(8) + 20;
D(10) = D(10) + 20;
D(11) = D(11) + 20;
D(12) = D(12) + 20;
```

- Hard to write
- Hard to modify
- Hard to get right
- Hard to generalize
- Programmer's "intent" is obscured

Brightening an image (Take 2)

- Using *iteration*

```
D(1)  = D(1) + 20;
D(2)  = D(2) + 20;
D(3)  = D(3) + 20;
D(4)  = D(4) + 20;
D(5)  = D(5) + 20;
D(6)  = D(6) + 20;
D(7)  = D(7) + 20;
D(8)  = D(8) + 20;
D(9)  = D(9) + 20;
D(10) = D(10) + 20;
D(11) = D(11) + 20;
D(12) = D(12) + 20;
```

```
for i = 1:12
    D(i) = D(i) + 20;
end
```

```
D = D + 20;
```

- *Vectorized code*
- *Usually much faster than loops*

  - Much easier to understand and modify the code
  - Better expresses programmer's intent

Many advantages to iteration

- Can do things with iteration that you can't do by just writing lots of statements
- Example: increment every vector cell
  - Without knowing the length of the vector!

```
len = length(D); % New Matlab function
for i = 1:len
    D(i) = D(i) + 20;
end
```

Introducing iteration into code

- Programming often involves “clichés”
  - Patterns of code rewriting
  - I will loosely call these “design patterns”
- Iteration is our first example

Brightening 2D images

```
C = [ 10  30  40 106 123 ;
       8  49  58 112 145 ;
      16  53  86 123 152  ]
```

3 x 5 matrix

```
for row = 1:3
    for col = 1:5
        C(row,col) = C(row,col) + 20;
    end
end
```

Called a “nested” for loop

Brightening 2D images

```
for row = 1:3
    for col = 1:5
        C(row,col) = C(row,col) + 20
    end
end
```

- What if it's not a 3x5 matrix?

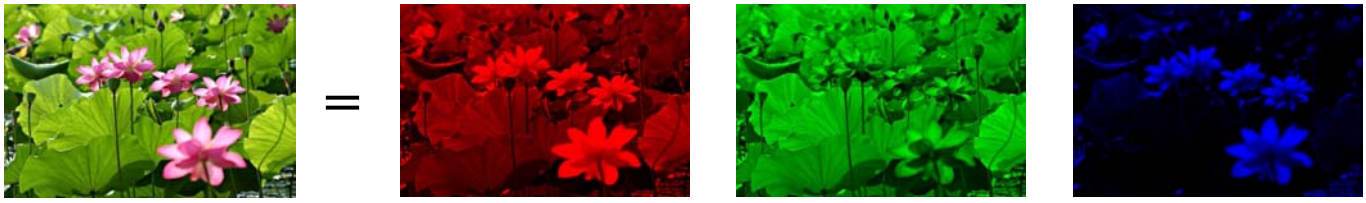
```
[nrows,ncols] = size(C) % New Matlab function
for row = 1:nrows
    for col = 1:ncols
        C(row,col) = C(row,col) + 20
    end
end
```


What about red pixels?

- A grayscale image is a 2D array
  - Brightest = 255, darkest = 0

What about red pixels?

- A color image is 3 different 2D arrays
  - For red/green/blue values (RGB)
  - We provide a way to create these 3 arrays
  - ♦ Why are there 3?



What about red pixels?

- Example colors:

red(1,1) == 255, green(1,1) == blue(1,1) == 0

red(2,1) == 100 == green(2,1) == blue(2,1)

red(3,1) == 0 == green(3,1) == blue(3,1)

red(3,1) == 255 == green(3,1), blue(3,1) == 0

For next time

- ……. Wait* and see** …… !!!***

---

* *Assuming you know how to 'wait'*

** *assuming you know how to 'see'*

*** *assuming you know why waiting helps seeing*