Vision

January 10, 2006

Agenda

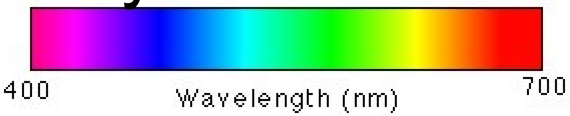
- The basics:
  - Colorspaces
  - Numbers and Java
  - Feature detection
- More advanced concepts:
  - Stereo
  - Rigid Body Motion
  - EM Algorithm

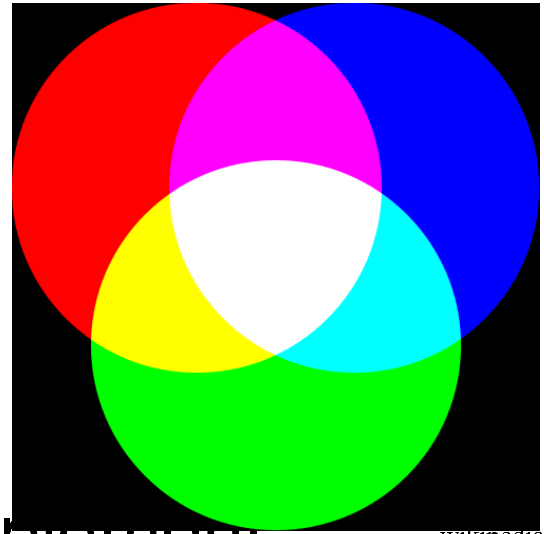
Basics

- Colorspaces
- Numbers and Java
- Feature detection

Representing color

- Frequencies are only part of the story...
- RGB good for light





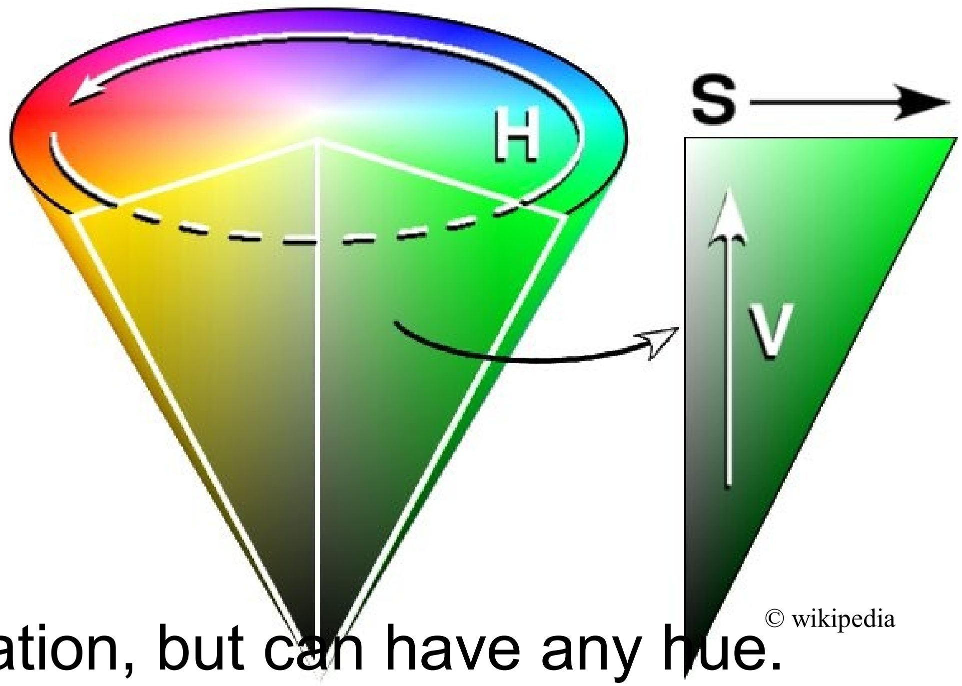
wikipedia

- CYMK good for pigment

  … but both mix color, tint, and brightness

Another Colorspace: HSV

- Hue (color): 360 degrees mapped to 0 to 255
  - Note red is both 0 and 255!
- Saturation (amount of color)
- Value (amount of light and dark)
- We provide the code to convert to HSV
- Note:



  - White is low saturation, but can have any hue.
  - Black is low value, but can have any hue.

Tips on Differentiating Colors

- Globally define thresholds
- Self-calibrate for different lights
- Use the gimp/bot client on real images
- Learn from a large sample set

... but you don't *have* to do it this way!

Last year's winning robot used RGB

How values are stored

- Uses Hexadecimal (base 16)
  - 0x12 = 18
- A color is four bytes = 8 hexadecimal numbers.
- For HSV, these bytes are
  - Alpha
  - Hue
  - Saturation
  - Value

Manipulating HSV values

- Use masks to pick out parts:
  - 0x12345678 & 0x00FF0000 = 0x00340000
- Shift to move parts around:
  - 0x12345678 >> 8 = 0x00123456
- Example: hue = (X >> 16) & 0xFF

Shift hue to least significant bits

Pick out the least significant byte

A note on java…

- All java types are signed
  - A byte ranges from –128 to 127
  - Coded in two’s complement: to change sign, flip every bit and add one
- Don’t forget higher order bits
  - (int) 0x0000FF00 = (int) 0xFF00
  - (int) ((byte) 0xFF) = (int) 0xFFFFFFFF
- Watch out for shifts
  - 0xFD000000 >> 8 = 0xFFFD0000

Example

- How about

```
int v = image.getPixel(25,25); // v = 0x8AD12390
byte hue = (v >> 16) & 0xFF   //hue = 0xD1
 if (hue > 200)
    foundRedBall();
```

200 is an int! When 0xD1 (is 209) is extended to an int, it will be a negative number!

