CS448f: Image Processing For Photography and Vision

Alignment

Assignment 1 Grading

- Graded out of 20
- 4 points for correctness
- 4 points for readability
- 8 points for accuracy
  - 4 for hitting the minimum (0.07)
  - 7 for hitting the goal (0.05)
  - Linear in between
- 4 points for speed
  - 2 for hitting the minimum (at least as fast)
  - 3 for hitting the goal (50% faster)
  - Linear in between

Assignment 2 Grading

- Graded out of 20
- 4 points for correctness
- 4 points for readability
- 4 points for accuracy
  - 2 for 0.007
  - 3 for 0.005 (the goal)
  - 4 for 0.003 or less
- 8 points for speed
  - 0 for the same speed
  - 4 for 2x faster
  - 7 for 5x faster (the goal)
  - 8 for 10x faster or better

Noise

- So far we've tried:
- Averaging pixels with nearby pixels
  - They're probably looking at the same material
- Averaging pixels with nearby pixels that have a similar value
  - They're probably looking at the same material
- Averaging pixels with nearby pixels that have similar local neighborhoods
  - They're probably looking at the same material

Noise

- How else can we get more measurements of the same material?

Noise

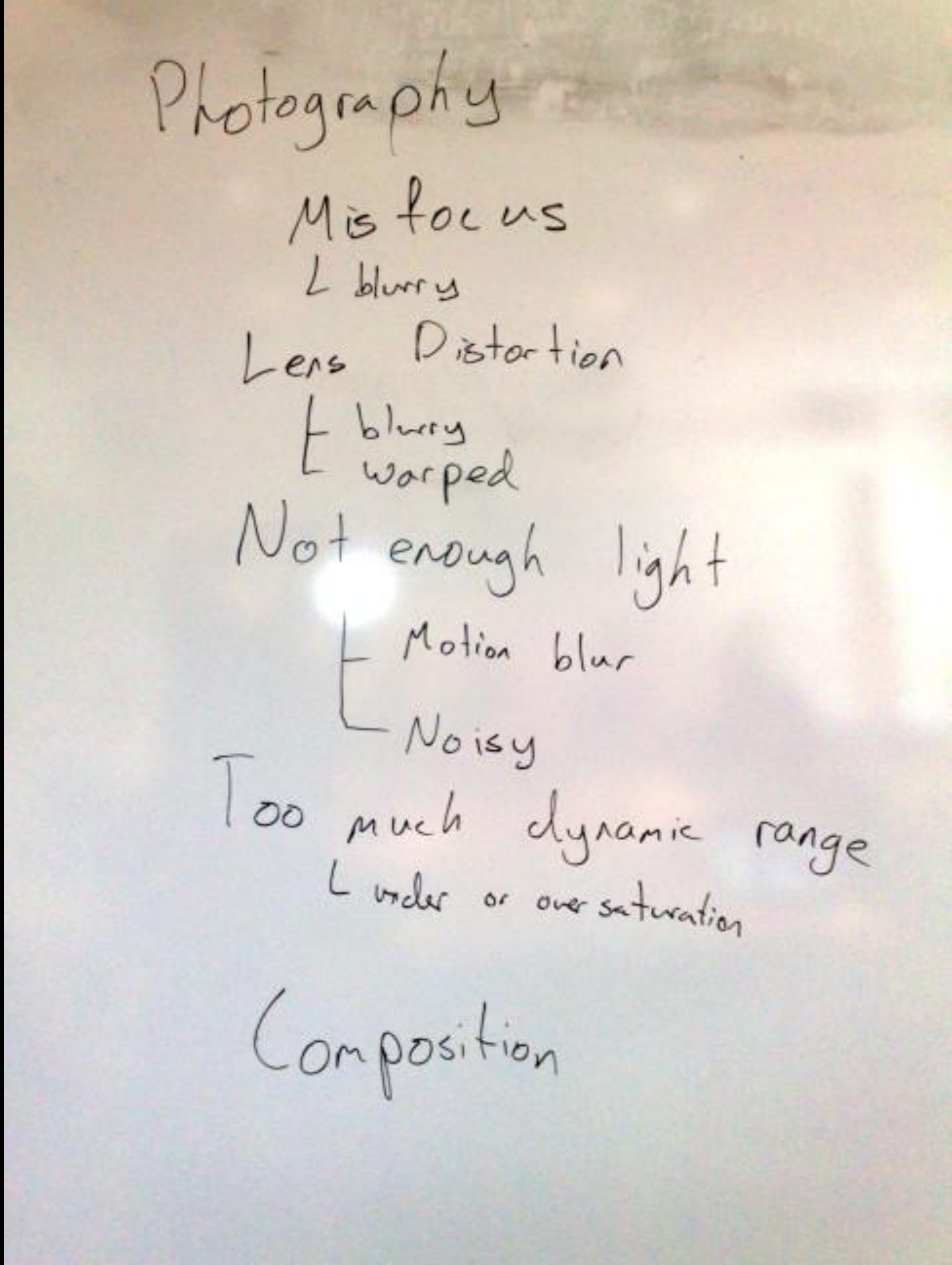
- Take multiple photographs of it!
- Average the results
  - Perfect for static scenes and perfect alignment
- Or take the median over time at each pixel
  - Perfect if there are transient occluders (eg a bird flies across your scene)
- Or use a bilateral filter over space and time
  - More robust if the alignment isn't perfect
- Or use non-local means on the entire burst
  - Can cope with pretty poor alignment

