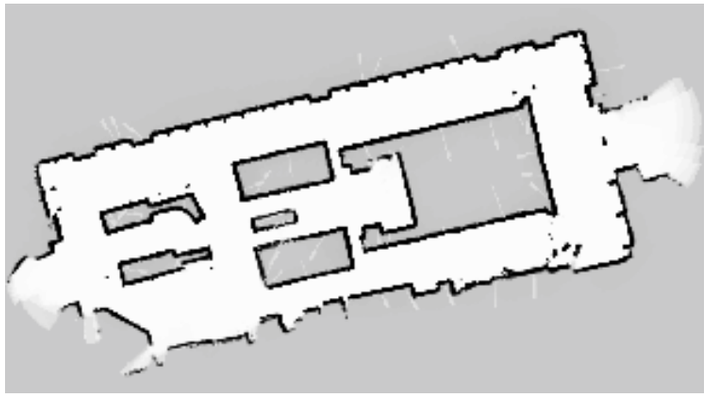
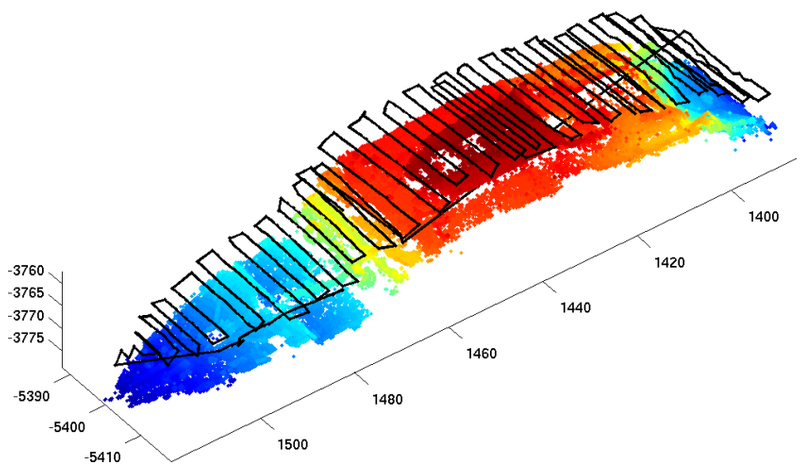
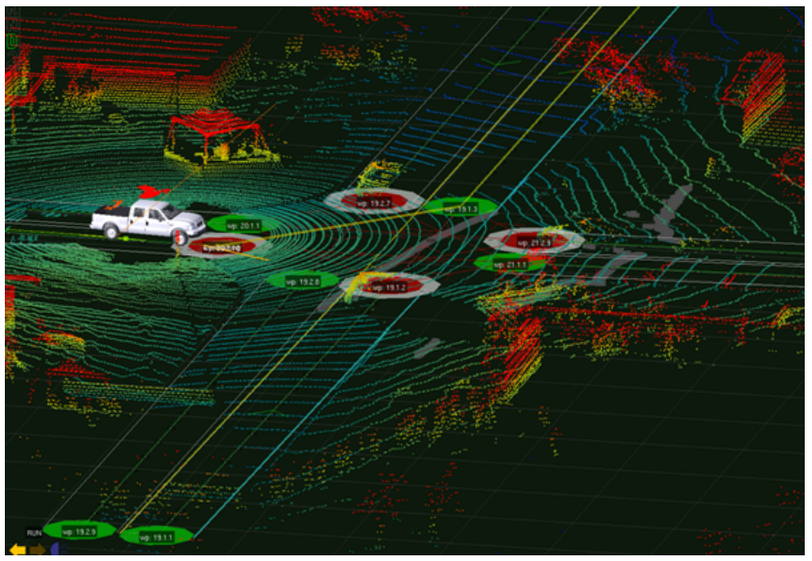






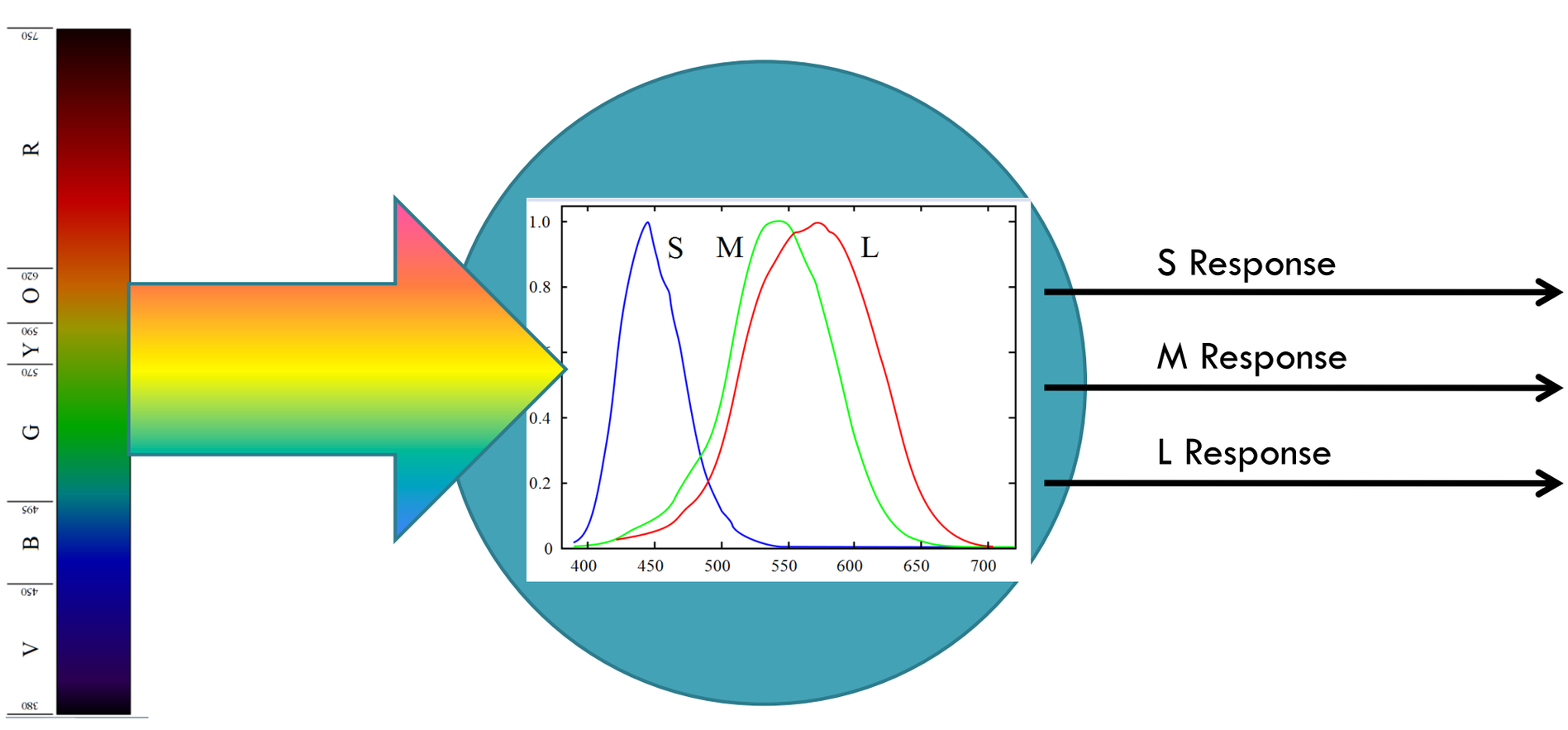
EECS568 Mobile Robotics: Methods and Principles

Prof. Edwin Olson

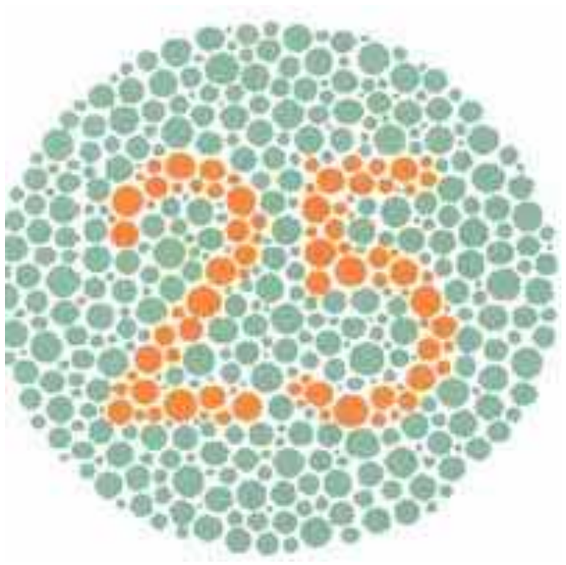
L21. Features from camera data

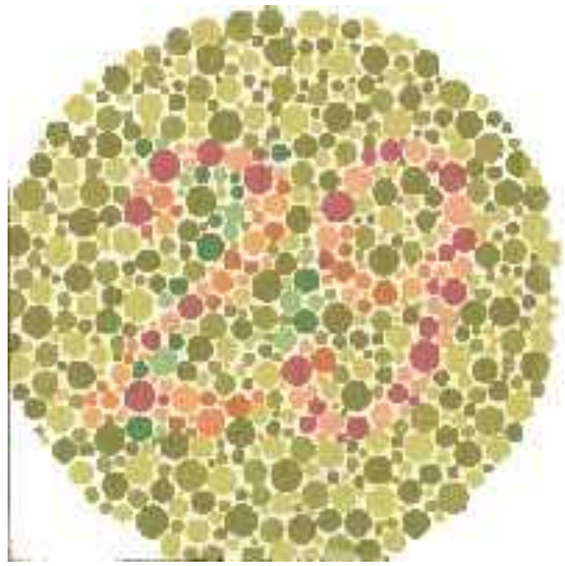
Color Cameras

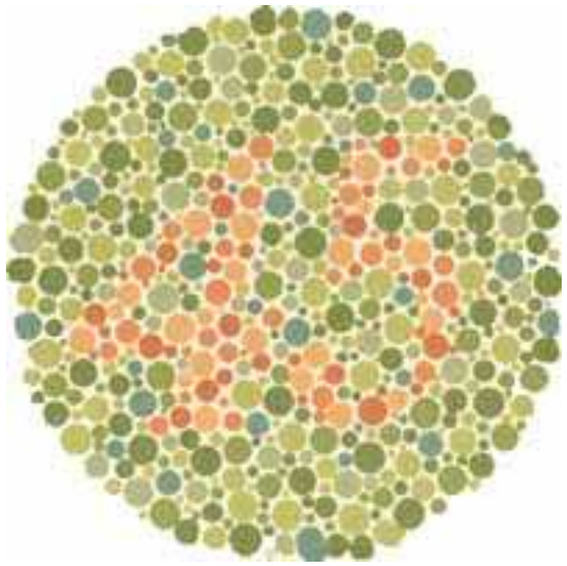
- Incoming light is described in terms of a *power spectral density*
- *"Color"* isn't a physical property of light
  - It's made up by our eyes and brain!
  - Different types of incoming light can have the same "color"

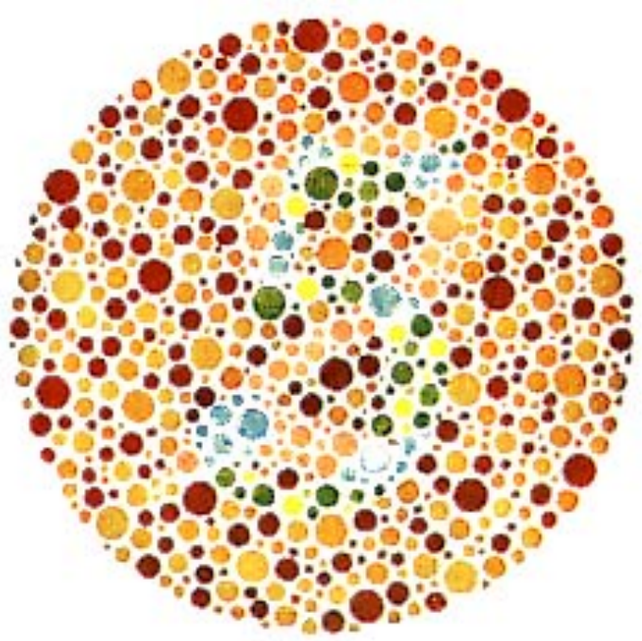


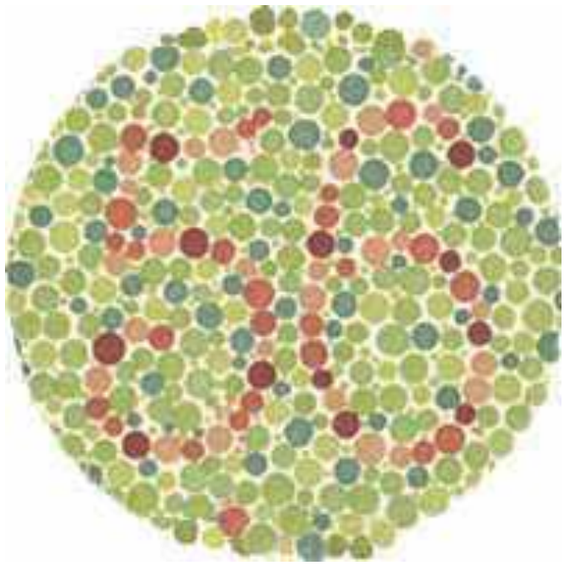
Just for fun...