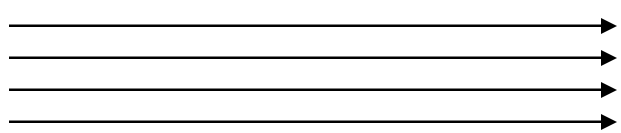
Solution

- Use

```
int v = image.getPixel(25,25); // v = 0x8AD12390
int hue = (v >> 16) & 0xFF   //hue = 0xD1
 if (hue > 200)
    foundRedBall();
```

Performance…

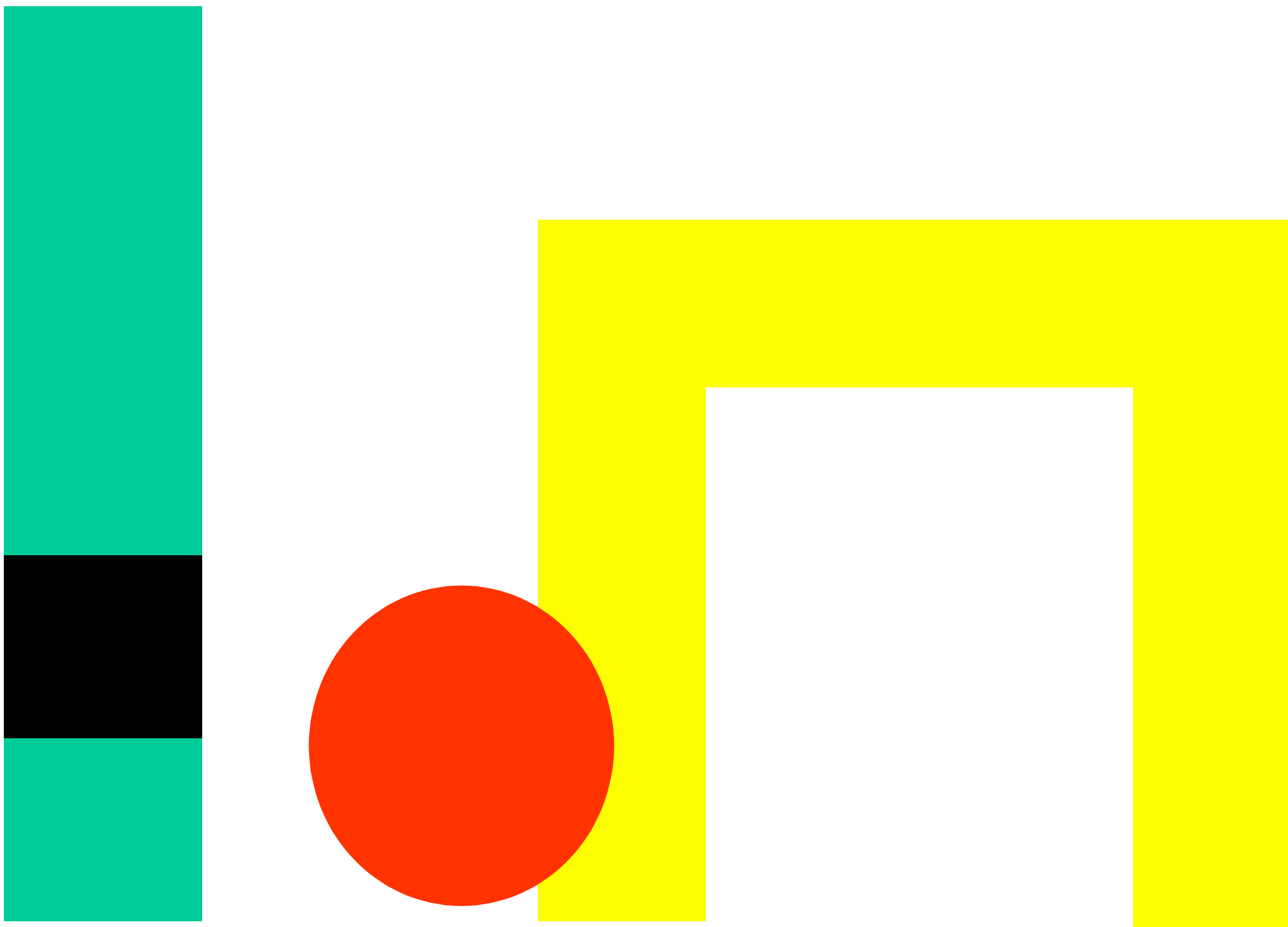
- Getting an image performs a copy
  - Int[] = bufferedImage.getRGB(…)
- Getting a pixel performs a multiplication
  - int v = bufferedImage.RGB(x,y)
  - offset = y*width + x
- Memory in rows, not columns…so go across rows and then down columns



Performance Note

- Faster access:
  - bufferedImage = ImageUtil.convertImage(bufferedImage, BufferedImage.INT_RGB);
  - DataBufferInt intBuffer = (DataBufferInt) bufferedImage.getRaster().getDataBuffer();
  - int[] b = dataBufferInt.getData();
- Need to keep track of where pixels are:
  - offset = (y*width + x)
  - (b[offset] >> 16) & 0xFF = red or hue
  - (b[offset] >> 8) & 0xFF = green or saturation
  - b[offset] & 0xFF = blue or value

Feature Detection… and other Concepts



Maslab Features

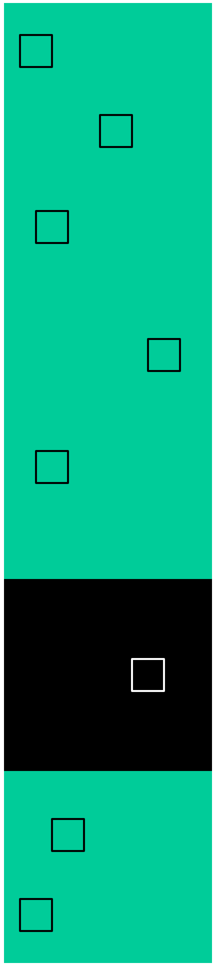
- Red balls
- Yellow Goals
- Blue line
- Blue ticks
- Bar codes

