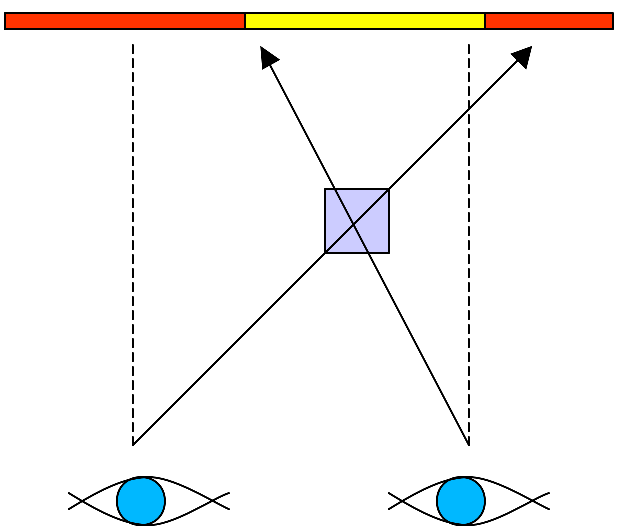
January 10, 2005

Agenda

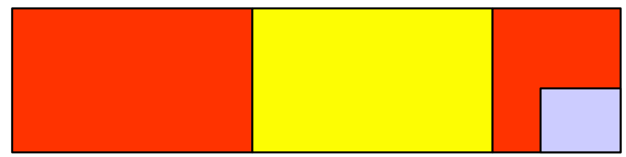
- Hodge Podge of Vision Stuff
  - Stereo Vision
  - Rigid body motion
  - Edge Detection
  - Optical Flow
  - EM Algorithm to locate objects
- May not be directly applicable, but we've tried to make it relevant.

Stereo Vision

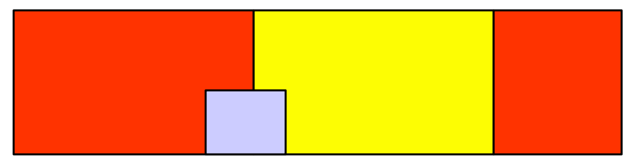
- We can judge distance based on the how much the object's position changes.



Left Eye   Right Eye



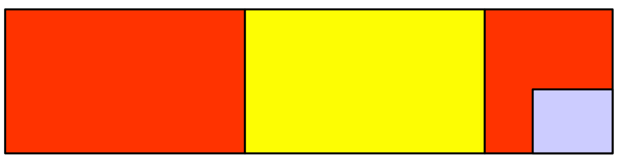
Left Image



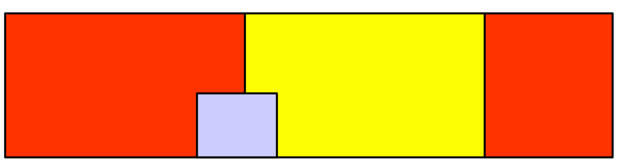
Right Image

Stereo Vision

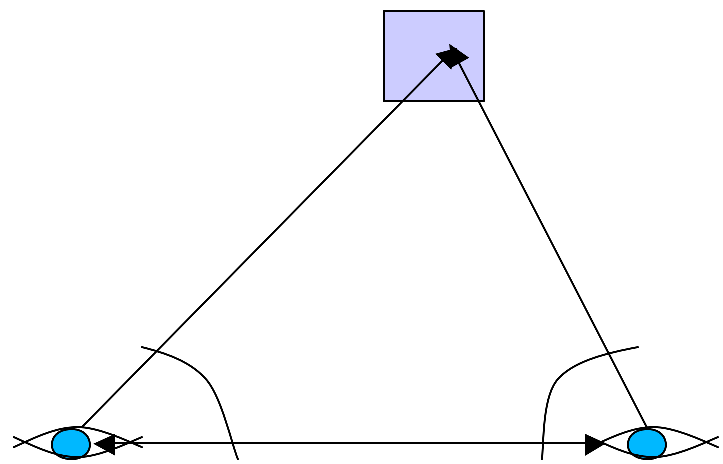
- Use the image to find the angle to the object, then apply some trig:



Left Image



Right Image

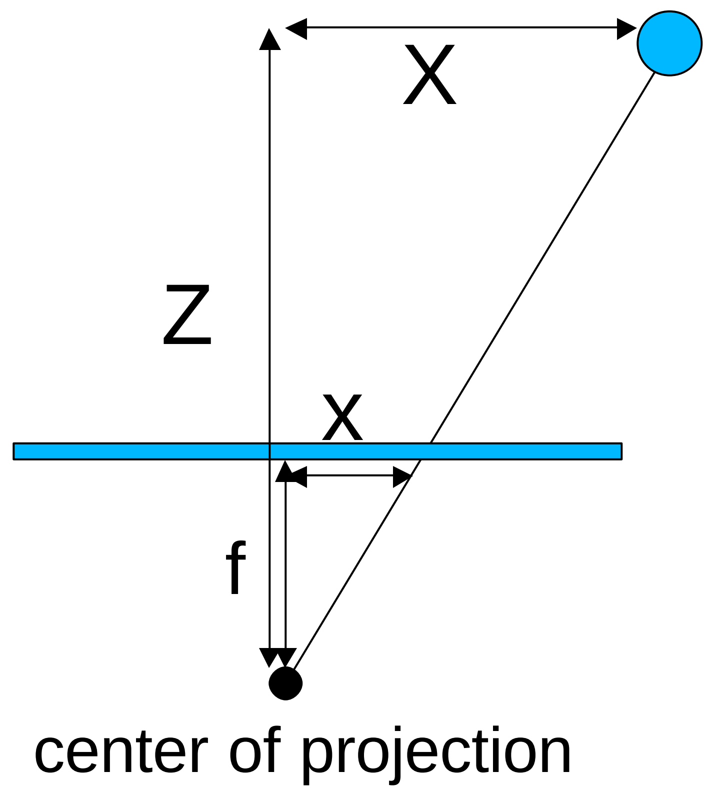


angle-side-angle gives you a unique triangle

Stereo Vision

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

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



Stereo Vision

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

Stereo Vision

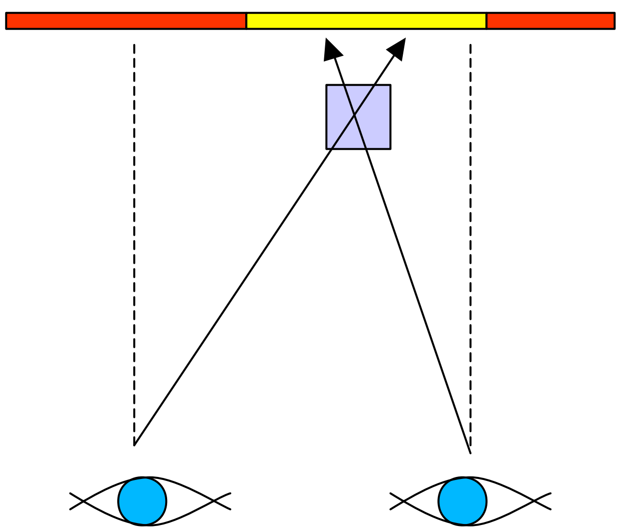
- Difference metric = Sum of $(L_i – R_i)^2$
- Search horizontally for best match (least difference)

6	5	5
5	6	5
5	5	7

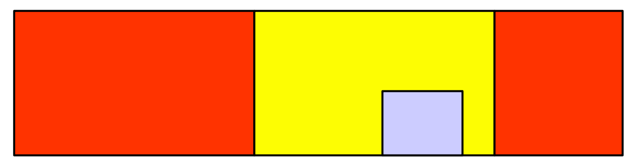
6	5	5
5	6	5
1	1	6

Stereo Vision

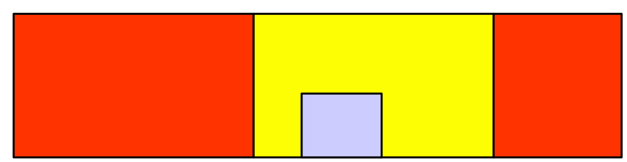
- Still have a problem: unless the object is really close, the change might be small…



Left Eye Right Eye



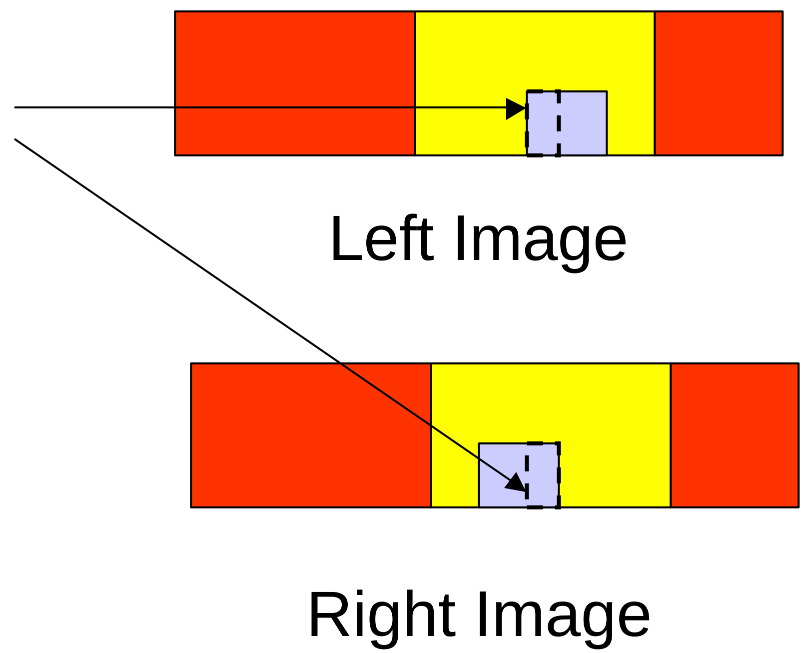
Left Image



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.