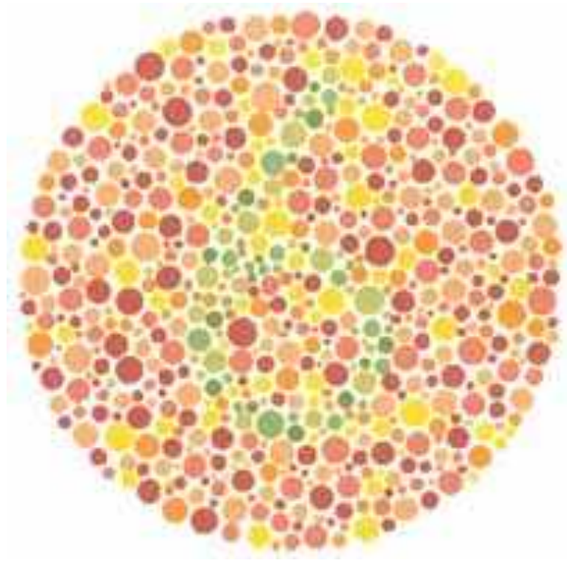


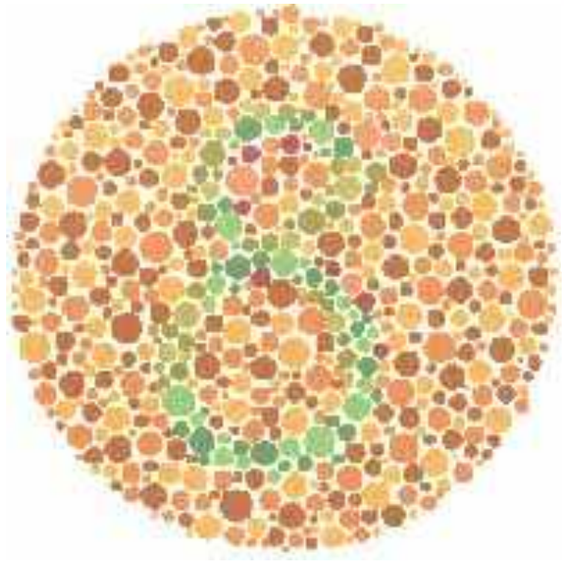




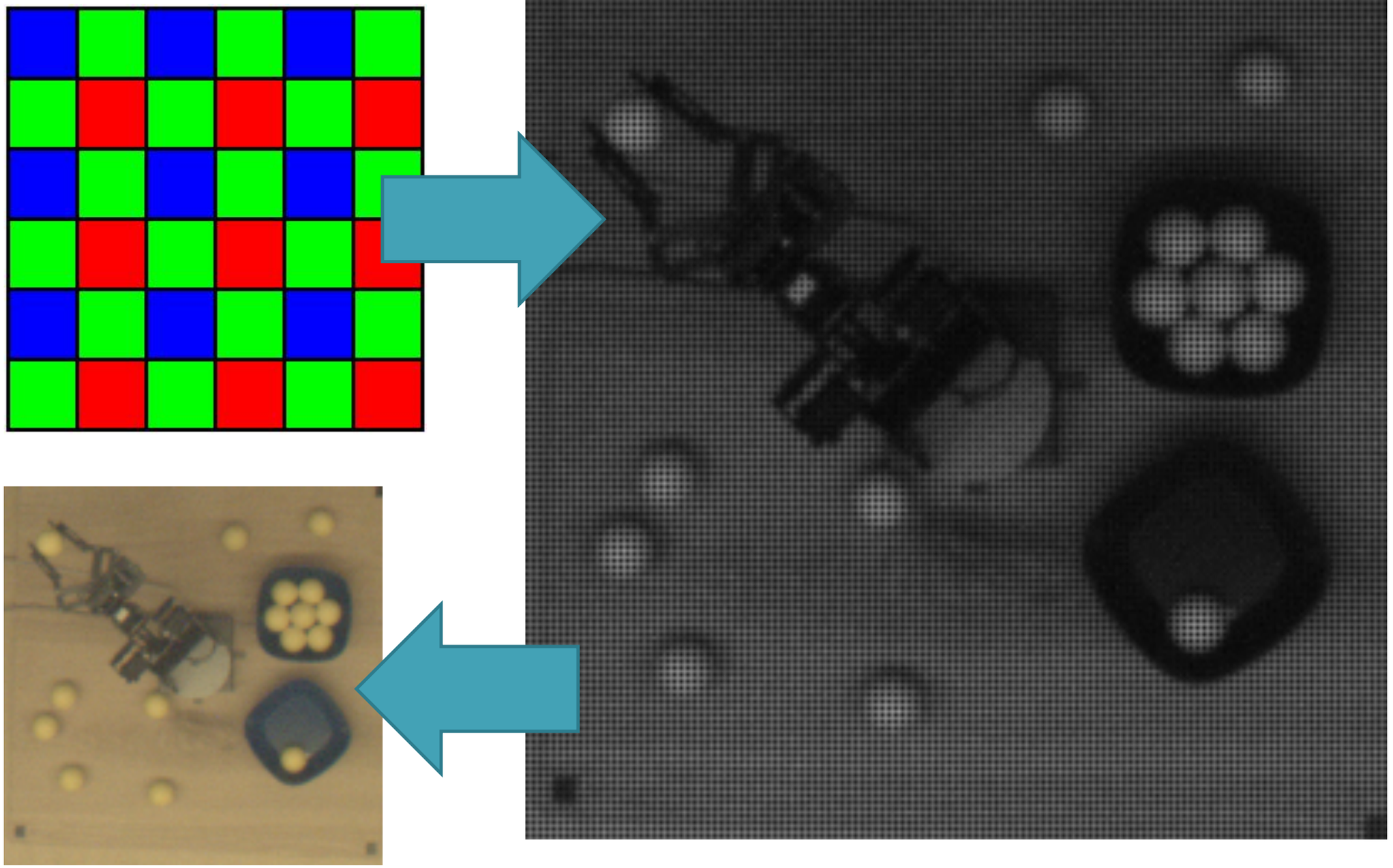






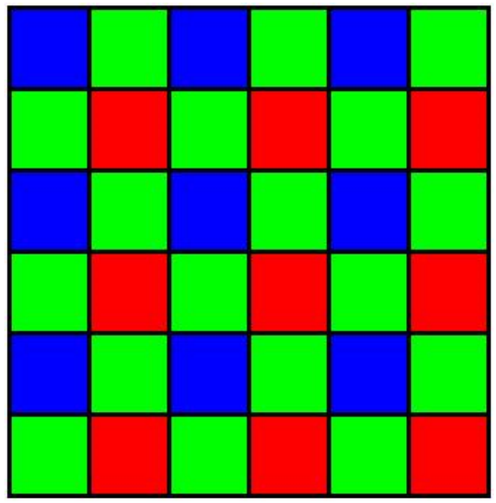


Bayer Patterns



Bayer Patterns

- Why does this matter?
  - At each pixel, two color channels are interpolated based on nearby pixels

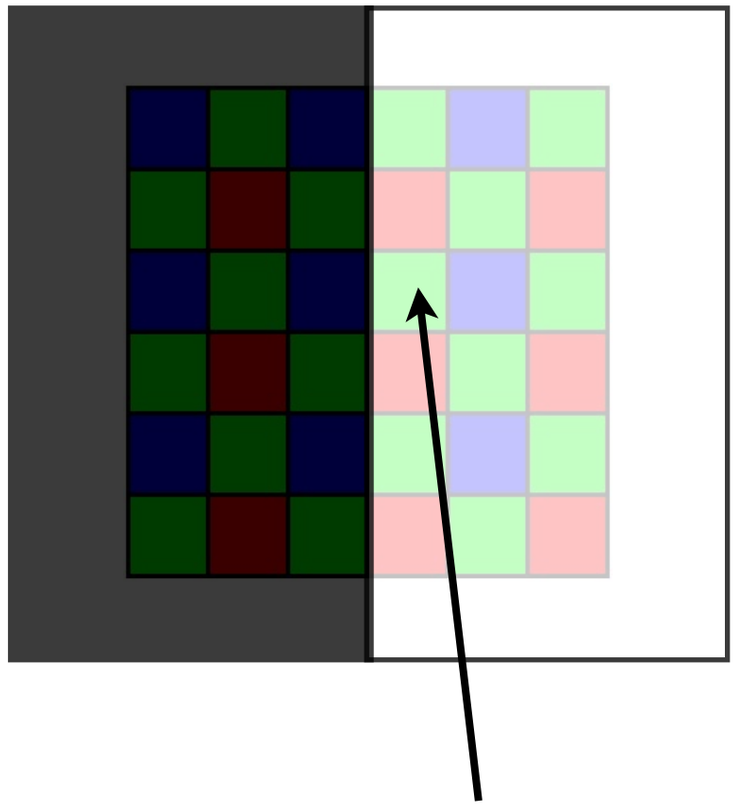


- Thus, a color camera is more blurry than a monochrome camera.
  - e.g., Monochrome cameras give slightly better results for AprilTags

Bayer Pattern Artifacts

- When the color of an area is uniform, Bayer patterns work well.

- What happens when there is a rapid change in color?
  - R, G, and B sub-pixels may observe different PSDs
  - Interpolated colors may not exist anywhere!



Average of nearby red pixels = red... so there will be a red output pixel even though the incoming light is either white or black.

Why extract features from camera images?

- Motivation: understanding images is really hard!
  - Lots of data
  - Some parts of the image are "boring"

- Idea: extract "good" features
  - From 1M pixels to 100s of features
  - Can make features robust

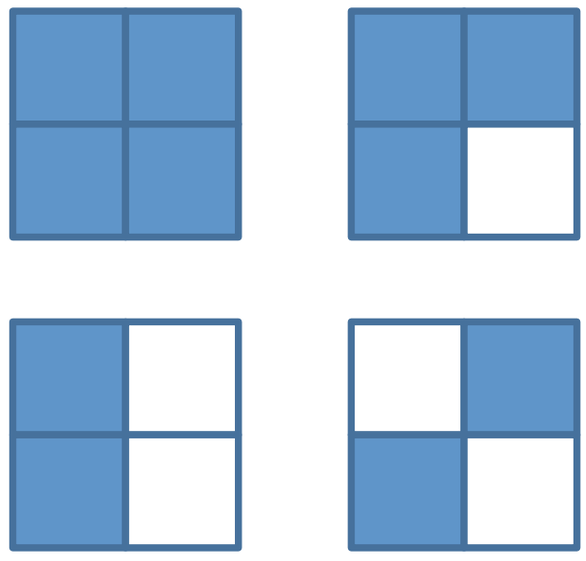
what do we mean by robust?

detect the same objects over a range of lighting, viewpoint, rotation, scale changes

Corner Detectors

- Intuitively, corners are a good feature.
  - Relatively easy to find
  - Trackable

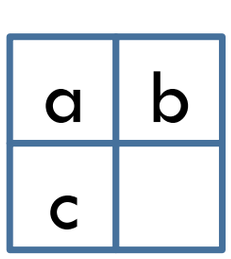
- But what is a corner?
  - We're processing from bottom-up
  - No idea (yet) about objects
    - a corner != object corner



- What isn't a corner?
  - Uniform areas
  - Edges/lines

Image Gradients

- Idea: let's look at gradients of a patch of pixels
  - Gradient at pixel a is (b-a, c-a)



- Compute gradients for 2x2 area
  - We need 3x3 input…

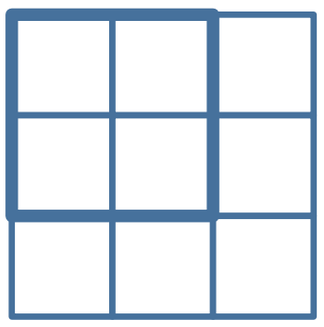


Image Gradients

- Are these good corners?
  - (What are the gradients?)

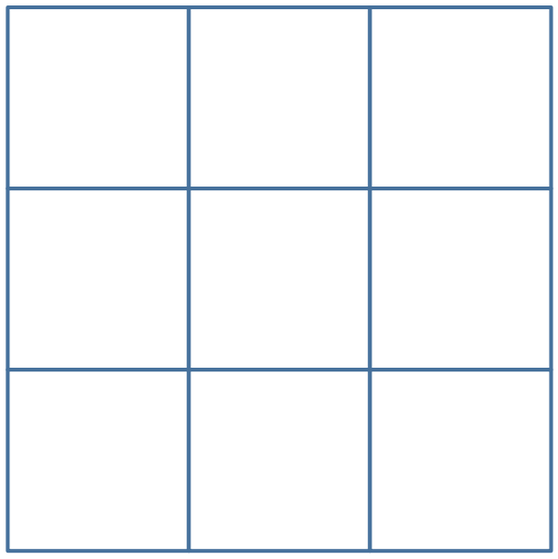


Image Gradients

- Are these good corners?
  - (What are the gradients?)

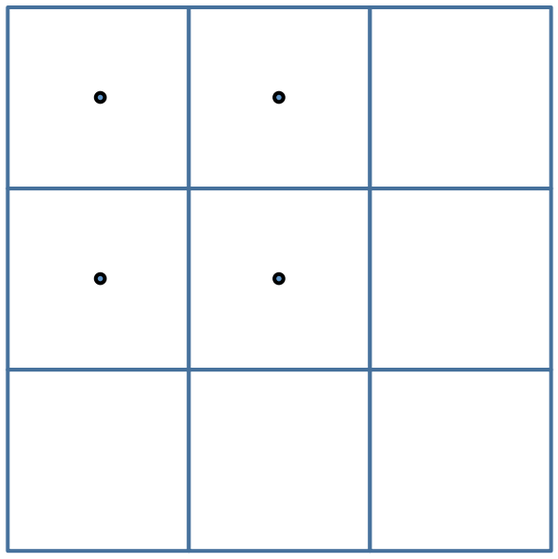
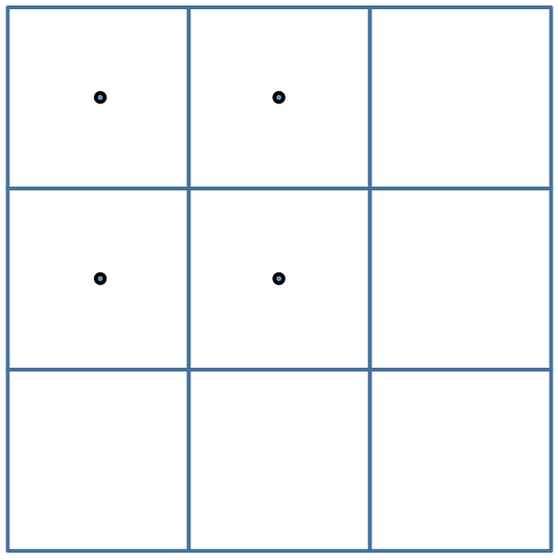


Image Gradients

- Are these good corners?
  - (What are the gradients?)



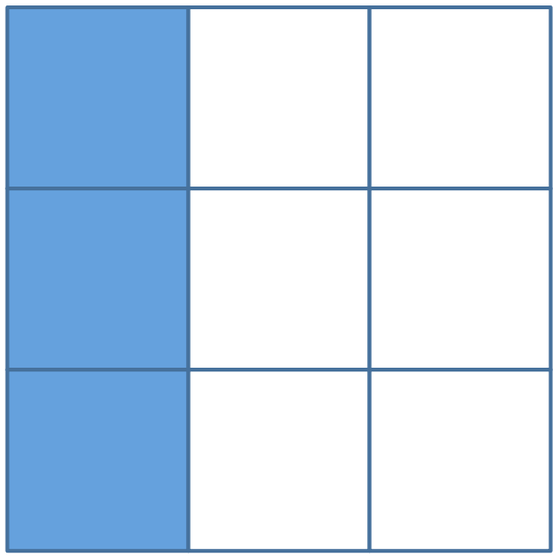
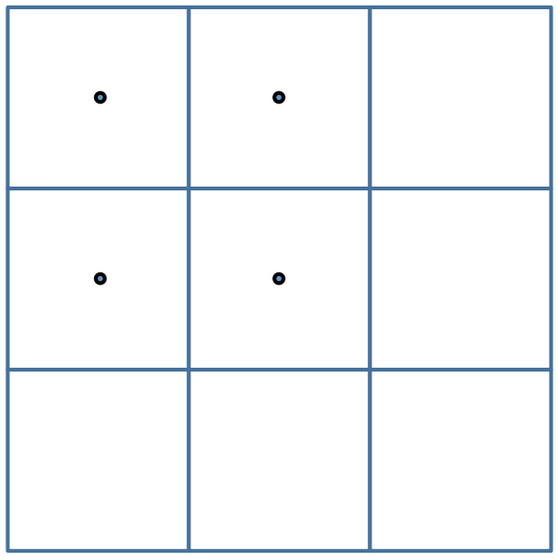


Image Gradients

- Are these good corners?
  - (What are the gradients?)



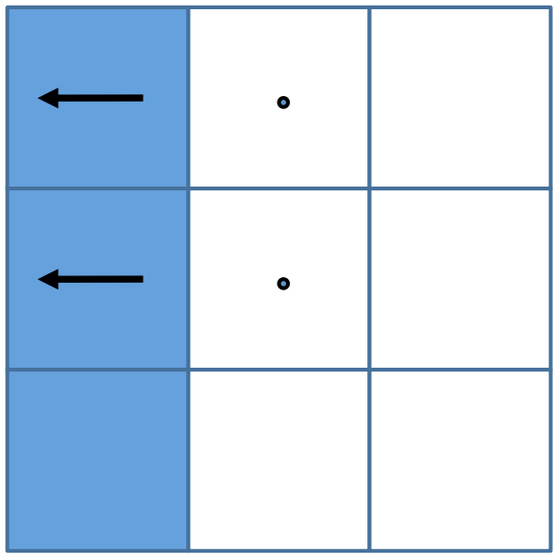
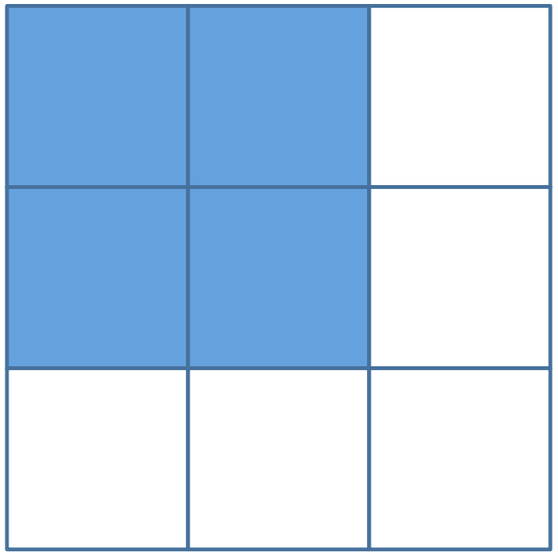
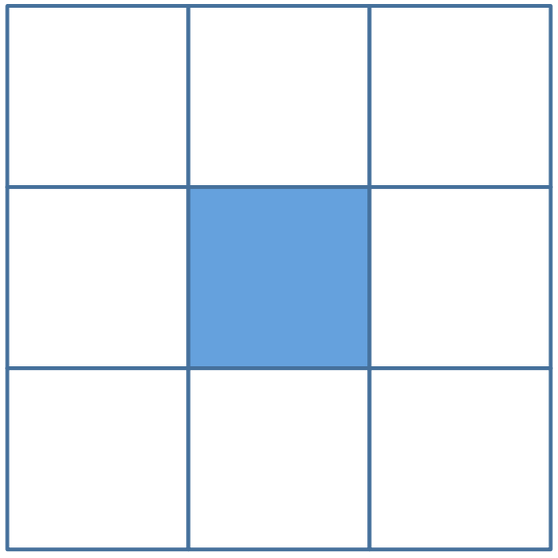


Image Gradients

- Are these good corners?
  - (What are the gradients?)