Noise

- Take multiple photographs of it!
- ALIGN THE PHOTOGRAPHS
- Average the results
  - Perfect for static scenes and perfect alignment
- Or take the median over time at each pixel
  - Perfect if there are transient occluders (eg a bird flies across your scene)
- Or use a bilateral filter over space and time
  - More robust if the alignment isn't perfect
- Or use non-local means on the entire burst
  - Can cope with pretty poor alignment

Alignment

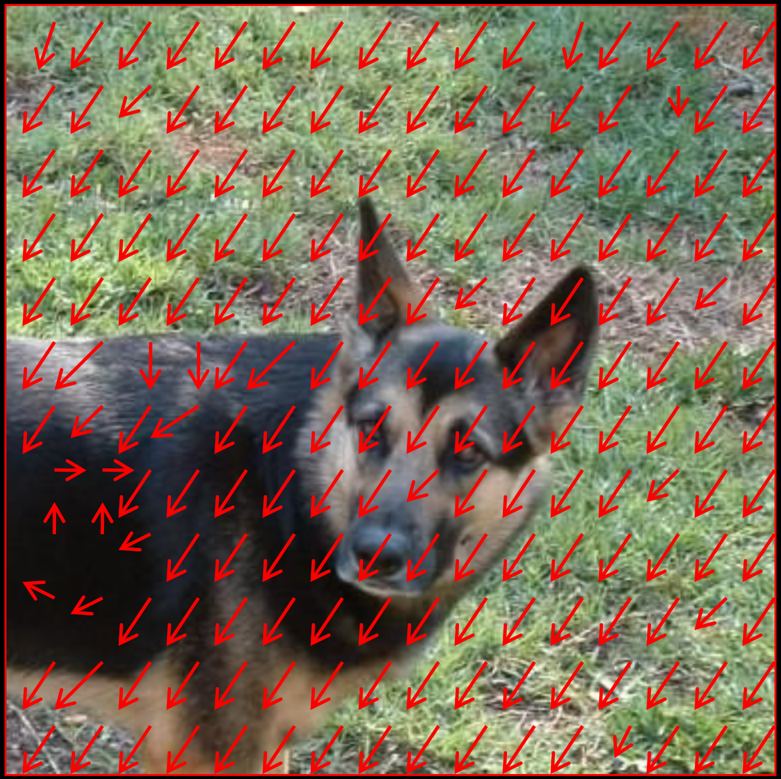
- What other problems might it help with?