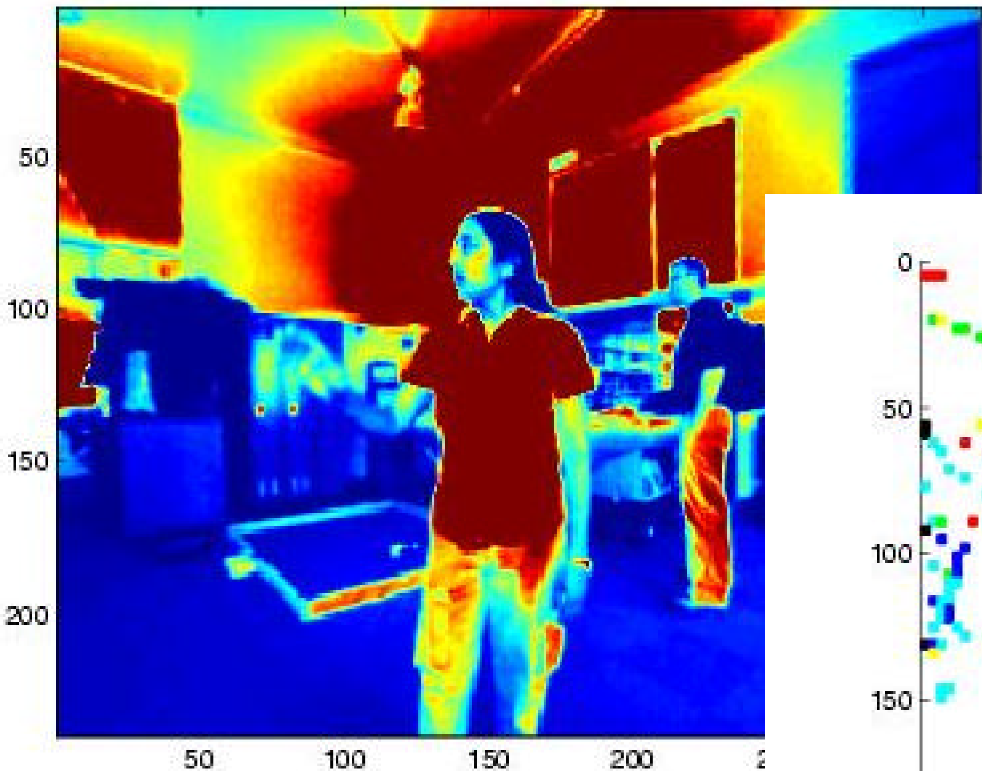
5	5	5
5	5	5
5	5	4

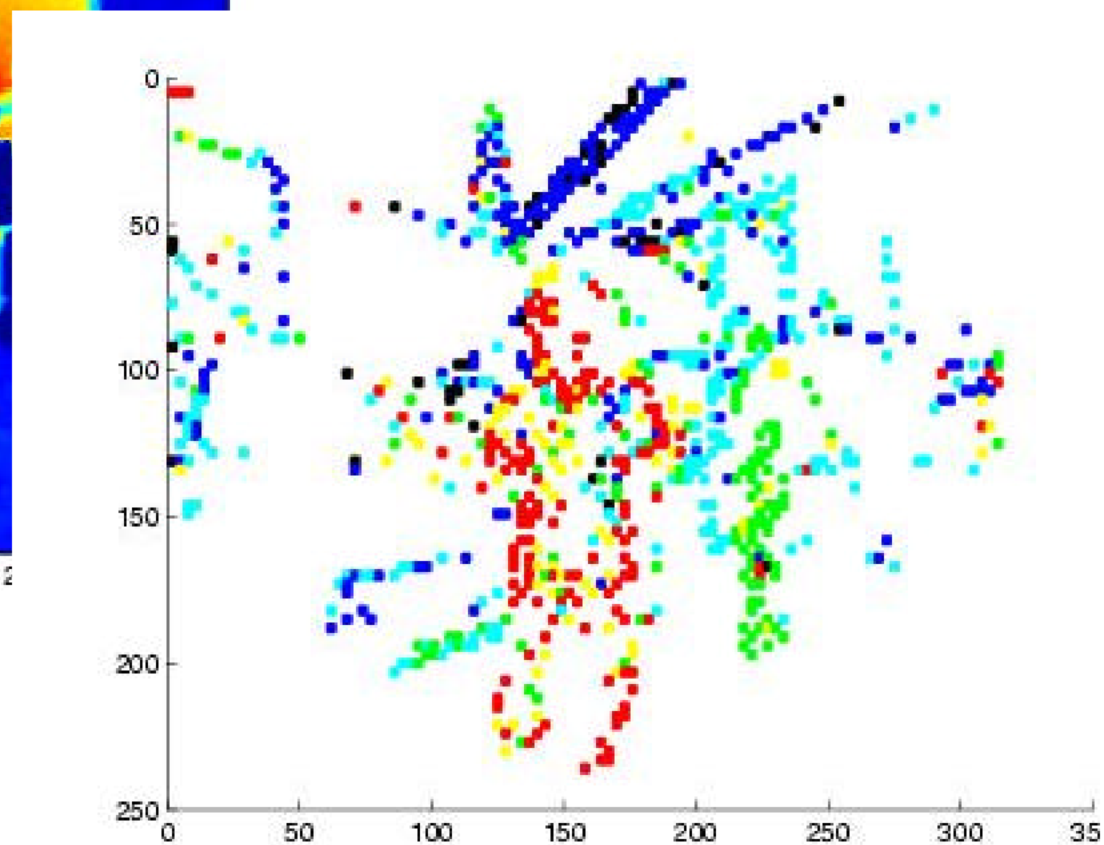
lowest eigenvalue = 0

8	5	2
5	1	5
5	5	4

lowest eigenvalue = 2.5

Stereo Vision



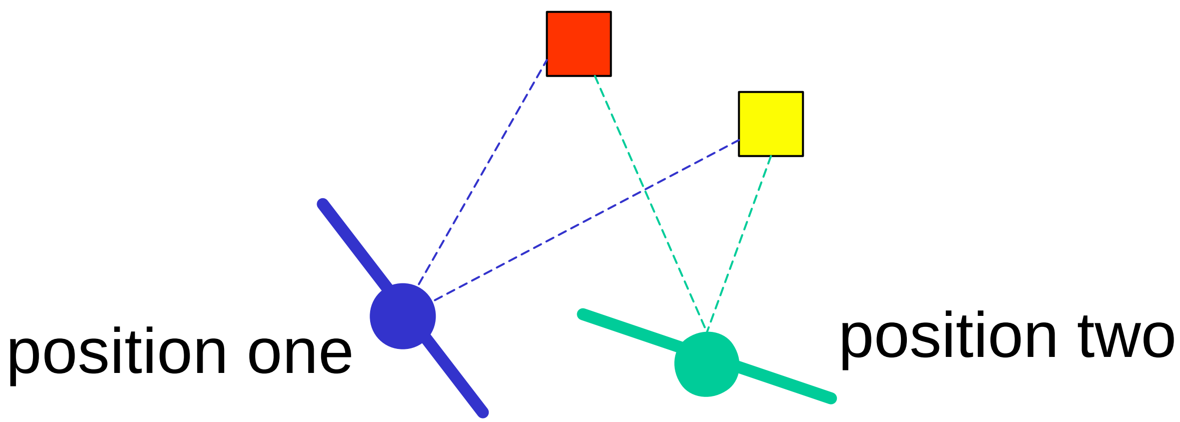


Stereo Vision