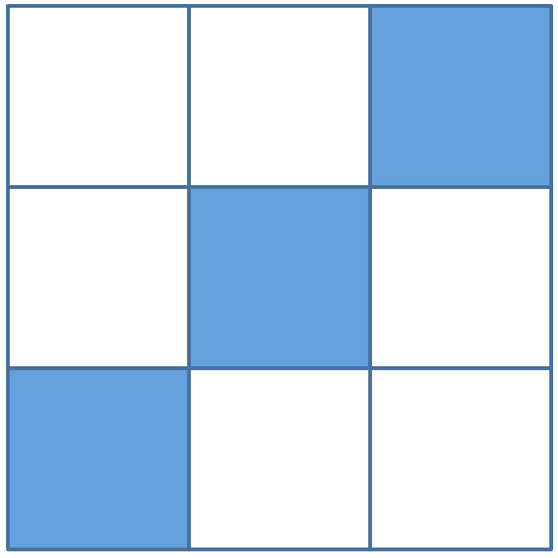
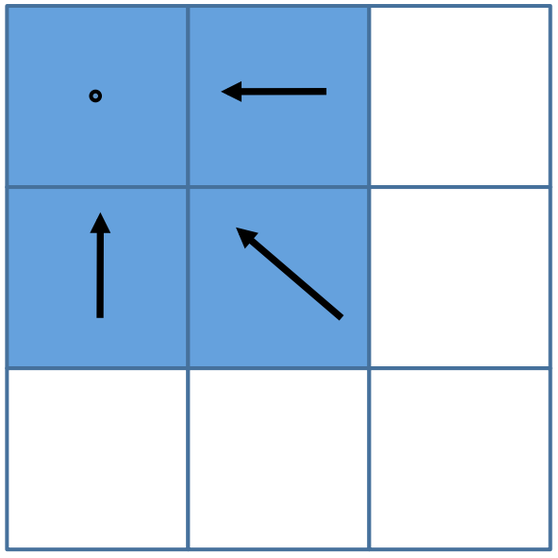
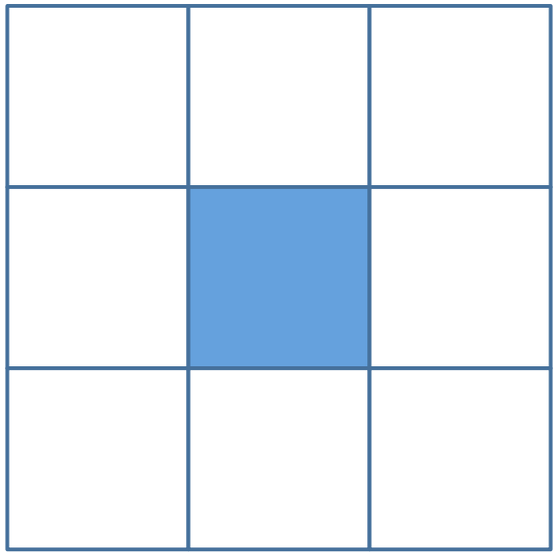


Image Gradients

- Are these good corners?
  - (What are the gradients?)





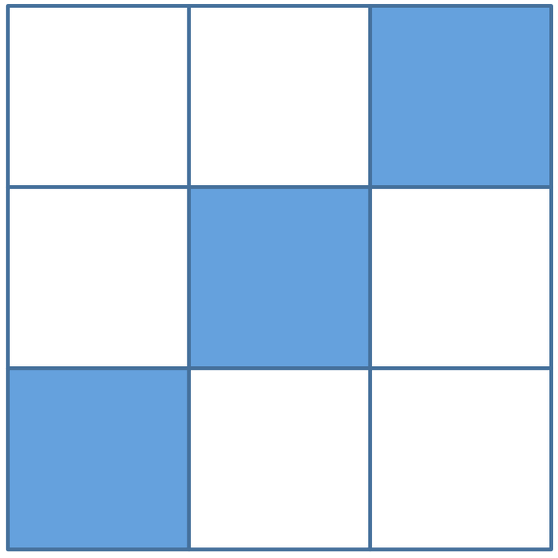
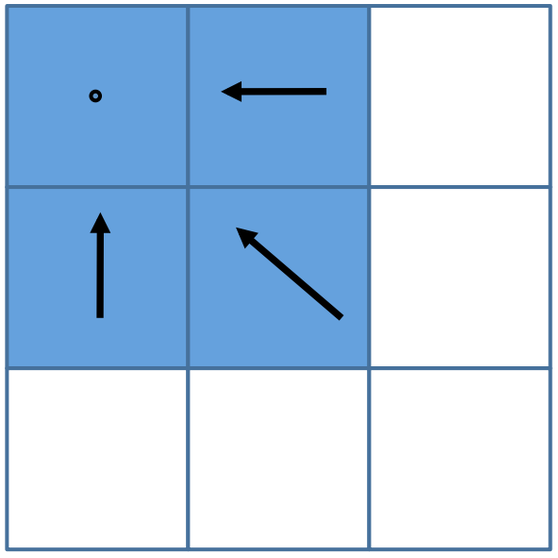
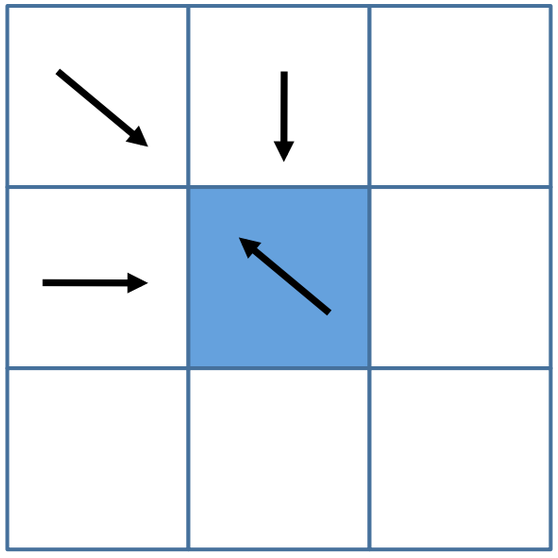


Image Gradients

- Are these good corners?
  - (What are the gradients?)





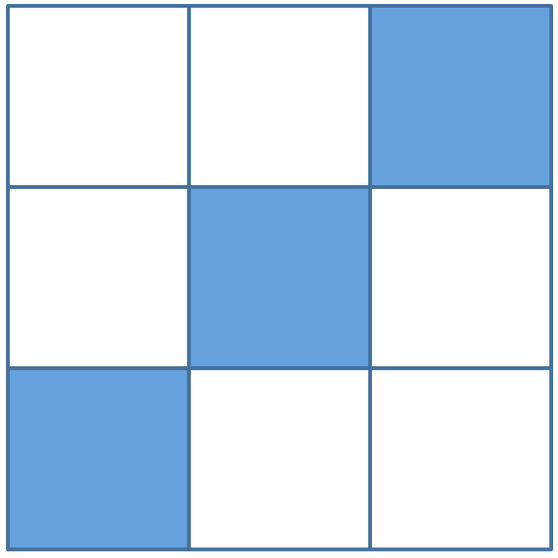
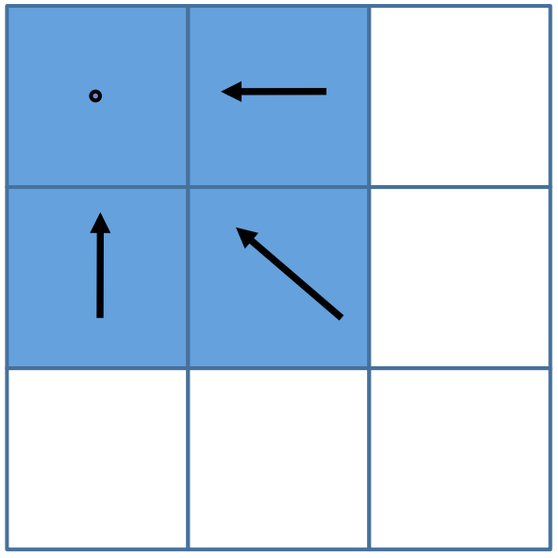
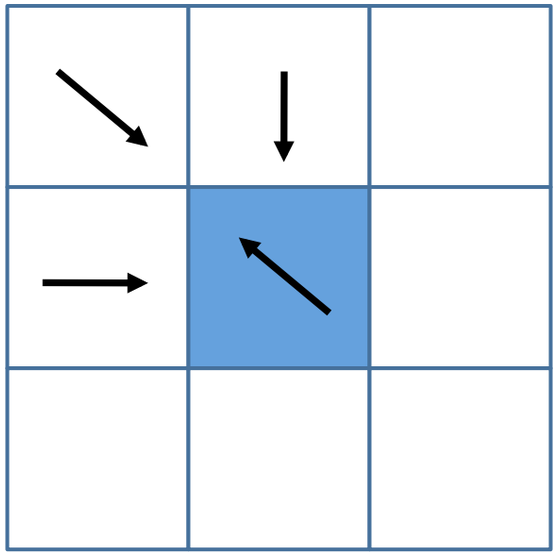
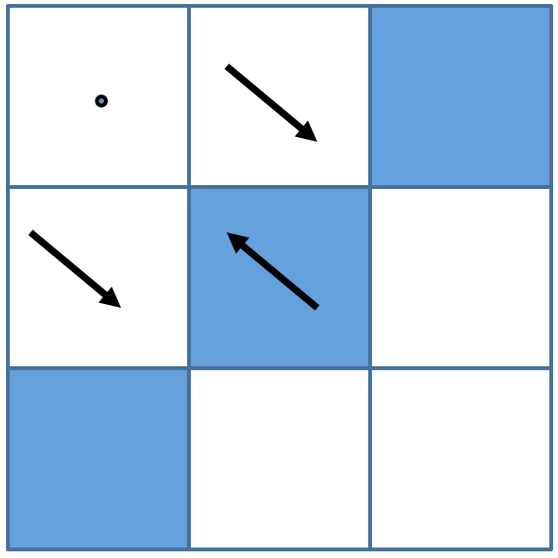


Image Gradients

- Are these good corners?
  - (What are the gradients?)

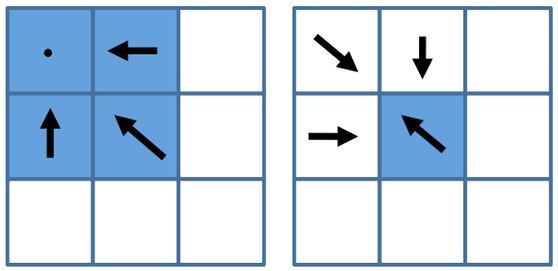




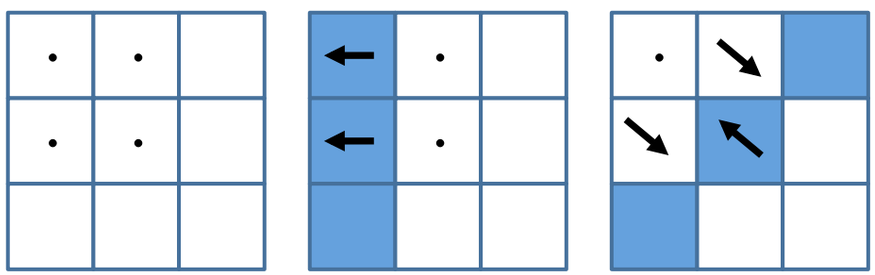


Good and Bad Corners

- Good Corners

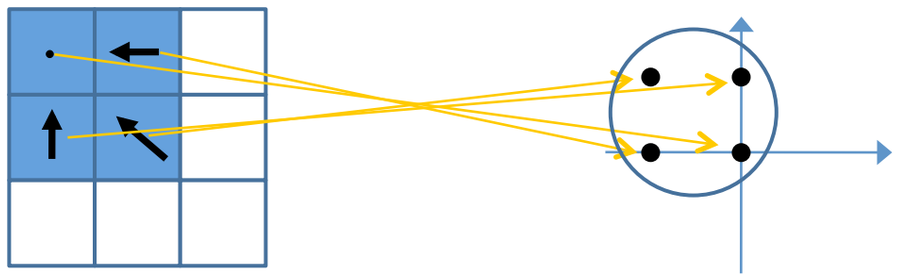


- Bad Corners

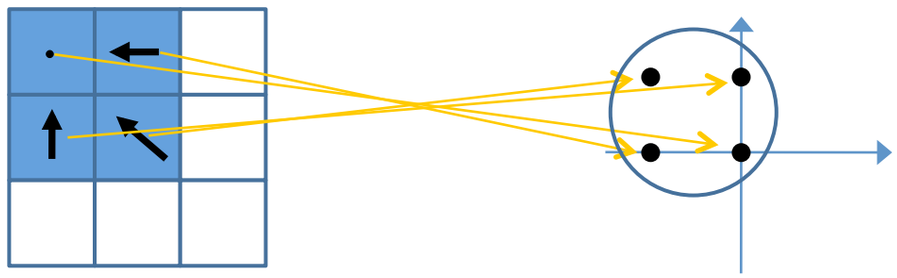


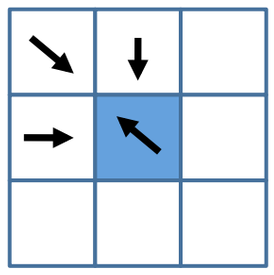
- What do good/bad corners have in common?