Blue line ideas

- Search for ‘n’ wall-blue pixels in a column
- Make sure there’s wall-white below?
- Candidate voting
  - in each column, list places where you think line might be
  - find shortest left to right path through candidates

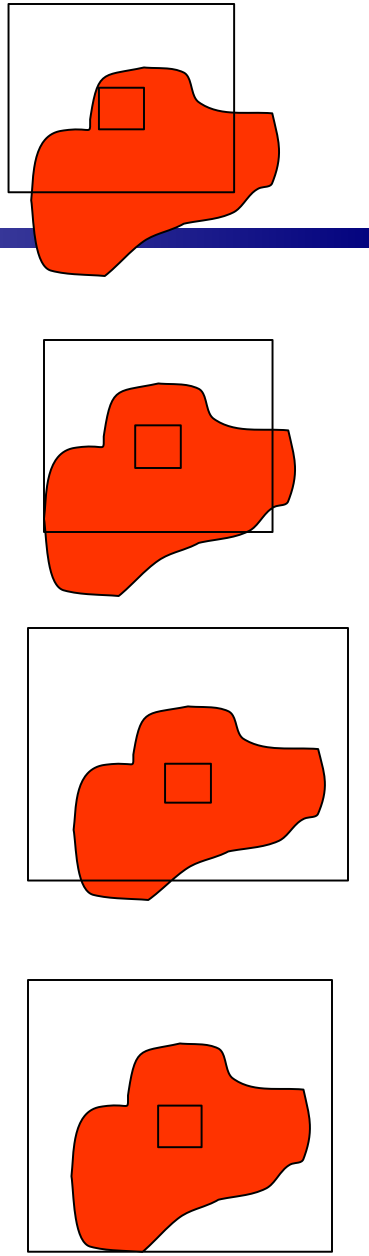
Bar code ideas

- Look for green and black
- Is there not-white under the blue line?
- Check along a column to determine colors
- RANdom SAmple Consensus (RANSAC)
  - Pick random pixels within bar code
  - Are they black or green?



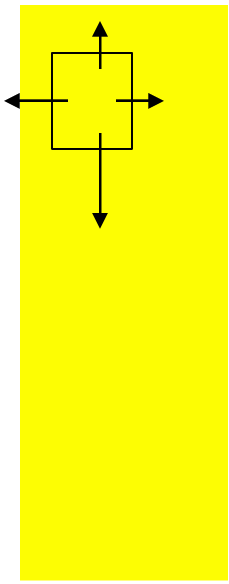
Looking for an object

- Look for a red patch
- Set center to current coordinates
- Loop:
  - Find the new center based on pixels within d of the old center
  - Enlarge d and recompute
  - Stop when increasing d doesn't add enough red pixels



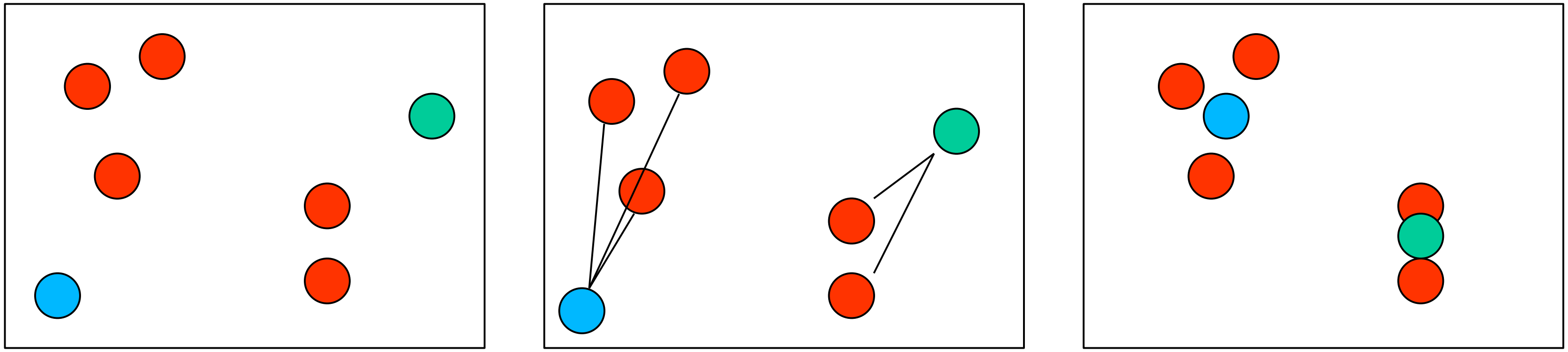
Or try fitting a rectangle

- Scan image for a yellow patch
- In each direction, loop:
  - Make rectangle bigger
  - If it doesn't add enough new yellow pixels, then stop



EM/Nearest Neighbor

- Assume there are k red objects
- Randomly choose object locations x_k, y_k
- Loop:
  - Assign each pixel to nearest x_k, y_k
  - Recenter x_k, y_k at center of all pixels associated with it

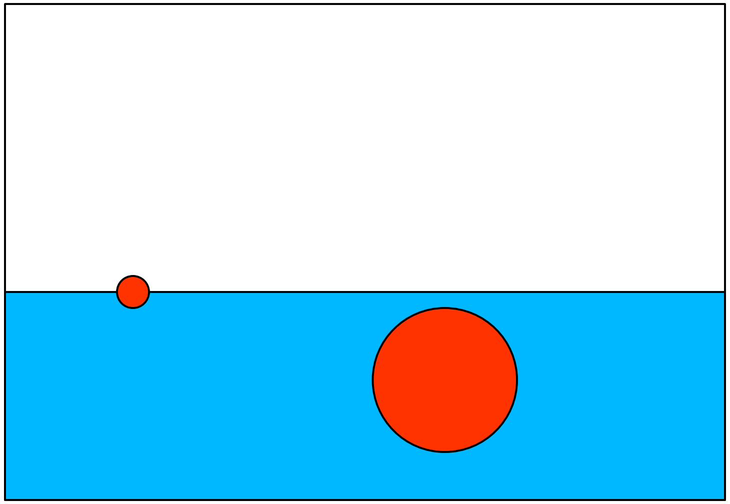


EM/Nearest Neighbor

- Key question: what is k?
  - Need to know how many objects
- Convergence criteria for random values?
  - Pick good guesses for centers