- 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

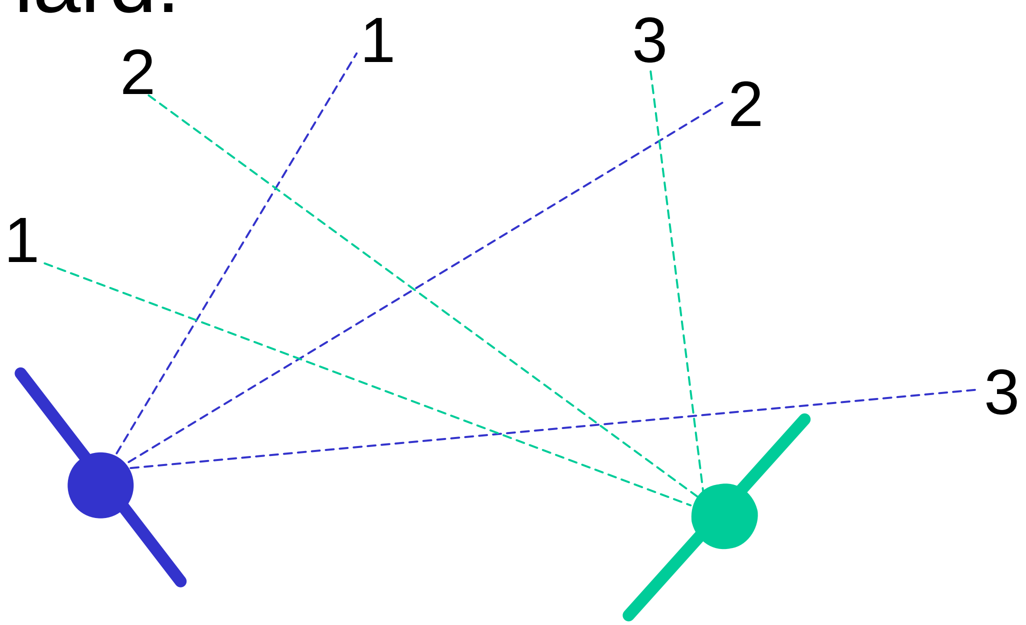
Rigid Body Motion

- Going from data association to motion
- Given
  - a starting x1,y1,?1
  - a set of objects visible in both images
- What is x2, y2, and ?2?