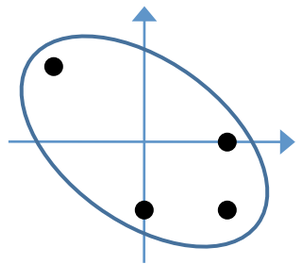
Corners in gradient space



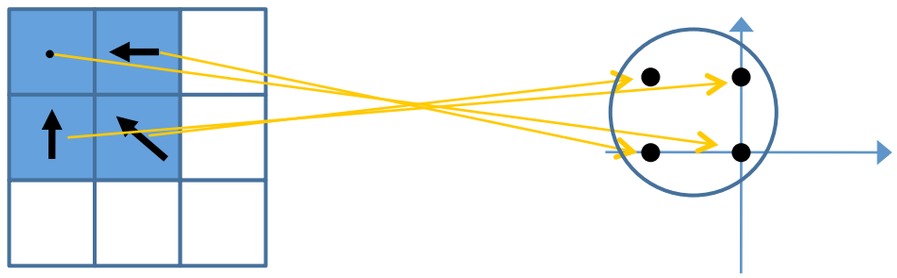
Corners in gradient space

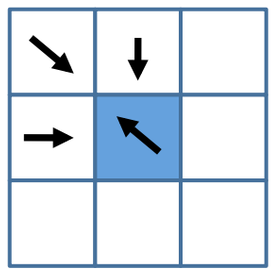


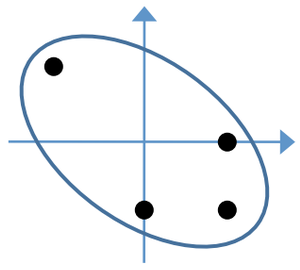


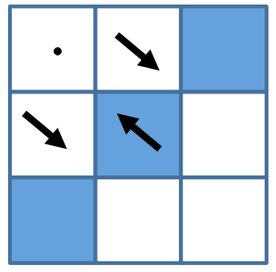


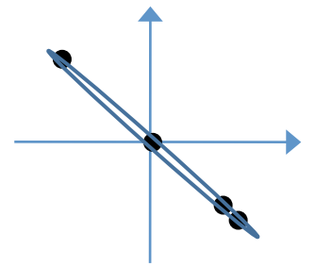
Corners in gradient space



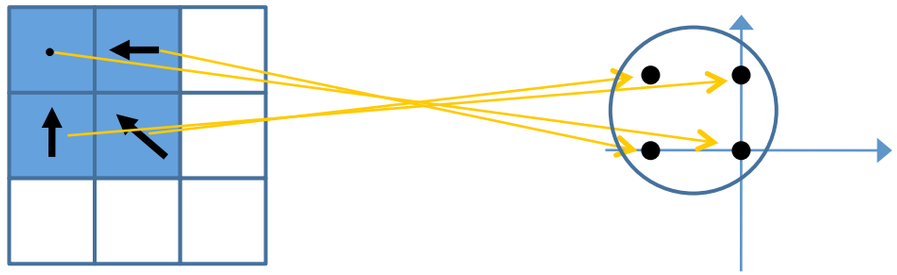


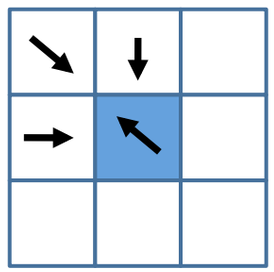


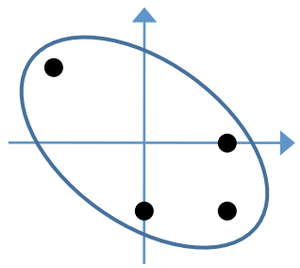


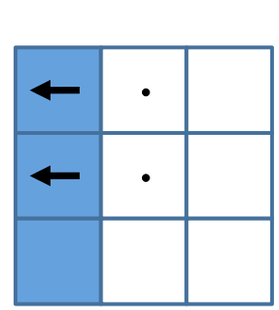


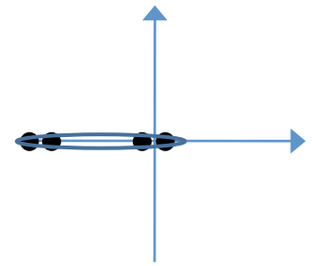
Corners in gradient space

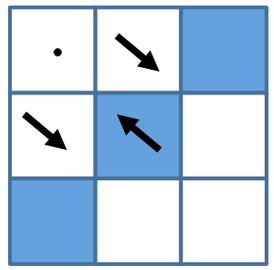


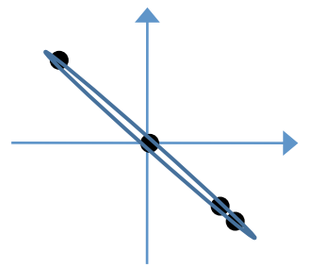




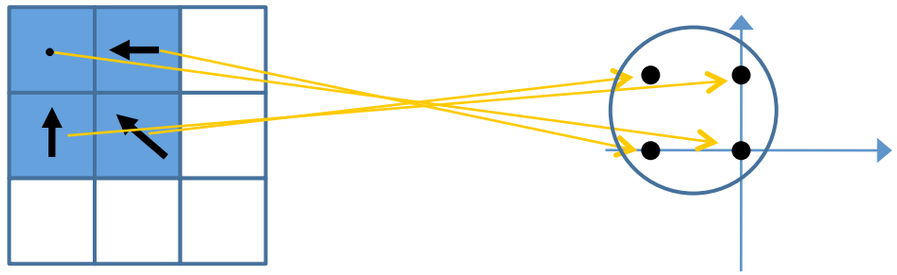


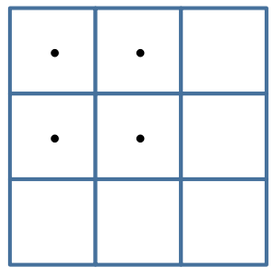


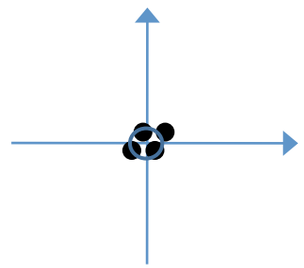


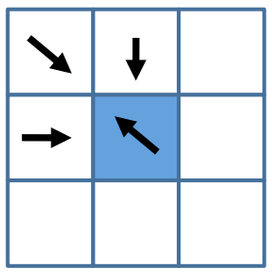


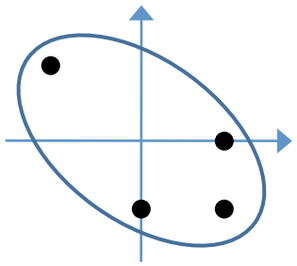
Corners in gradient space

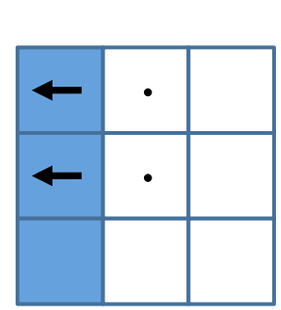


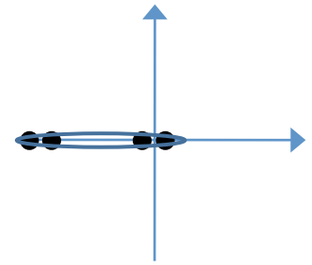


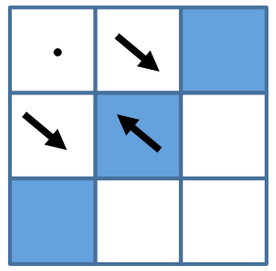


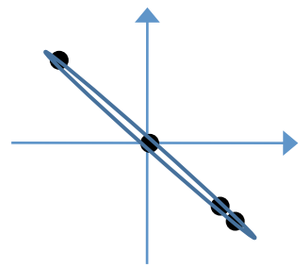




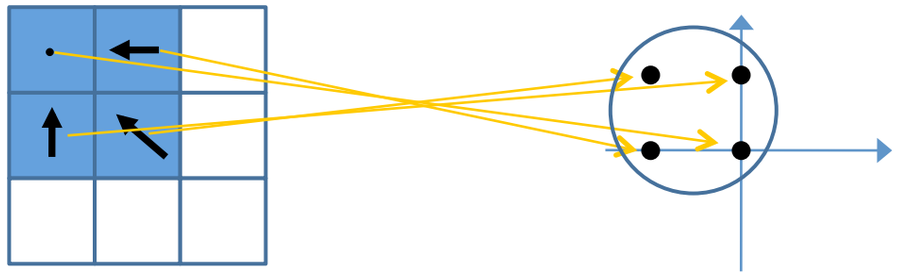


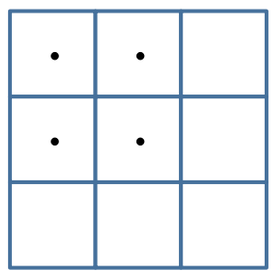


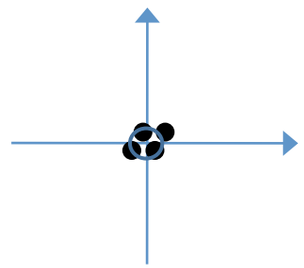


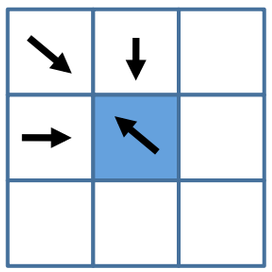


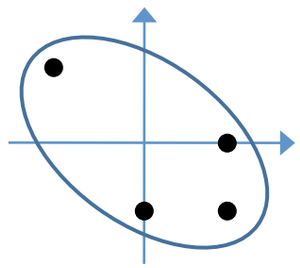
Corners in gradient space

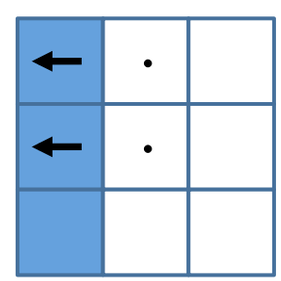


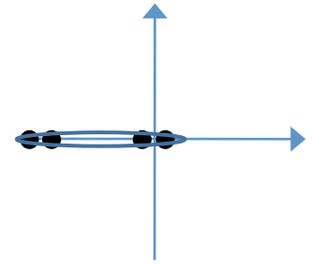


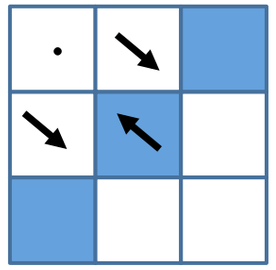


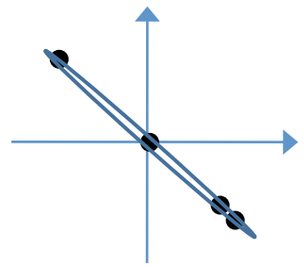






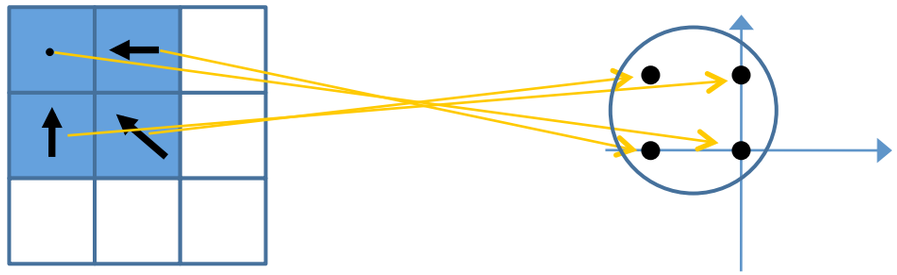


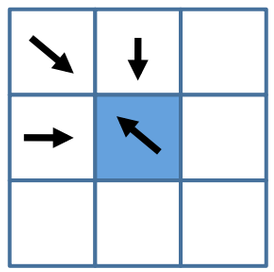


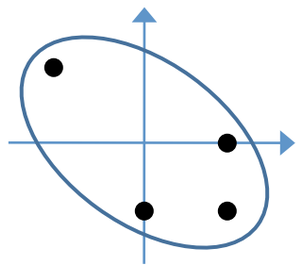


Good Corners

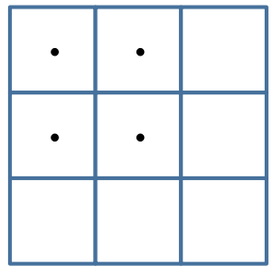
Corners in gradient space

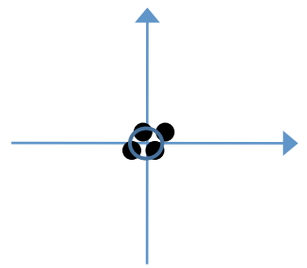


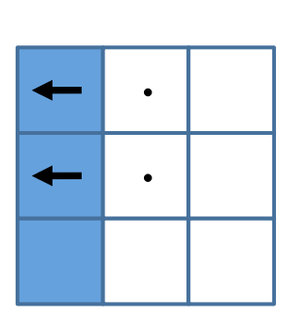


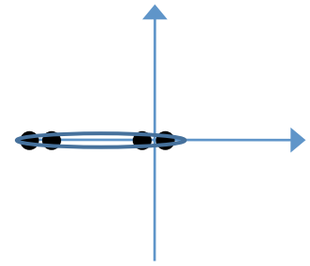


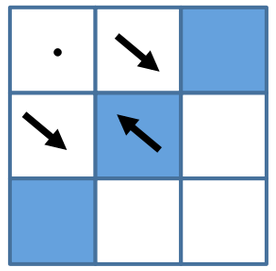
Good Corners

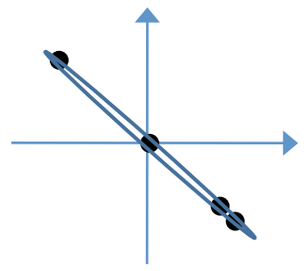








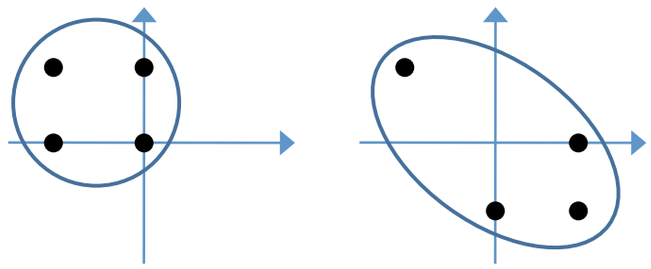




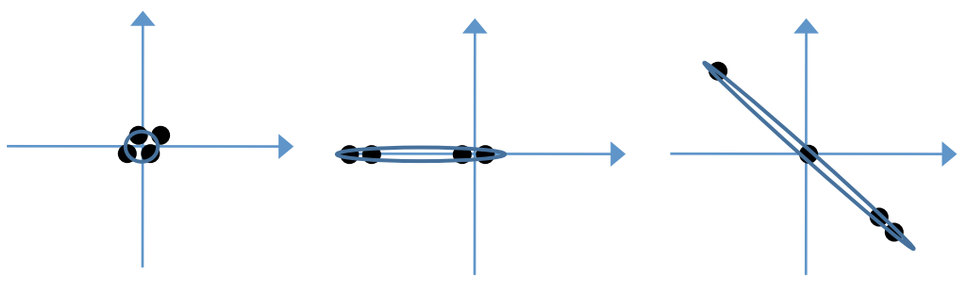
Bad Corners

Harris Corner Detector

- Good Corners



- Bad Corners



- Idea: The "fatness" of the covariance ellipse of the gradient directions is a measure of "cornerness"
  - How do we compute this?