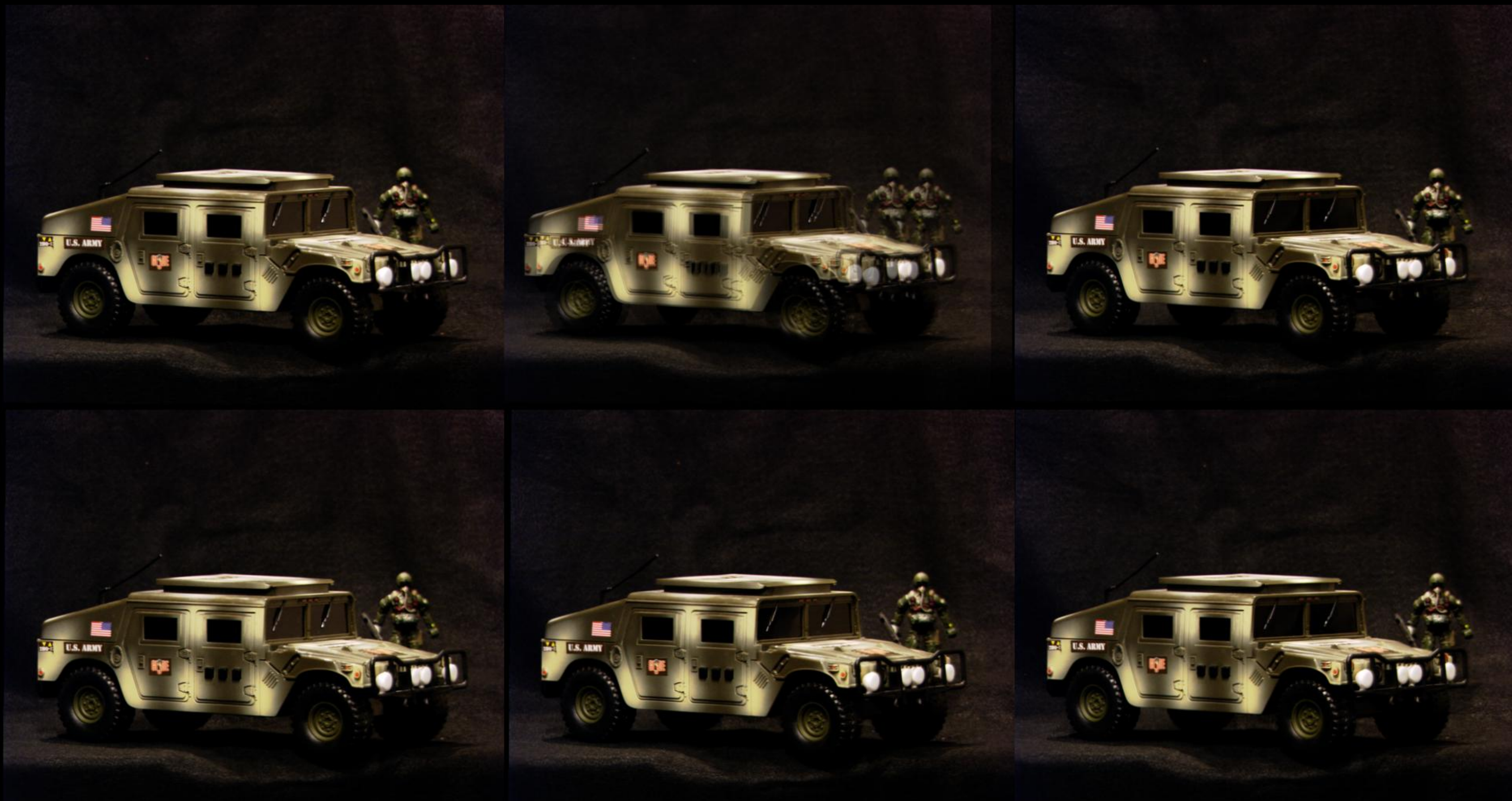
Optical Flow

Optical Flow

Optical Flow



Application: View Interpolation



Left Input Output Right Input

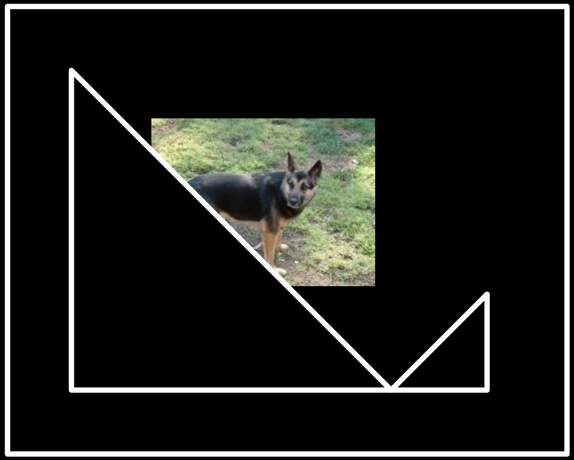
Moving Gradients

- Mahajan et al. Siggraph 2009.