Rigid Body Motion

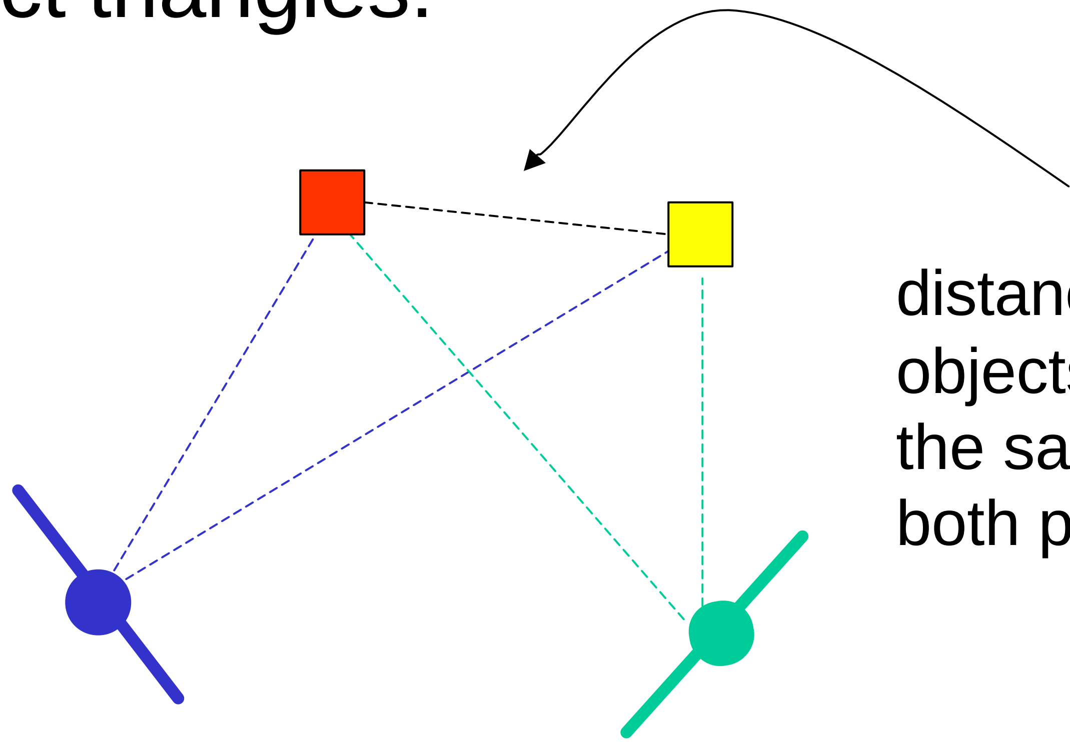
- If we only know angles, the problem is quite hard:



- 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(?)*x_{2i} + sin(?)*y_{2i} + x_0

  y_{1i} = cos(?)*y_{2i} - sin(?)*x_{2i} + y_0

- Solve for x_0, y_0, and ? with least squares:

  S (x_{1i} - cos(?)*x_{2i} - sin(?)*y_{2i} - x_0)^2 +

  (y_{1i} - cos(?) *y_{2i} + sin(?)*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