Computing corner response

- Compute S matrix
  - Covariance of the gradients

$$S = \sum_i g_i g_i^T$$

- Compute eigen-values
  - Corner response = *smallest* eigen value
  - How bad (computationally) is this?
- Identify pixels with "good" corner responses
  - Thresholding

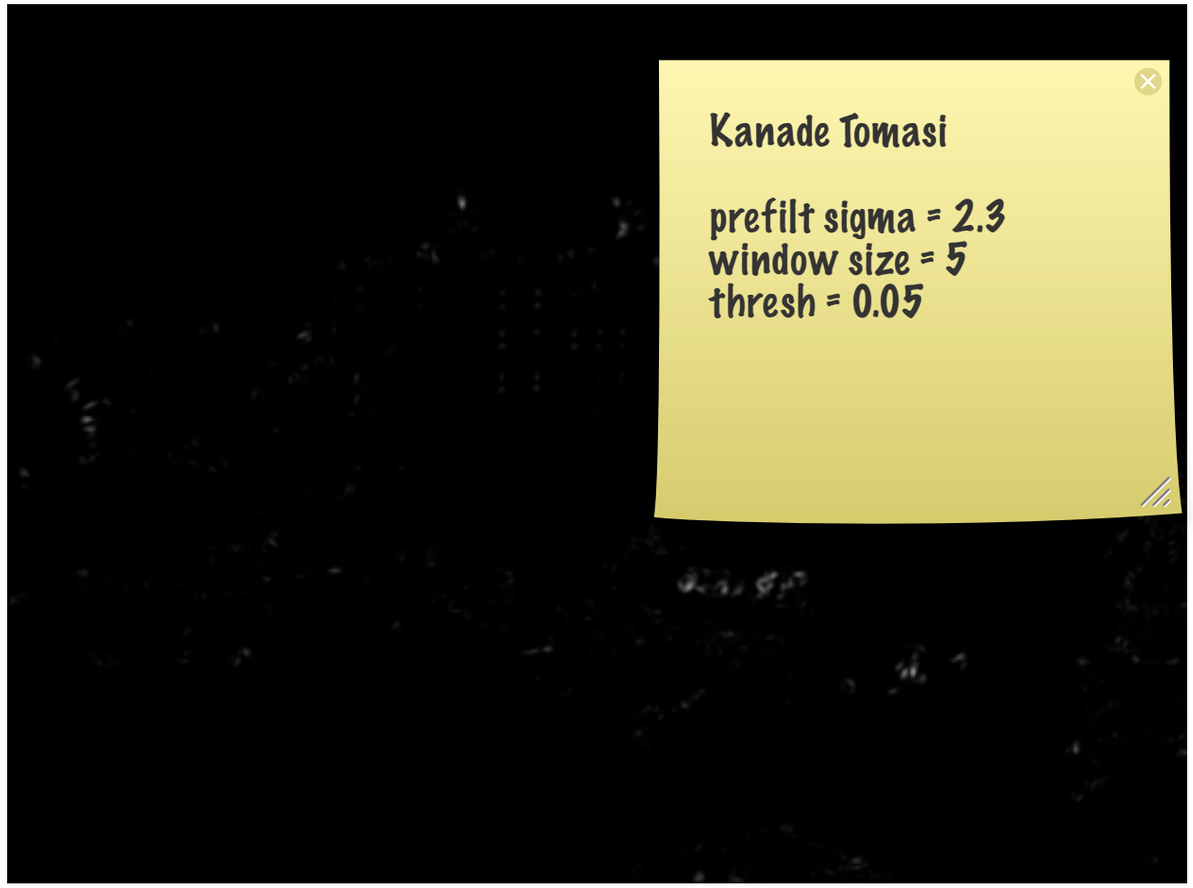
Computing corner response

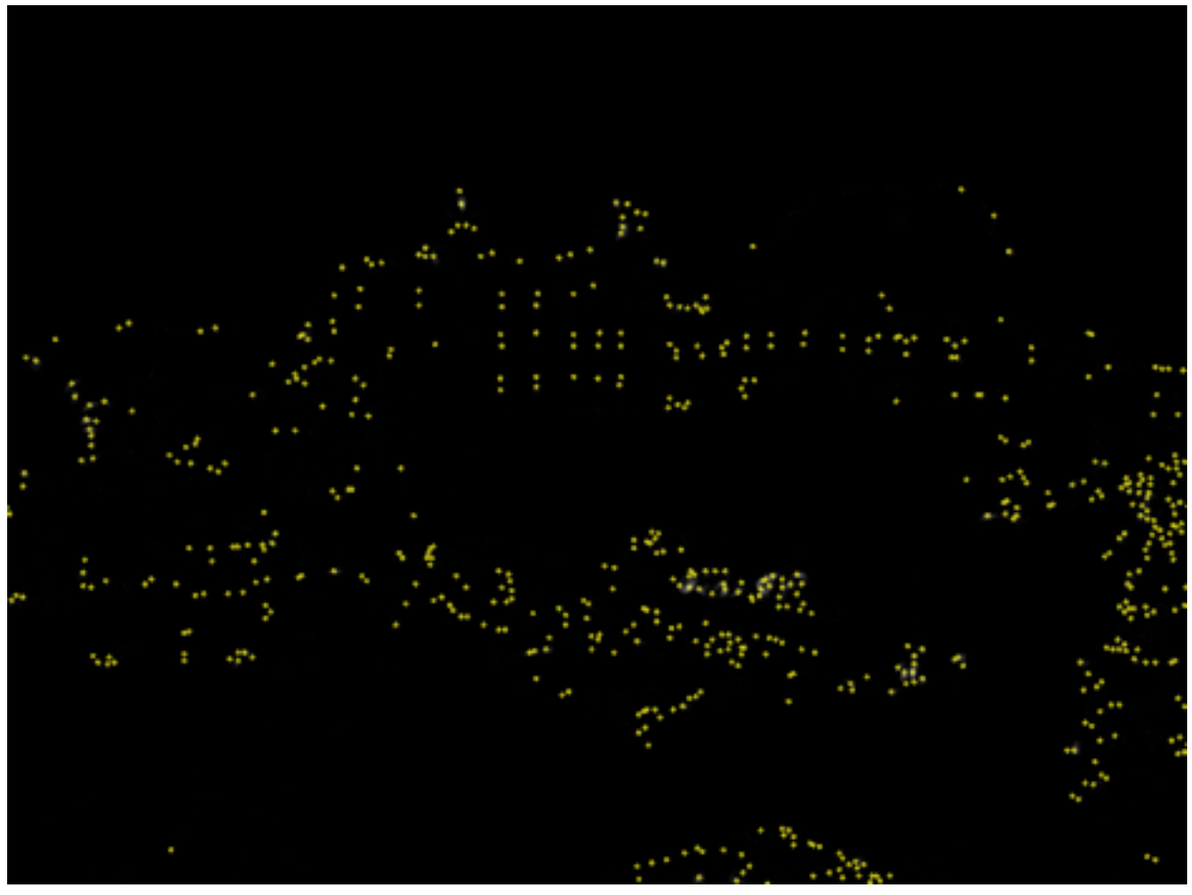
- Compute S matrix
  - Covariance of the gradients

$$S = \sum_i g_i g_i^T$$

- Compute eigen-values
  - Corner response = *smallest* eigen value
  - How bad (computationally) is this?
- Identify pixels with "good" corner responses
  - Thresholding

- A handful of practical issues!
- Noise in original image
  - Creates false positives
  - Apply low-pass filter *first*
    - Also improves isotropicity of response
  - Local maximum suppression
- How big a patch to compute gradients over?


Kanade Tomasi
prefilt sigma = 2.3
window size = 5
thresh = 0.05



Actually, I lied.

- There are two very closely related corner detectors
  - Kanade-Tomasi
    - what we described (uses eigenvalues)
  - Harris
    - identical, except uses (bad) approximation of eigenvalue.

$$\mathrm{trace}(\mathrm{S}) = \lambda_1 + \lambda_2$$
$$\det(\mathrm{S}) = \lambda_1 \lambda_2$$
$$M = \det(S) - \kappa \mathrm{trace}(S)^2$$

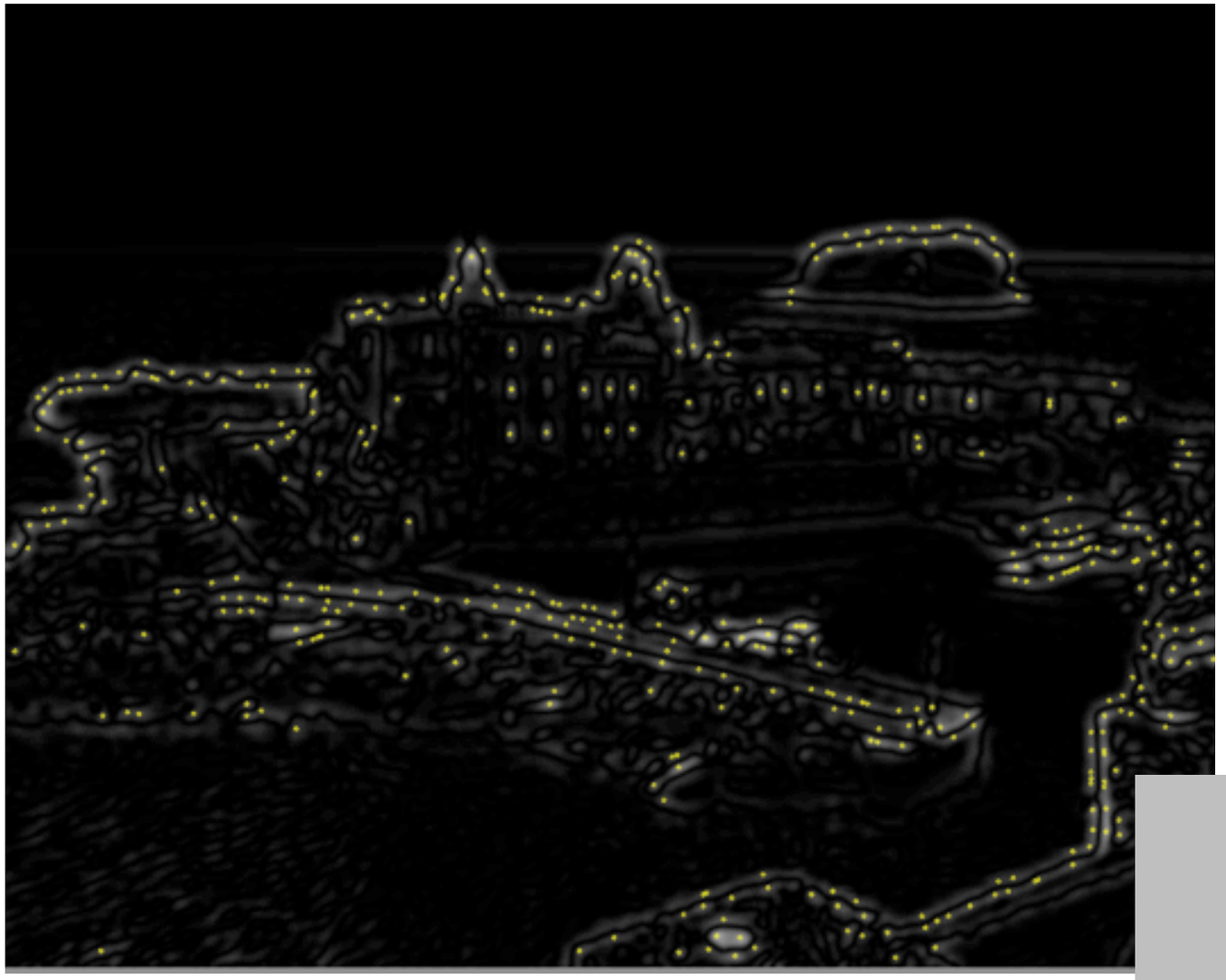
"Area of ellipse, minus a penalty for those that are highly eccentric."

Difference of Gaussians

- Another way of looking at corner detectors
  - Look for areas with high frequency in both directions

- What frequencies to look for?
  - We want a band pass
    - not too high (it's noise!)
    - not too low (it's not a corner!)

- Filter an image with two different Gaussians
  - Each corresponds to a low-pass filter
  - Difference corresponds to a band-pass filter

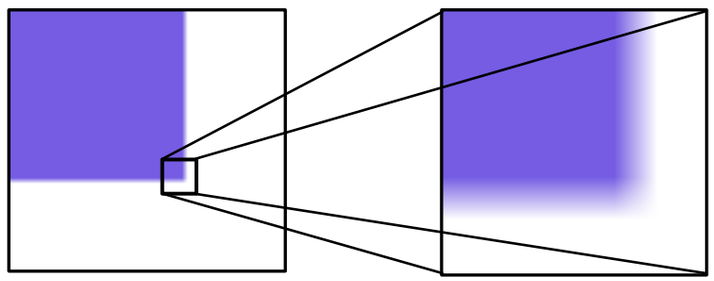


DoG, sigma1 = 5, sigma2 = 7

Of course, we could evaluate this for different band passes.

Multiple Scales

- Corners of high-resolution images are often blurry or noisy at fine levels of detail



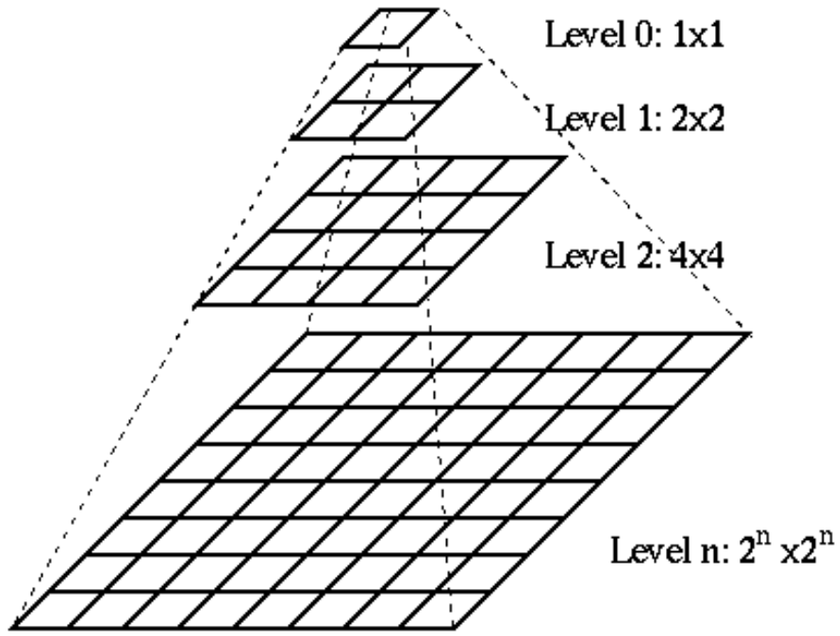
Corner indistinct at high resolution: no corner extracted

- Idea: run Harris corner detector on down-sampled versions of the image
  - Extract corners, blur, decimate, repeat.

- Idea: repeatedly compute DoG, increasing both sigma1 and sigma2
  - Look for successively lower frequency corners
  - Better yet, once we've band-limited "enough", we can decimate the image!

Image Pyramids

- Look for features on multiple scales
  - Just repeat image processing algorithm on successively lower-resolution images
  - Must produce lower-resolution images



- Avoiding aliasing requires low-pass filters
  - Ideal low-pass filter?
  - Don't create new features when filtering
    - Avoid ringing!
    - Want a monotonic filter

Can be shown that the only admissible filter is a Gaussian low-pass filter. However, can't achieve perfect frequency response… some tradeoffs necessary when building filter

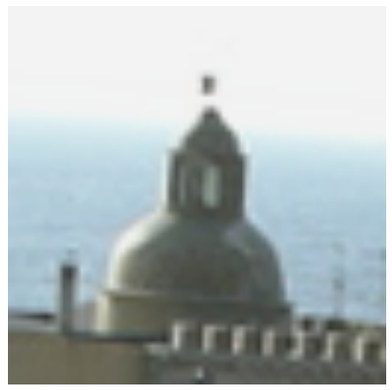
Feature Tracking

- We often want to track (or match) features across two frames.
  - Which corners in image A match those in image B?
  - i.e., data association

- Can we use *more* information?
  - Why not use the local appearance?

Image patch patching

- Consider the pixel patch around a feature
  - Sum of absolute/squared (SAD/SSE) differences/errors
- How robust is this to small alignment errors/rotations/ changes in viewpoint/etc.?

These will probably match


These probably won't

Invariances

- Our goal: detect distinctive features, maximizing repeatability
  - Transform pixel patch into a space where a simple comparison (SAD/SSE) *is* effective.

- Scale invariance
  - Robust to changes in distance

- Rotation invariance
  - Robust to rotations of camera

- Affine invariance
  - Robust to tilting of camera

- Brightness invariance
  - Robust to minor changes in illumination

SIFT: Scale-Invariant Feature Transform

- David Lowe (Univ. British Columbia)

- Probably the single most commonly used tool in computer vision
  - For better or for worse... often used "reflexively" even if it's not a good choice!