Estimating distance

- Closer objects are bigger
- Closer objects are lower



Reminders

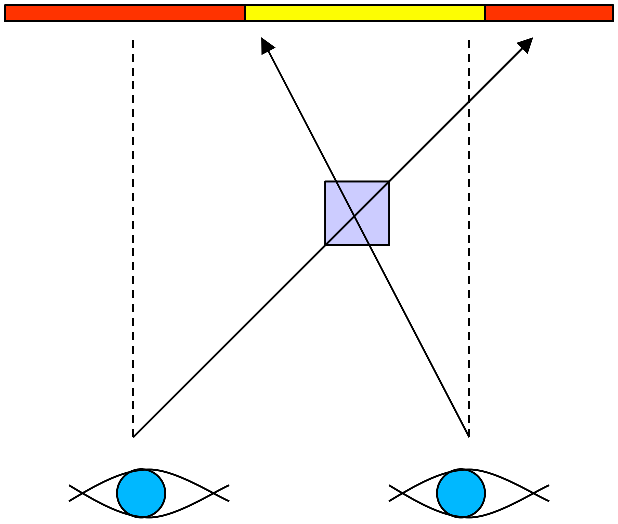
- Try out your own algorithms! Have fun!
- Must prune out silly solutions:
  - Noise
  - Occlusion
  - Acute viewing angles
  - Overly large thresholds

More Advanced Concepts

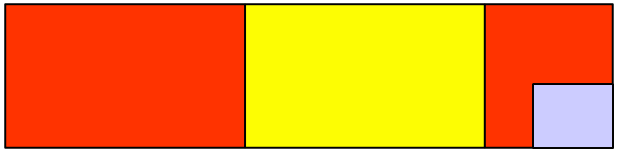
- Stereo
- Rigid Body Motion

Stereo Vision

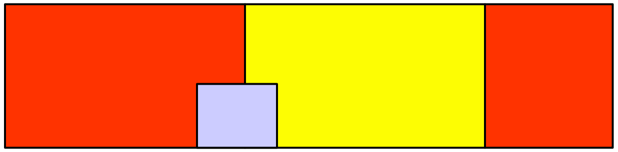
- We can judge distance based on the how much the object's position changes.



Left Eye Right Eye



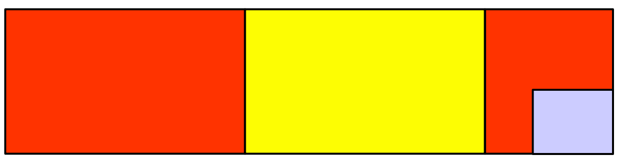
Left Image



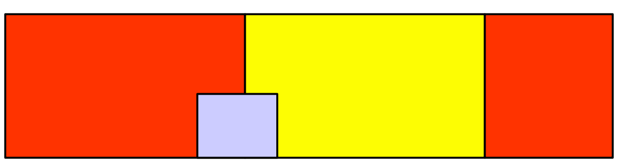
Right Image

Stereo Vision

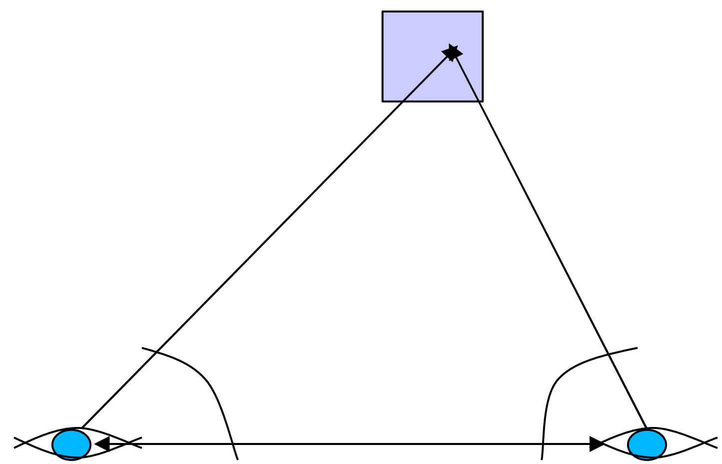
- Use the image to find the angle to the object, then apply some trig:



Left Image



Right Image

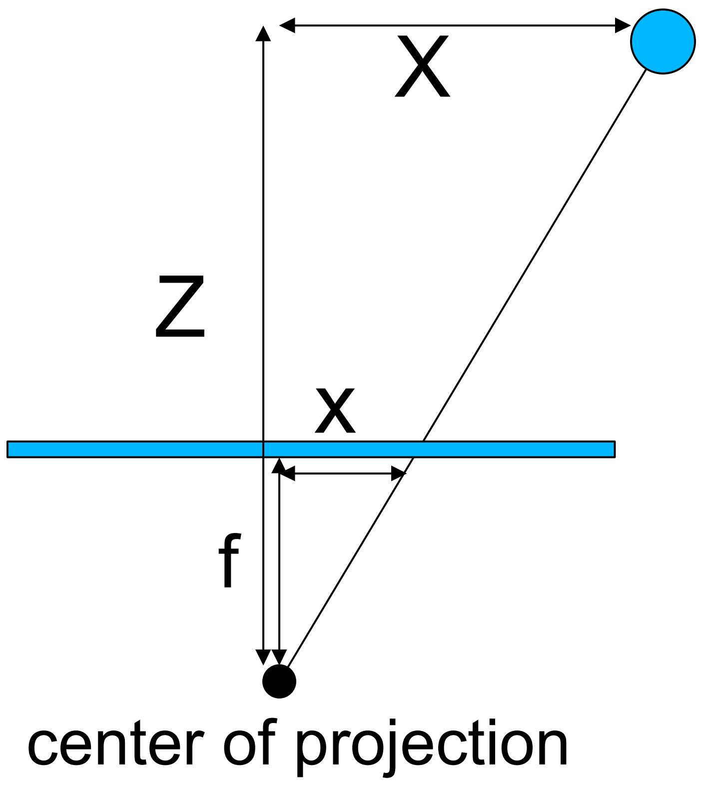


angle-side-angle gives you a unique triangle

Stereo Vision

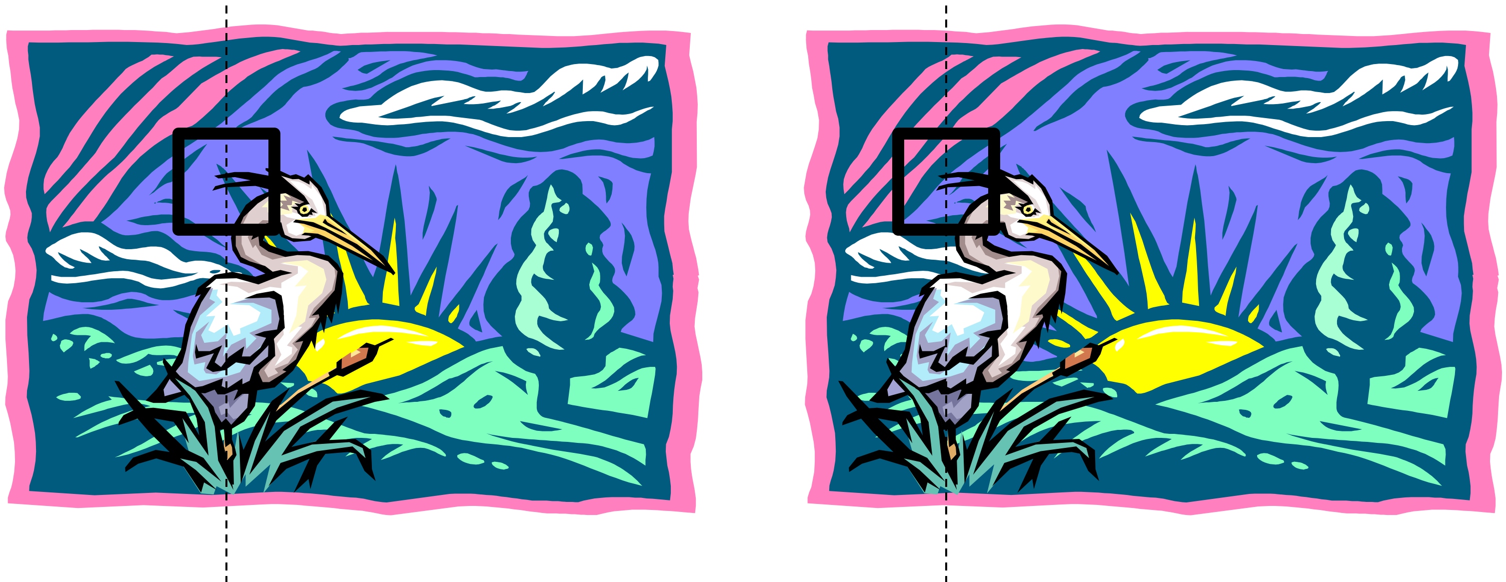
- What's the angle?
- Perspective projection equation tells us

  $x/f = X/Z$
- f is focal length, x is pixel location
- $\tan(\varphi) = X/Z = x/f$



Stereo Vision

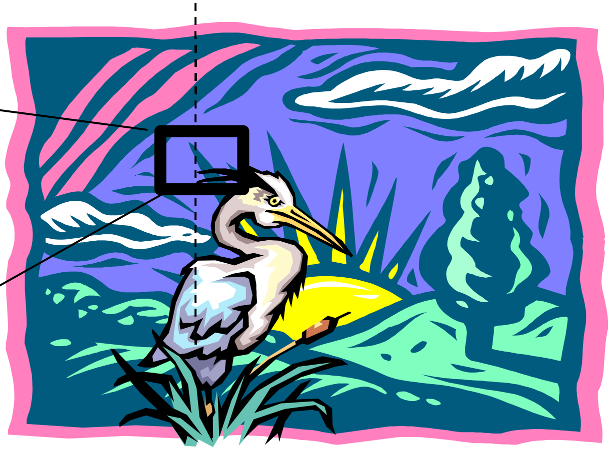
- But in a complex image, objects may be hard to identify…
- Try to match regions instead (block correlation)



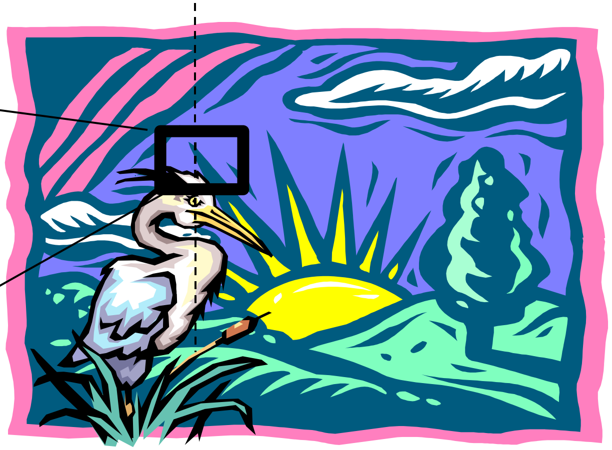
Stereo Vision

- Difference metric = Sum of $(L_i - R_i)^2$
- Search horizontally for best match (least difference)