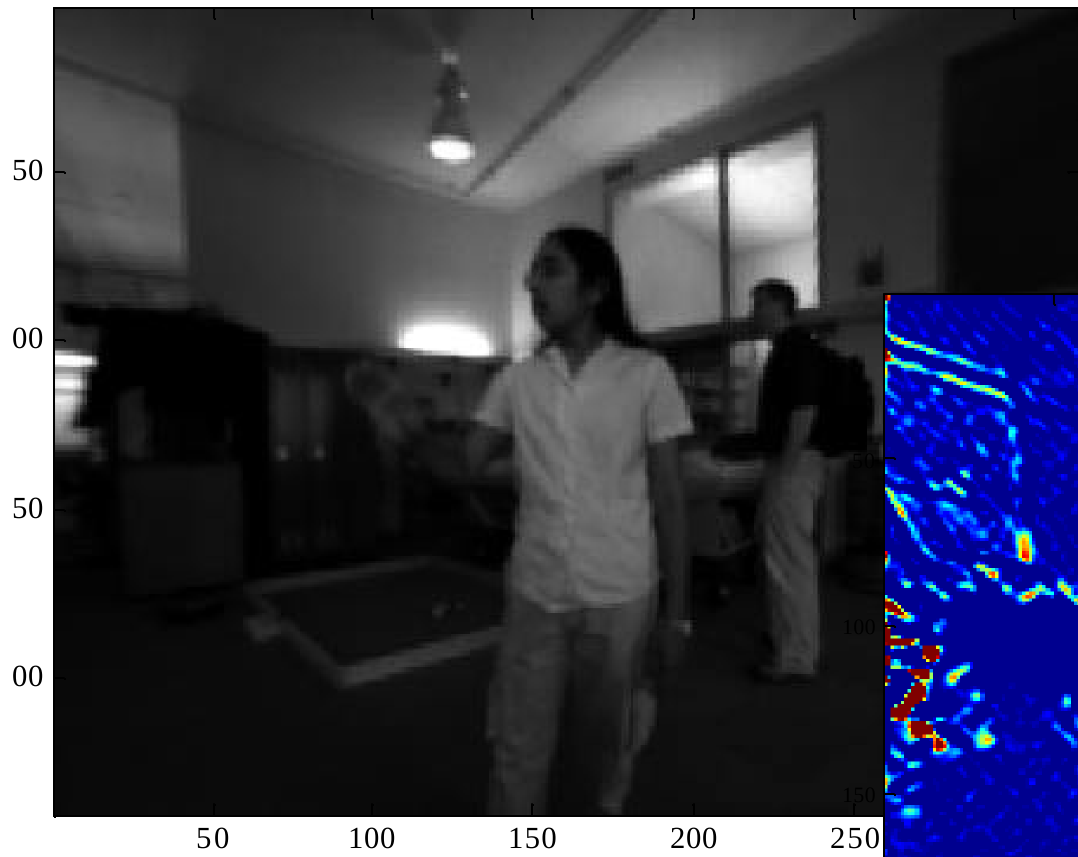
Edge Detection

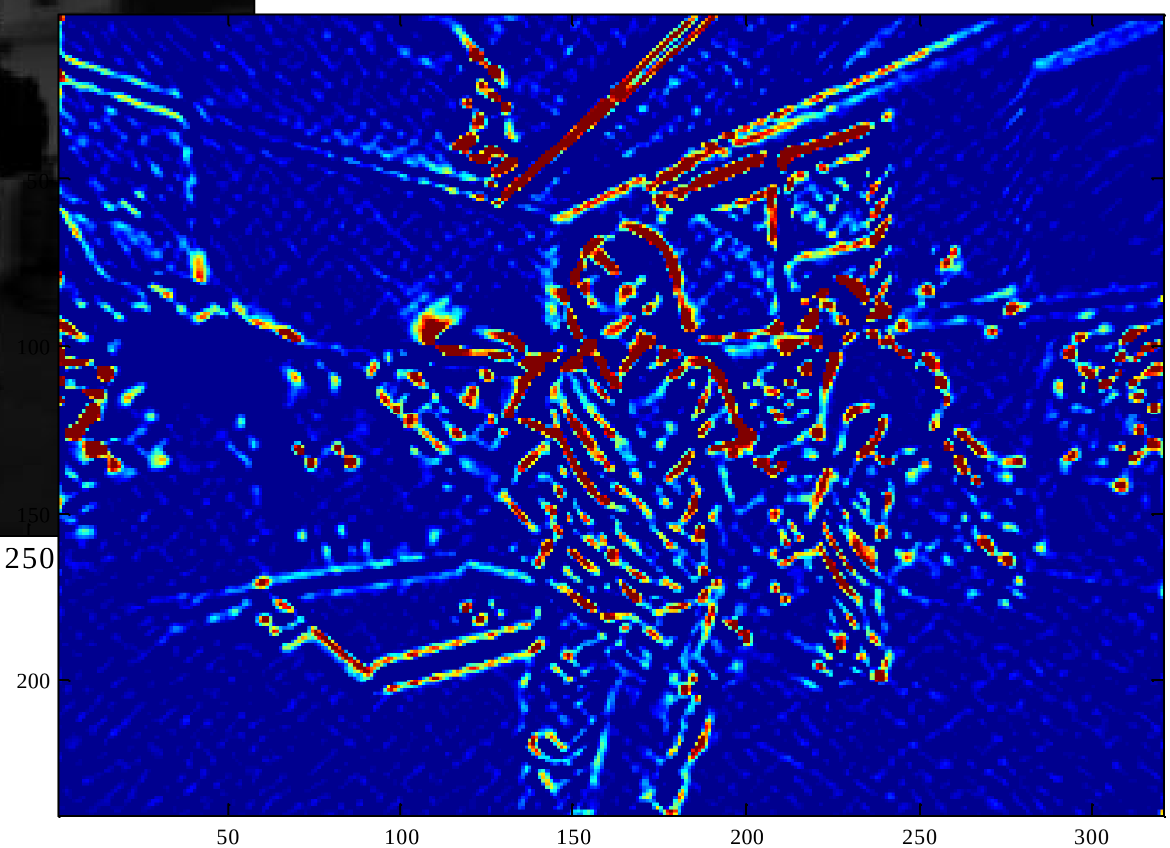
- Edges are places of large change
- Scan the image with little computational molecules or a ‘kernel’