- Watch out!
  - Patented, commercial use restricted

SIFT

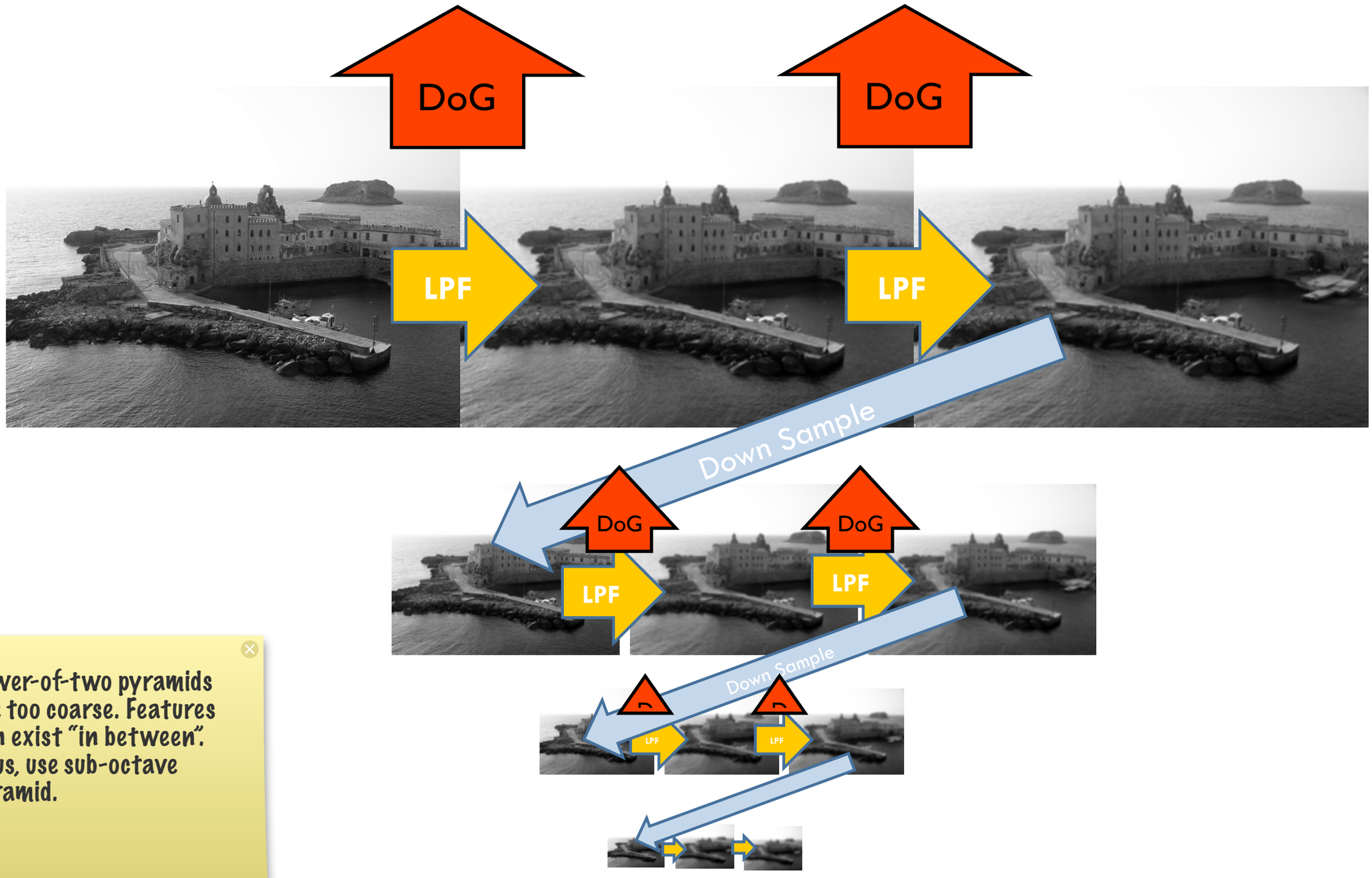
- Detect interest points
  - Image pyramid using DoG "corners"
  - Output: corners *and* scale (which level of the pyramid?)

- Output a "descriptor"
  - Consider pixel match around corner
  - Compute a histogram of the gradient directions
  - "Rotate" the histogram so that the dominant direction is first.

Sub-Octave Image Pyramids

power-of-two pyramids are too coarse. Features can exist "in between". Thus, use sub-octave pyramid.

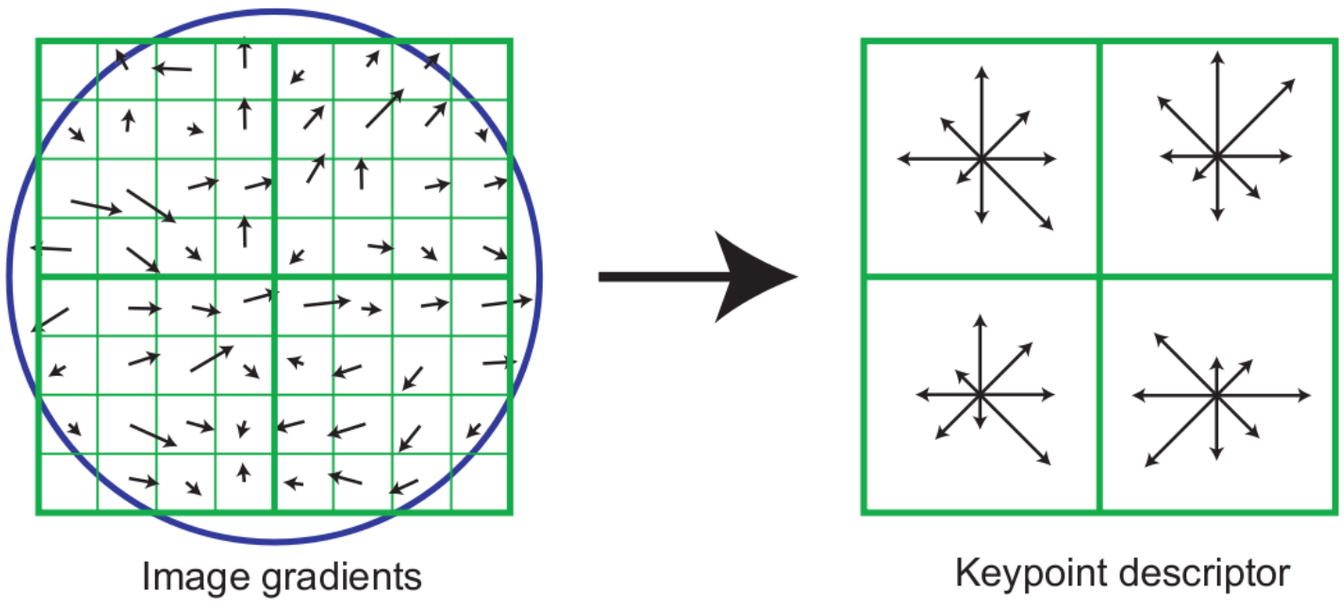
Sub-Octave Image Pyramids



power-of-two pyramids are too coarse. Features can exist "in between". Thus, use sub-octave pyramid.

SIFT Descriptor

- Histogram of gradients gives good information about a pixel patch
  - But building just one histogram loses a lot of spatial information.
  - Idea: For a given interest point, compute a set of histograms; output each.
  - Shift histograms so dominant direction is first in histogram ==> rotational invariance.



- "Official" SIFT uses 16x16 pixel patches, 4x4 bins, 8 histogram buckets
- How many degrees of freedom in SIFT descriptor?
  - # bins * # histogram buckets = 4*4*8 = 128

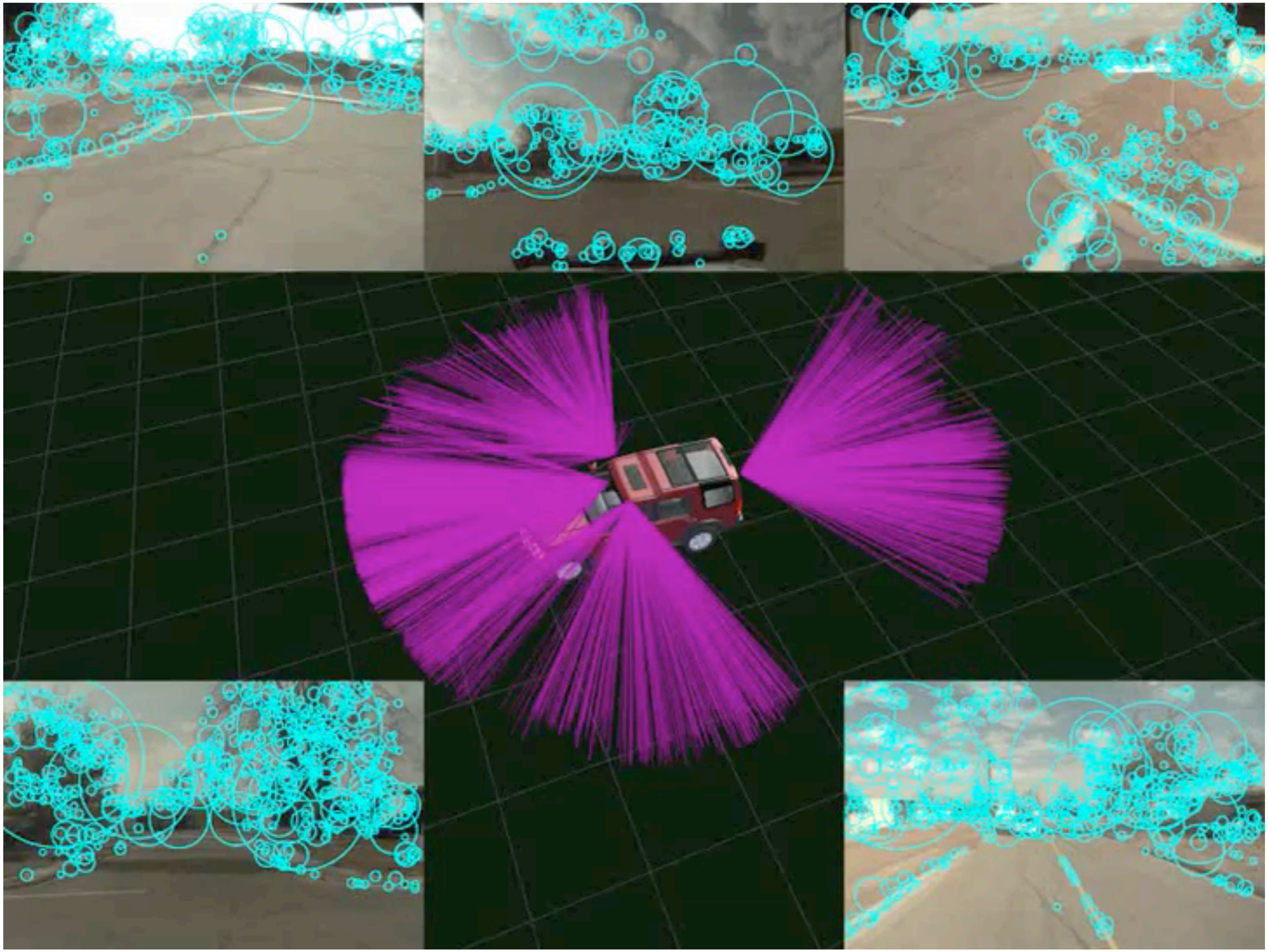
Matching SIFT Descriptors

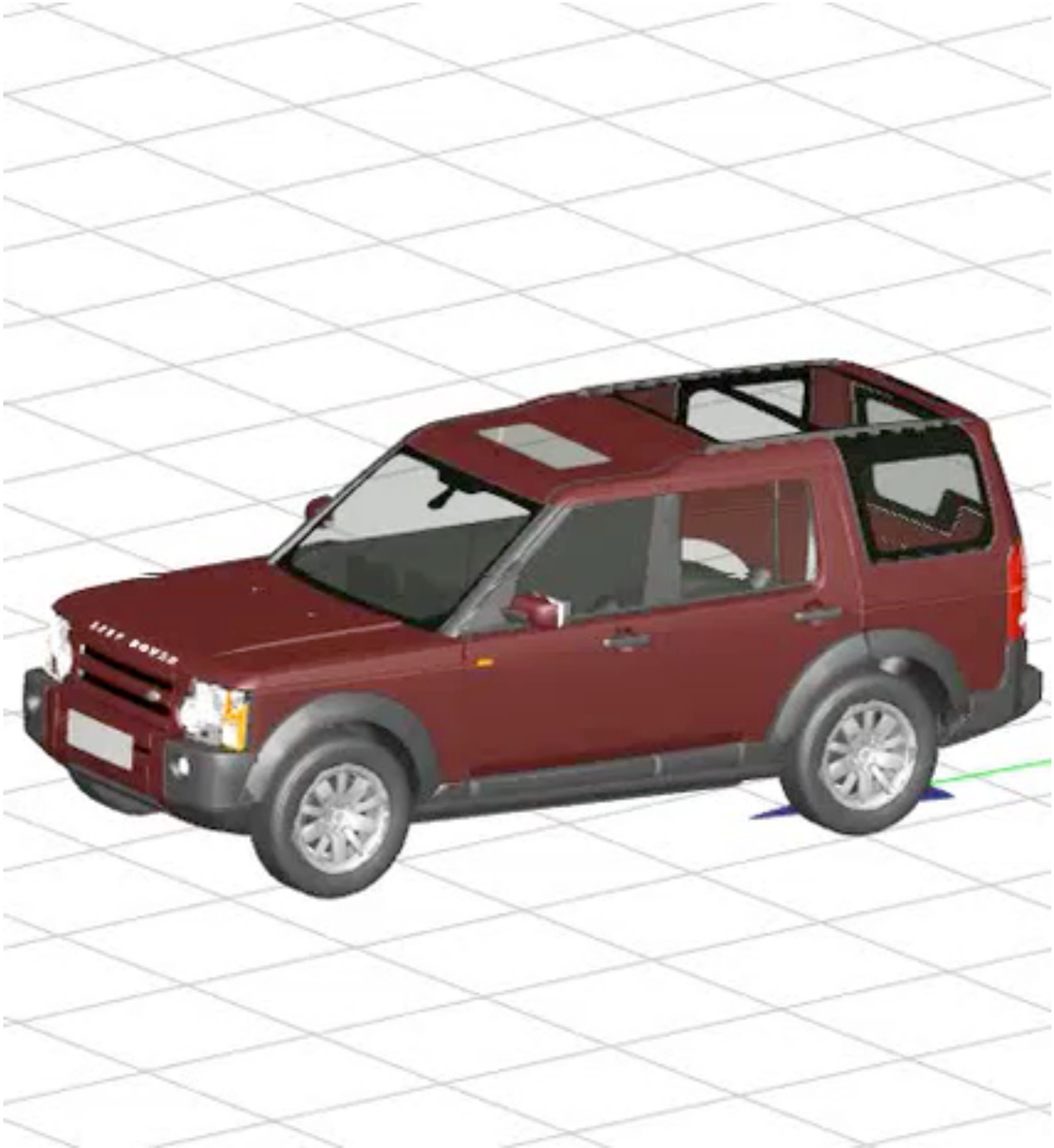
- Each SIFT feature:
  - (x,y,scale) (ignore scale if you want scale invariance!)
  - descriptor[128]

- Two descriptors can be compared using Euclidean distance…
  - Small distances = similar descriptors
  - What if same/similar feature appears more than once? nearest neighbor may not be good enough

- Common approach:
  - Suppose best match for Ai is Bj (with dij).
  - Suppose next best match for Ai is Bk (with dik).
  - Require dij < alpha dik. (alpha typically 0.8).

- "Marriage" constraint: Ai and Bj match only if Bj is the best feature for Ai *and vice versa*.