6	5	5
5	6	5
5	5	7

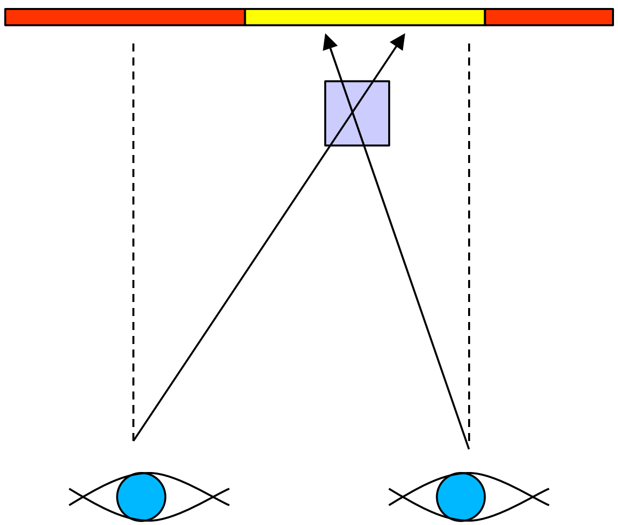


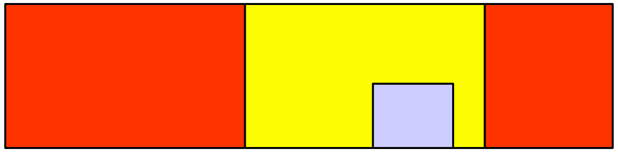
6	5	5
5	6	5
1	1	6



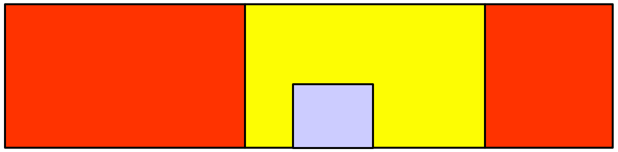
Stereo Vision

- Still have a problem: unless the object is really close, the change might be small…





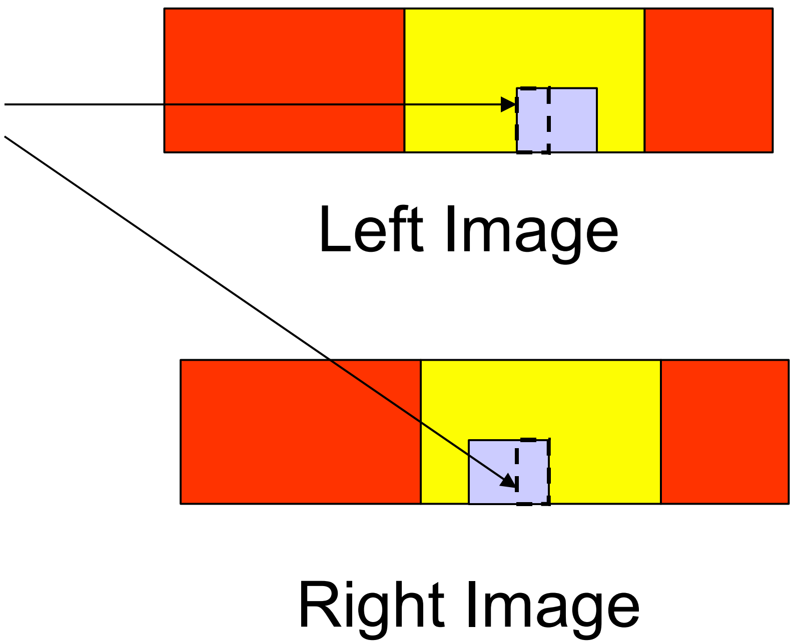
Left Image



Right Image

Stereo Vision

- And many regions will be the same in both pictures, even if the object has moved.
- We need to apply stereo only to “interesting” regions.



Stereo Vision

- Uniform regions are not interesting
- Patterned regions are interesting
- Let the “interest” operator be the lowest eigenvalue of a matrix passed over the region.