1
0
-1

1	0	-1

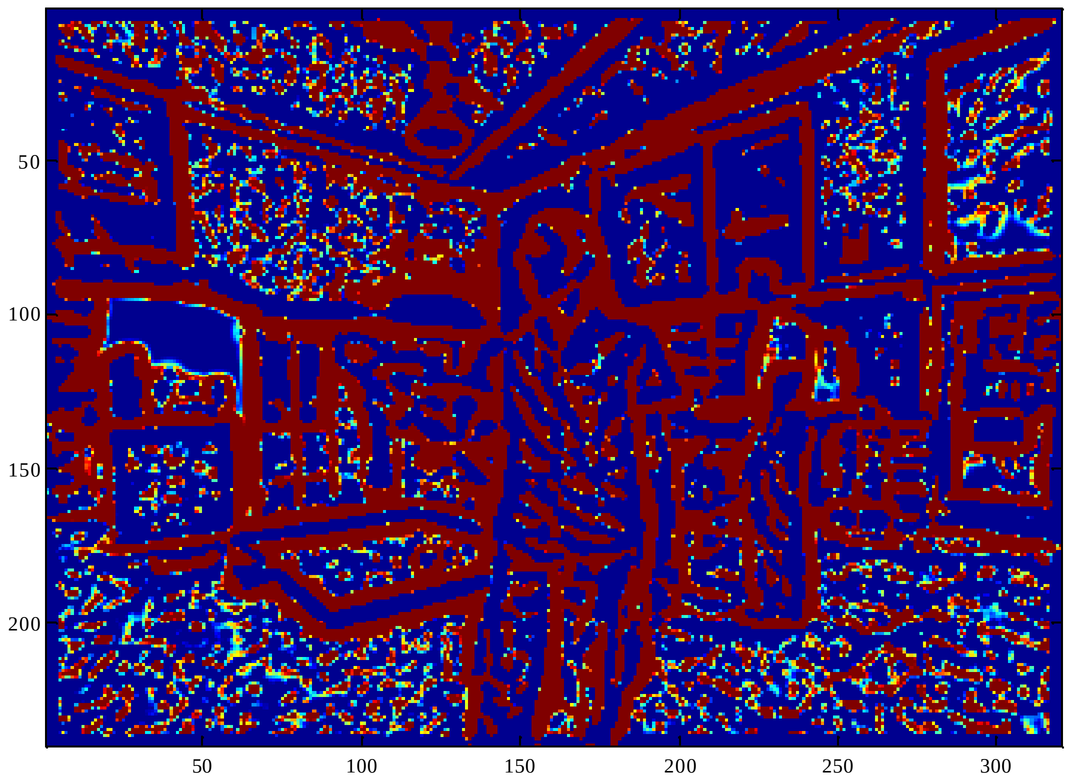
Edge Detection





Edge Detection

- More sophisticated filters work better (Laplacian of Gaussian, for example)