Object recognition

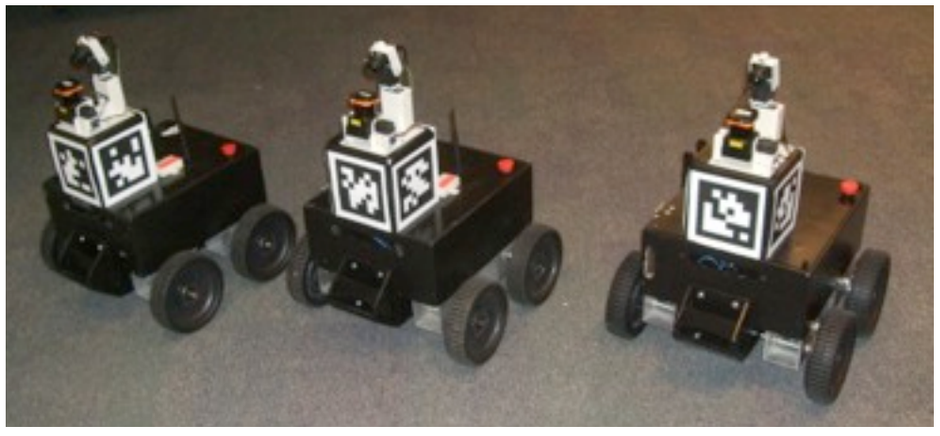
- SIFT also used to build object recognition systems

Artificial Features

Applications

- Ground truthing
- Recognizing robots
- Commanding robots

- Education
  - Often useful to bypass open-ended perception problems




TagTest
Wed Nov 4, 9:58 AM
smoothing sigma (segmentation)
smoothing sigma (sampling)
magnitude threshold
theta threshold (radians)
debug

Related Work

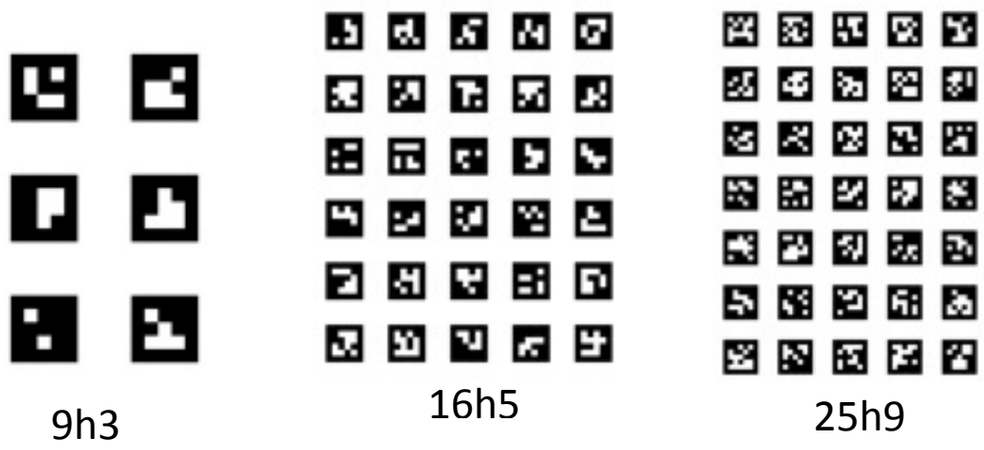
- ARToolkit
  - Widely used
  - Primitive binarization scheme => high failure rate in unstructured environments
  - Weak coding system
  - Freely available
- ARTag (Fiala, 2005)
  - Seems to address many shortcomings in ARToolkit
  - Methods are not well-documented
  - Source code not available

- Bokode (Mohan et al, 2009)
- Fourier codes (Sattar et al, 2007)
- Quick Response (QR) Tags

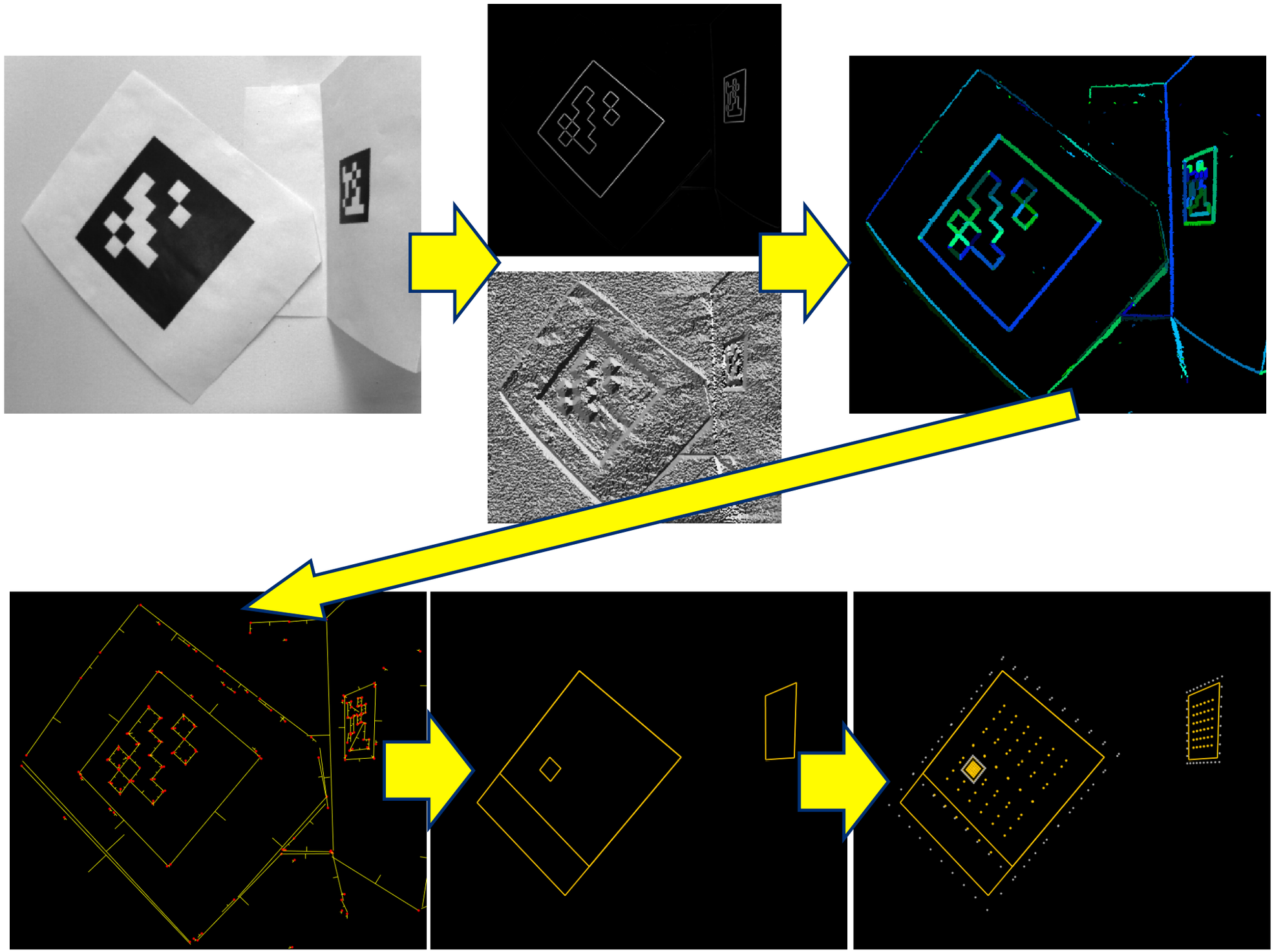
(a) Layout (b) ID=6 (6 bits) (c) ID=625518 (36 bits)

AprilTags

- Robust detection
  - Not based on threshold-based binarization scheme
  - Works better in unstructured environments
  - Accurate localization
- Strong coding system
  - Low false positive rate
- Parameterizable
  - Pick your own tag family



Detection Approach




id 0
(err=0, rot=3)

Coding System

- Based on lexicographic code
  - Nearly optimal coding family

- Simple algorithm:

```
codebook = {}
for i = 0 : max-codeword
    if (hamming_distance(i, codebook) > H))
        codebook = codebook U i
return codebook
```

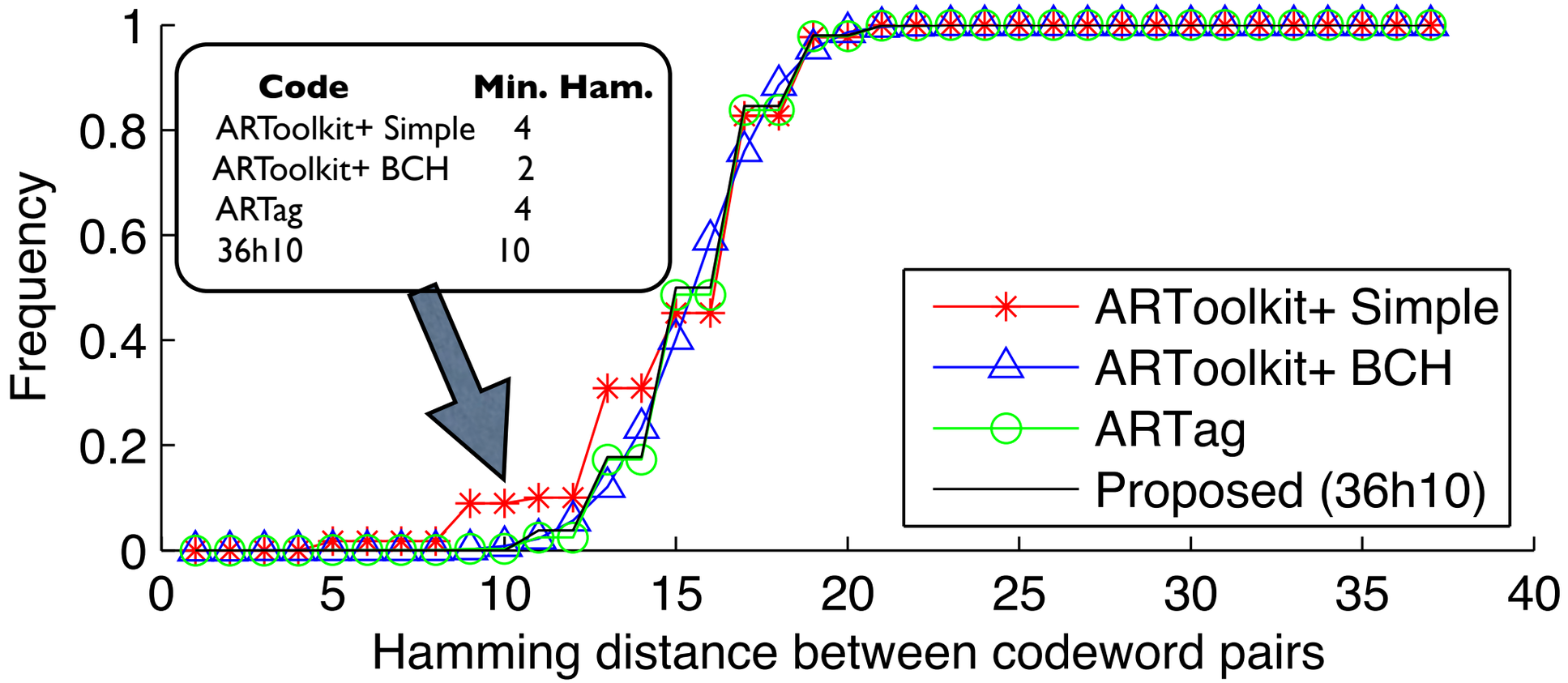
Coding System

- But we need rotational invariance too!

```
codebook = {}
for i = 0 : max-codeword
    if (hamming_distance(i, codebook) > H))
        codebook = codebook U { i, rot90(i), rot180(i), rot270(i) }
return codebook
```

- Note how code generation system can be easily modified to incorporate additional constraints.

Coding System: Optimality

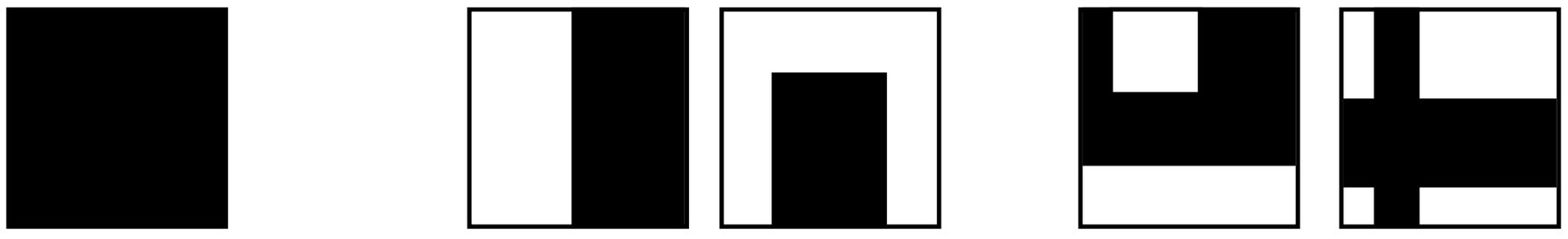


Tag Complexity

- Coding scheme can generate "perceptually weak" tags
  - All black tag--- likely to appear by chance in images
  - ARTag manually identified two "bad" tags and rules them out.

- How can we measure the "badness" of a tag automatically?

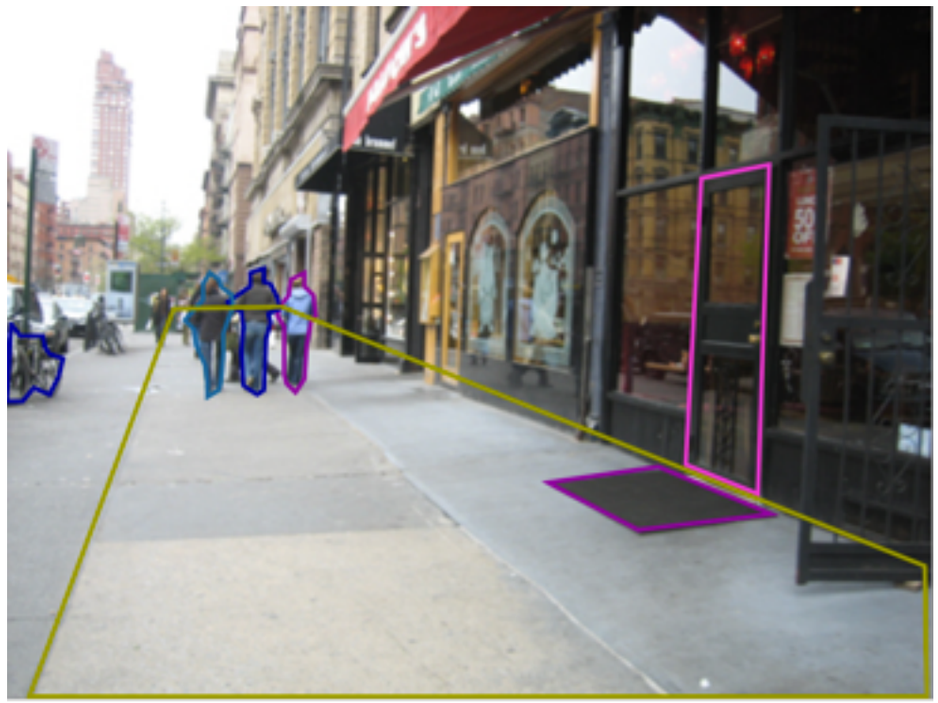
- Idea: How many rectangle drawing operations would it take to generate a tag?
  - Simple greedy search computes upper bound



c=1 c=2 c=3

Results: False Positives

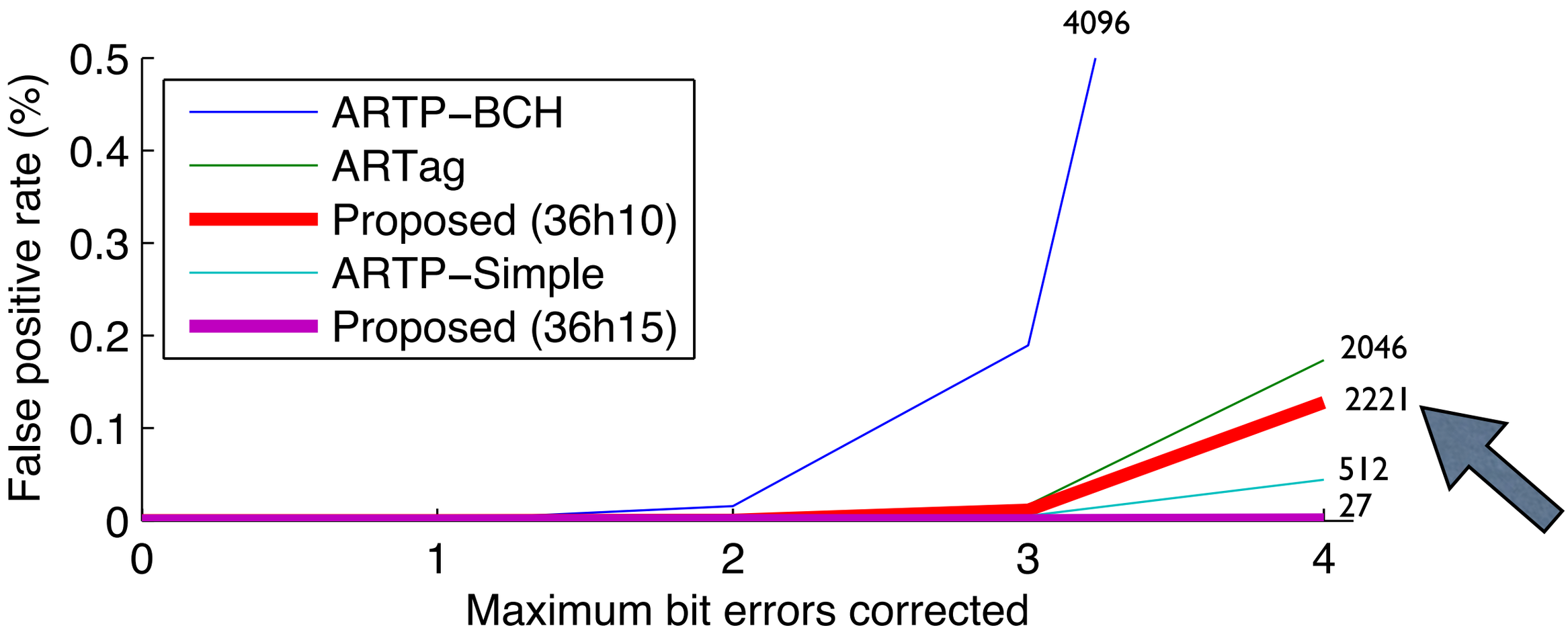
- Use large corpus of images *not* containing AprilTags. Do we detect any?