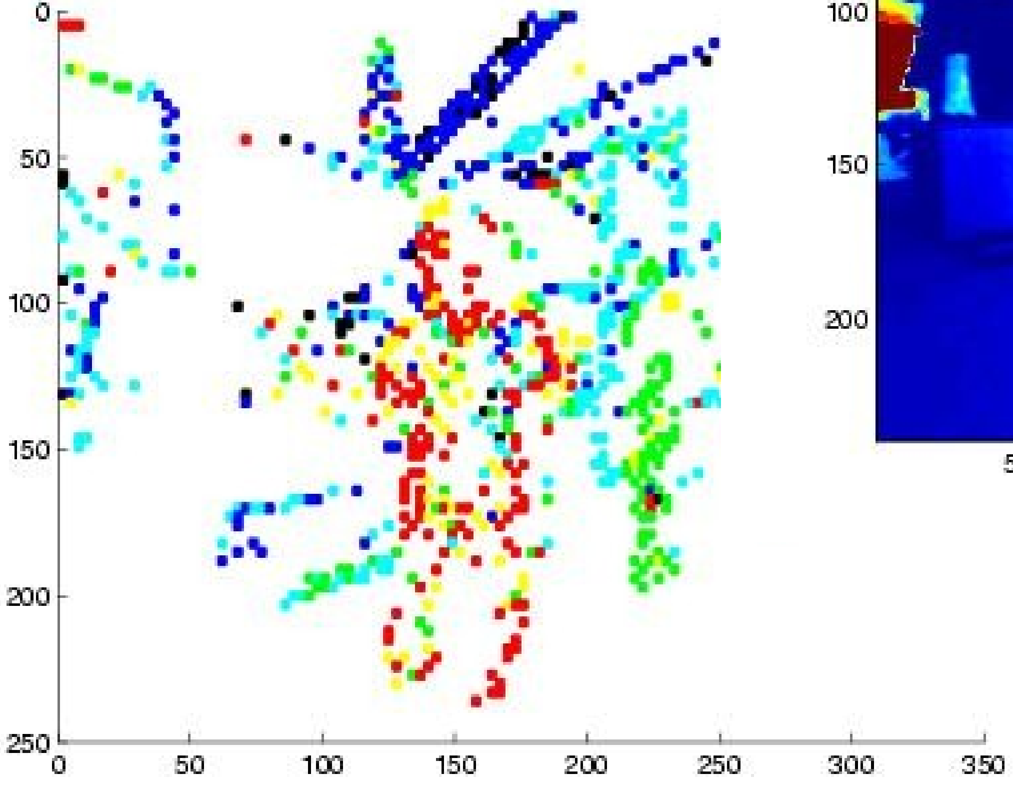
5	5	5
5	5	5
5	5	4

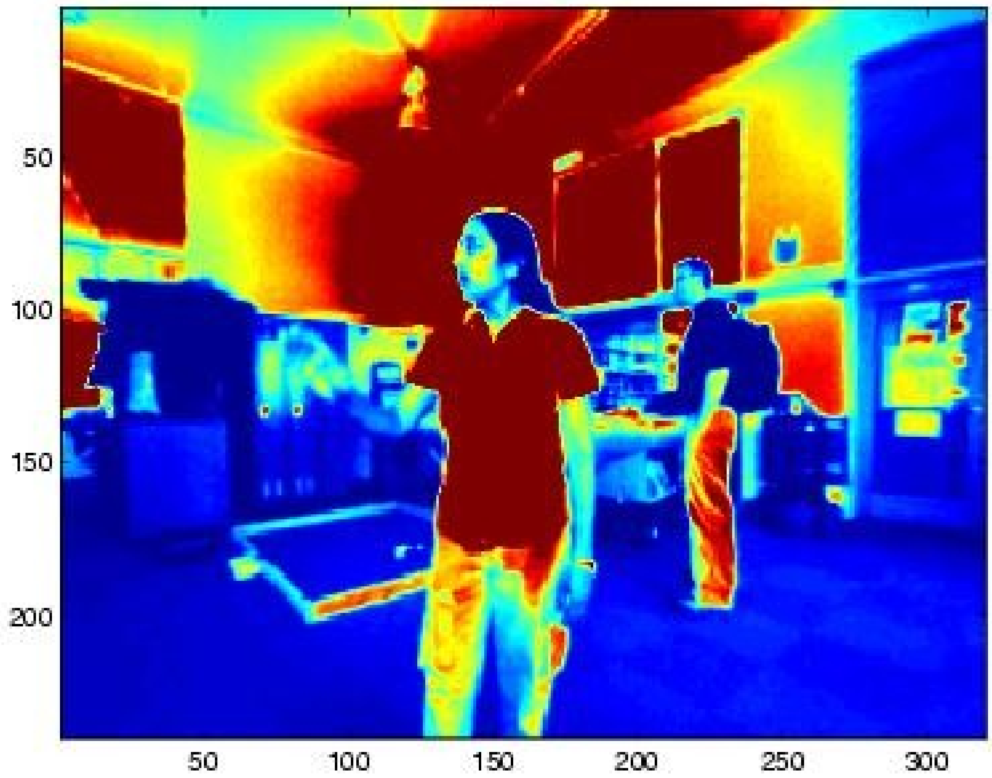
lowest eigenvalue = 0

8	5	2
5	1	5
5	5	4

lowest eigenvalue = 2.5

Stereo Vision



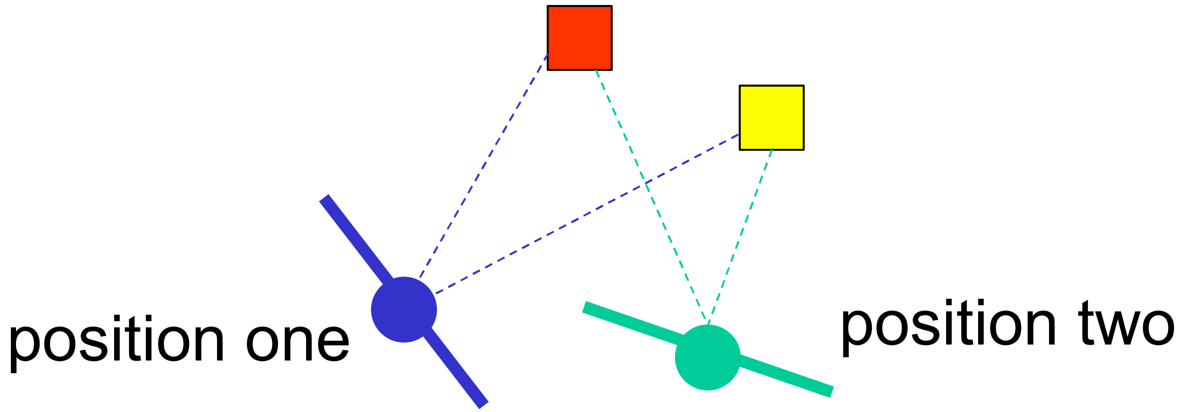


Stereo Vision

- For Maslab, the problem is simpler… can easily identify objects and compute horizontal disparity.
- To convert disparity to distance, calibrate the trig.
- Use two cameras… or mount a camera on a movable platform… or move your robot