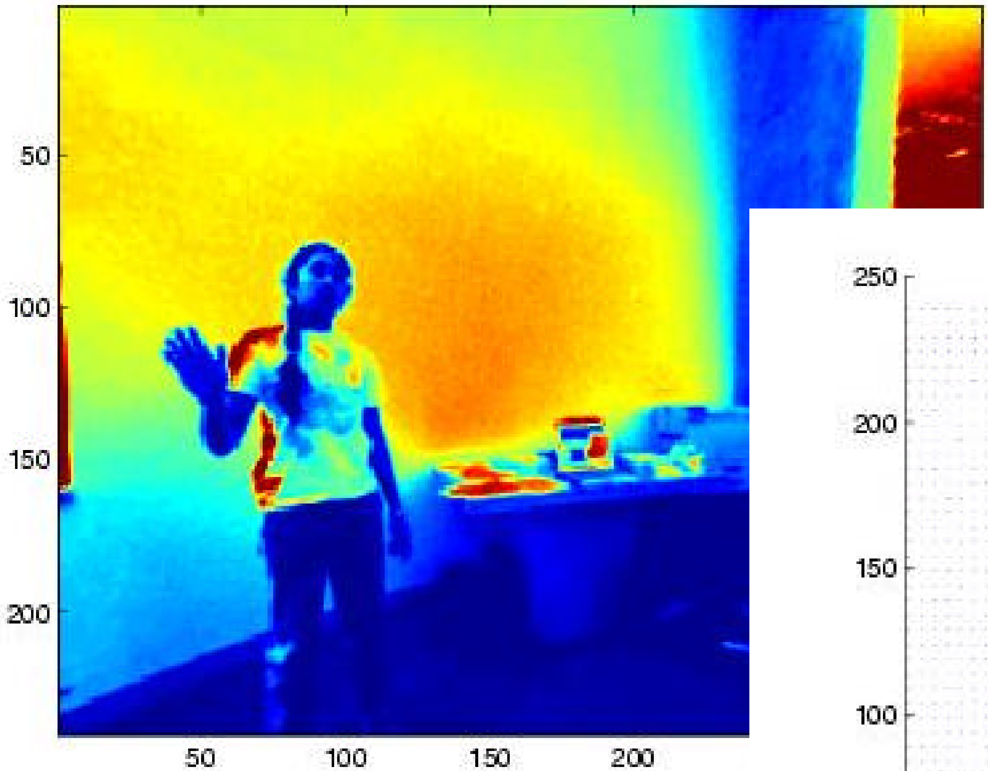
Edge Detection

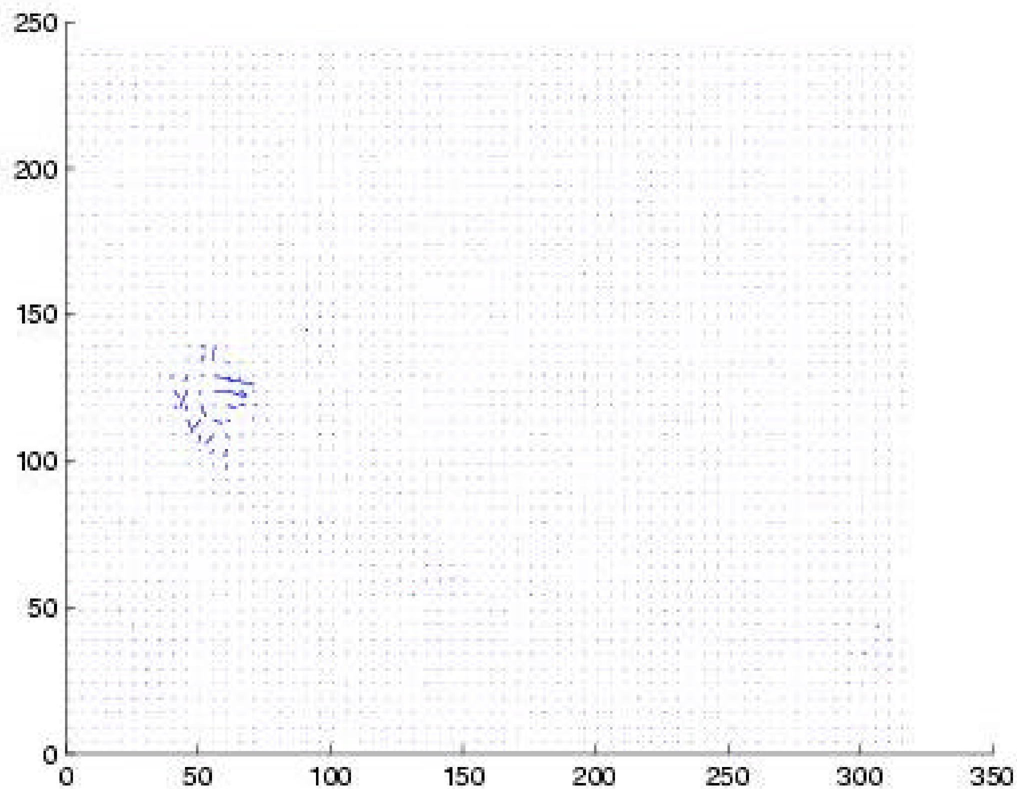
- Need to choose a good value for threshold
  - Too small—gets lots of noise, fat edges
  - Too big—lose sections of edge
- What do you do with an edge?
  - Extract lines for a map?
  - Use to separate regions?

Optical Flow

- Look at changes between successive images
  - identify moving objects
  - identify robot motion (flow will radiate out from direction of motion)
- For each point on image, set total derivative of brightness change to zero:
  - 0 = u*Ex + v*Ey + Et

Optical flow





Optical Flow

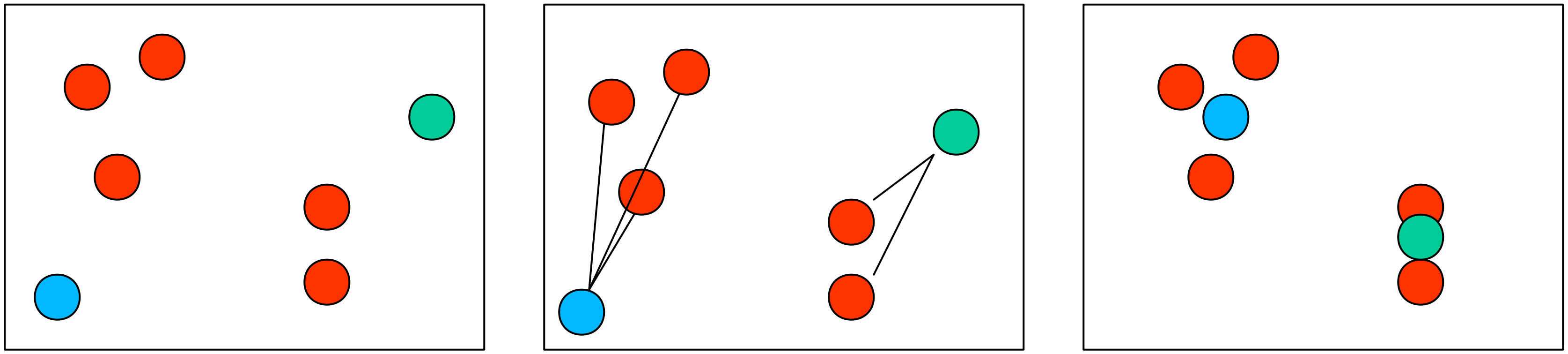
- Computationally expensive and requires very fast frame rates… or very slow robots
- Idea from optical flow: looking at change between frames can help segment an image (only edges will move).

EM Algorithm

- Given an image with k objects
- How can we find their locations?

EM Algorithm

- 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 Algorithm

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

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

Reminders

- No lecture tomorrow
- Design Review Wednesday
- Check Point Two: Friday

- If you haven't completed check point one, you finish it today!