LabelMe (Russell 2005)

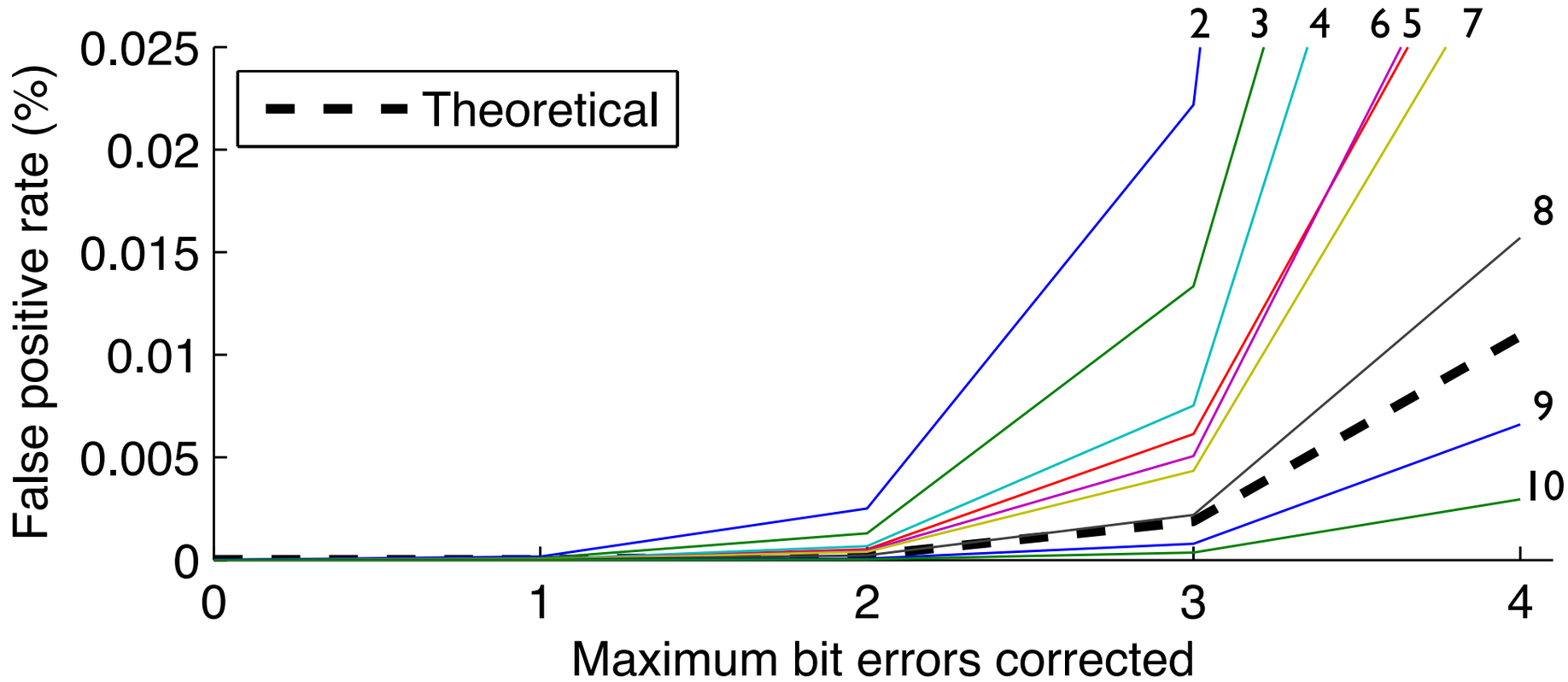
- Which dataset should I use?
  - Should convince you that I haven't "cooked" it
  - Used open "Label Me" dataset
    - Huge! (180,829 images)
    - Wide variety of topics, cameras, image quality

False Positives (Label Me)



AprilTag 36h11 outperforms ARTag: more encodable tags at lower false positive rate!

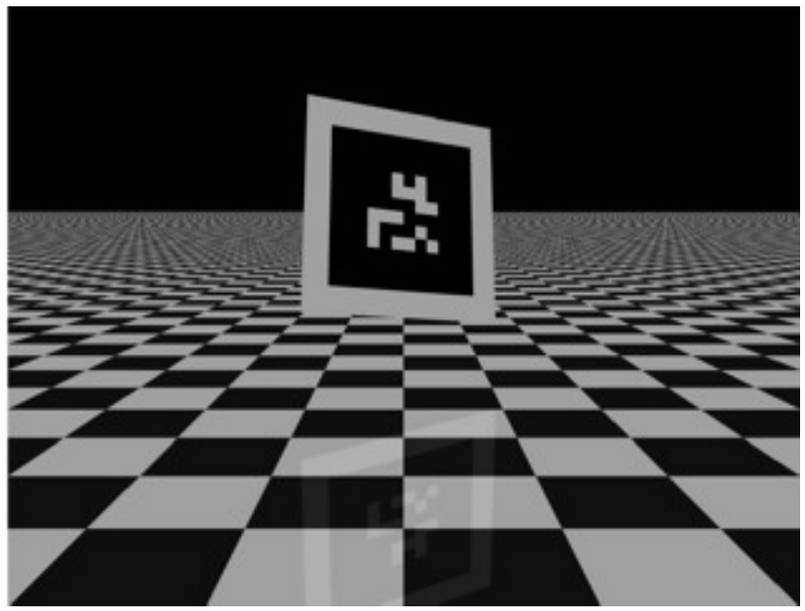
False positives versus complexity: Label Me



- Conclusion: Our tag complexity heuristic *does* help us reject naturally-occurring patterns
  - At c > 8, our tags are less likely to appear in real-world images than completely random tags!

Evaluation: Accuracy

- For reliable ground-truth, used synthetic ray tracing images
  - Correct answer known exactly

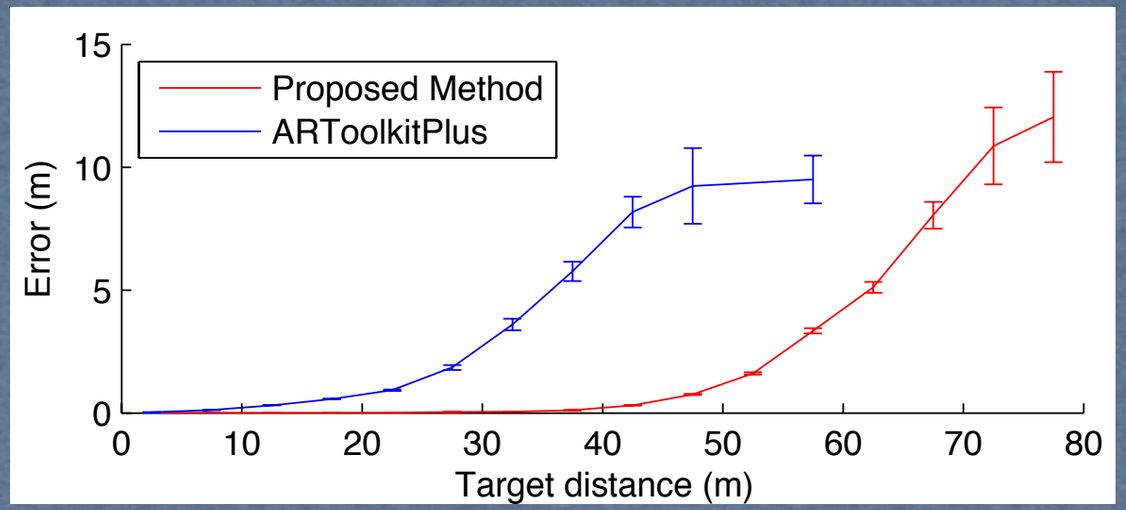


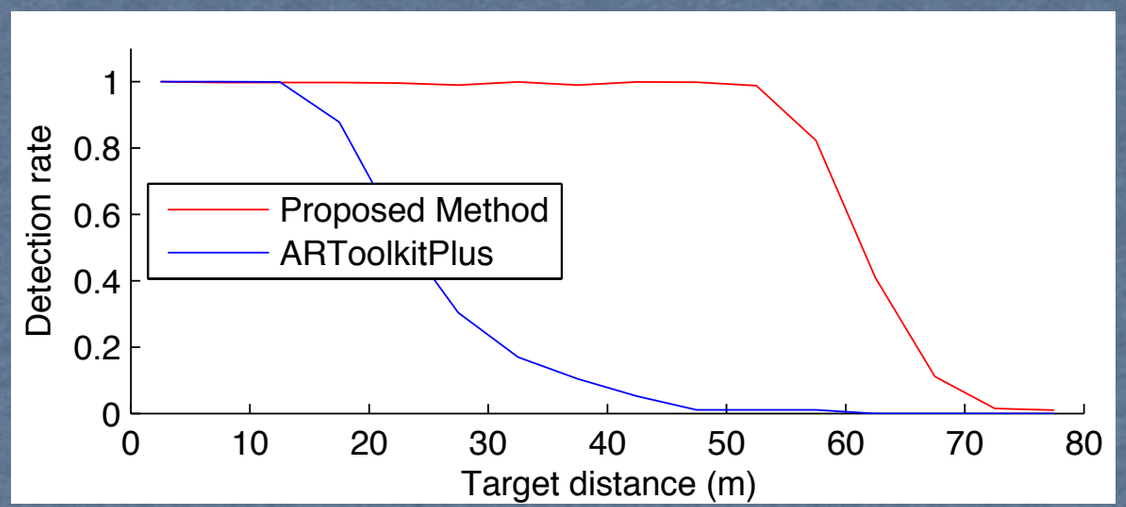
- Two main factors in detection accuracy
  - Distance from camera
  - Angle to camera

- For each particular experiment, we want to know
  - Accuracy
  - Detection rate

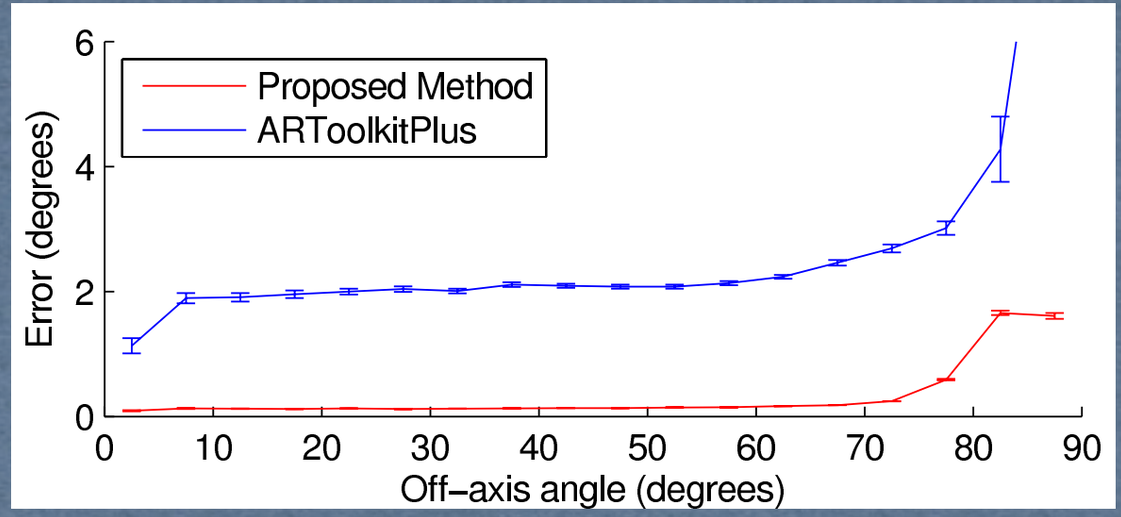
Results: Accuracy

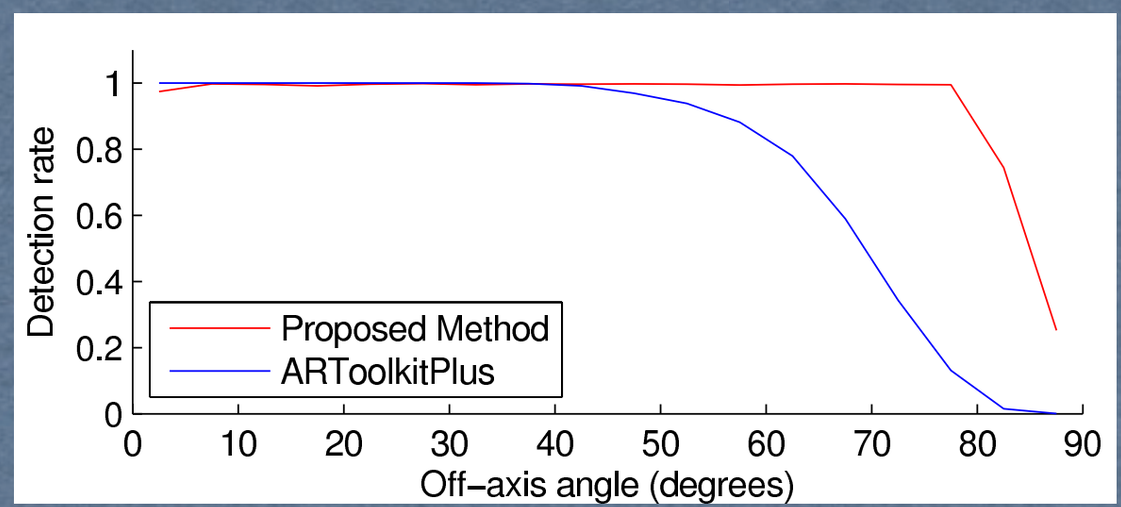
Distance





Orientation





- AprilTag both more accurate and detects more cases!

AprilTag: Conclusions

- Very useful for robotics
  - Ground truthing
  - Commanding robots
  - Robots recognizing each other
  - Education

- Out-performs ARToolkit and ARTag
  - Better detection method
  - Better coding system

- Free and Open Source!

http://april.eecs.umich.edu