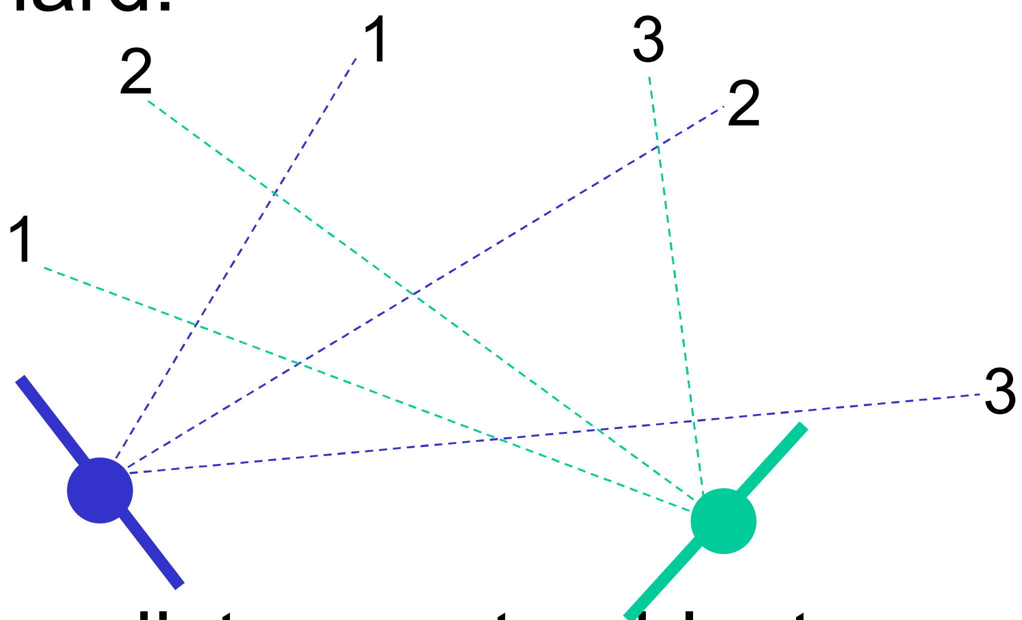
Rigid Body Motion

- Going from data association to motion
- Given
  - a starting x1,y1,θ1
  - a set of objects visible in both images
- What is x2, y2, and θ2?



Rigid Body Motion

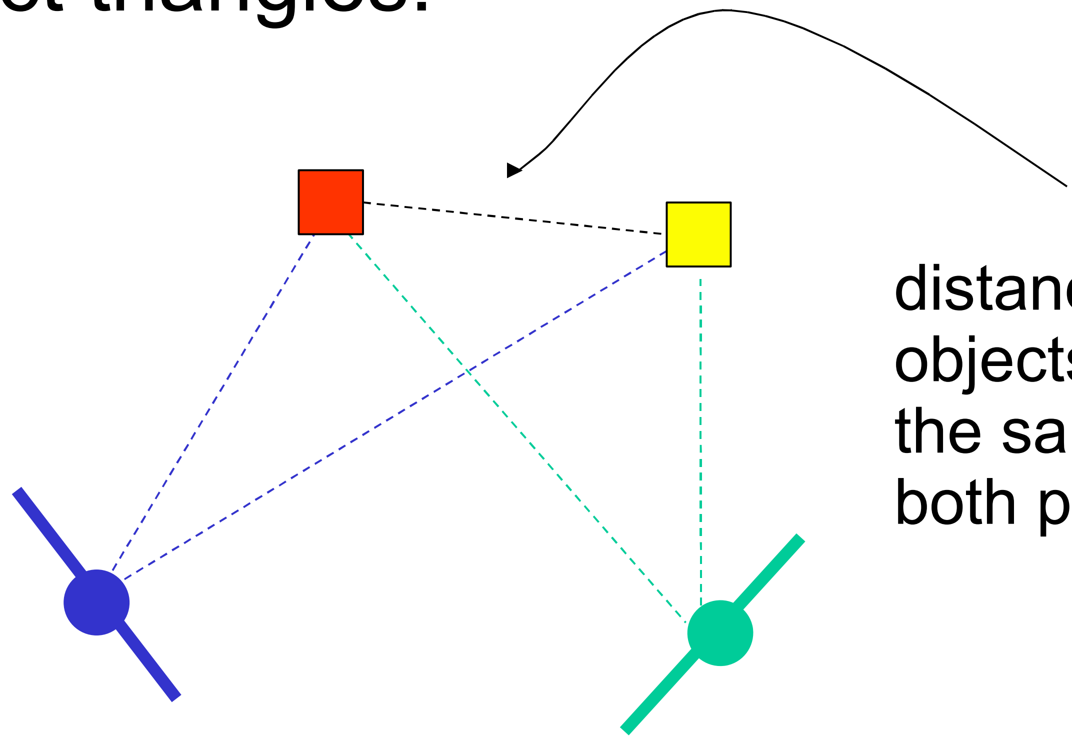
- If we only know angles, the problem is quite hard:



- Assume distances to objects are known.

Rigid Body Motion

- If angles and distances are known, we can construct triangles:



distance between objects should be the same from both positions

Rigid Body Motion

- Apply the math for a rotation:

  $x_{1i} = \cos(\theta)*x_{2i} + \sin(\theta)*y_{2i} + x_0$

  $y_{1i} = \cos(\theta)*y_{2i} - \sin(\theta)*x_{2i} + y_0$
- Solve for x_0, y_0, and θ with least squares:

  $\Sigma\ (x_{1i} - \cos(\theta)*x_{2i} - \sin(\theta)*y_{2i} - x_0)^2 +$

  $(y_{1i} - \cos(\theta)*y_{2i} + \sin(\theta)*x_{2i} - y_0)^2$
- Need at least two objects to solve

Rigid Body Motion

- Advantages
  - Relies on the world, not on odometry
  - Can use many or few associations
- Disadvantage
  - Can take time to compute

Your job for today

- Finish yesterday’s activities
- Read a barcode
- Work on tomorrow’s check point: turn until you see a ball