(a) Input Image A (b) Interpolated Frames - Our Method (c) Input Image B

Downsides:

- Slow
  - Search required at each patch
  - How far to search?
- Error prone
  - Regions without texture will fail
  - Occlusion boundaries may fail
  - Regularization can help



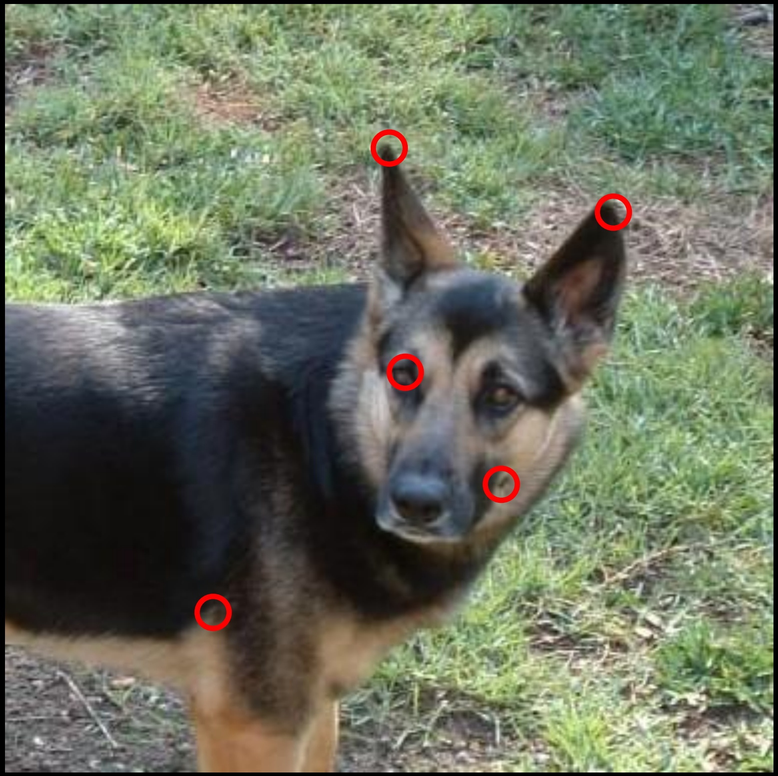
What if we just need a global motion?

Point Features

- Why use the whole image?
  - Much of it was problematic to align
- Better just to pick a few good landmarks
  - You don't need to memorize what every building on campus looks like. If you get lost, just look for Hoover tower.
- Compresses the problem
  - Can be very fast!

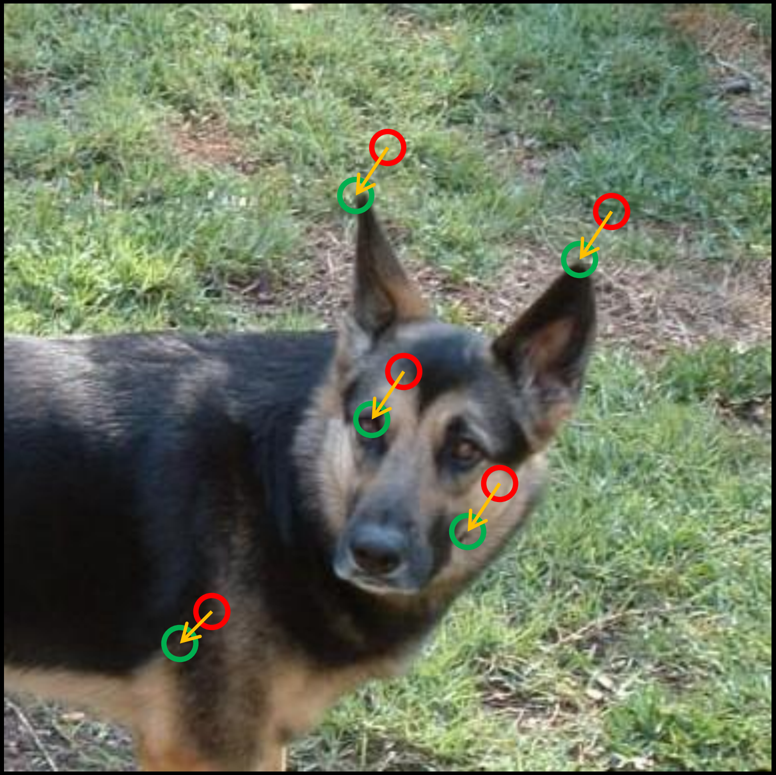
Inputs:

1) Find Point Features



1) Find Point Features

2) Find Correspondences



Finding Features

- 1) Figure out what points to extract
  - Features
- 2) Figure out what other points they match
  - Correspondences
- 3) Find a warp that satisfies the correspondences

Finding Features

- A good point is **localizable** in space
  - unlike its neighbours
- Therefore: take the average of the neighbours (gaussian blur), and subtract it from the original
- (demo)
- Picks up very fine corners

Point Tracking

- Can change the scale of points we're looking for by using the difference of two Gaussians
- (demo)
- more robust to noise, looks for larger, stronger features
- There are many corner detectors
  - difference of Gaussians is pretty good

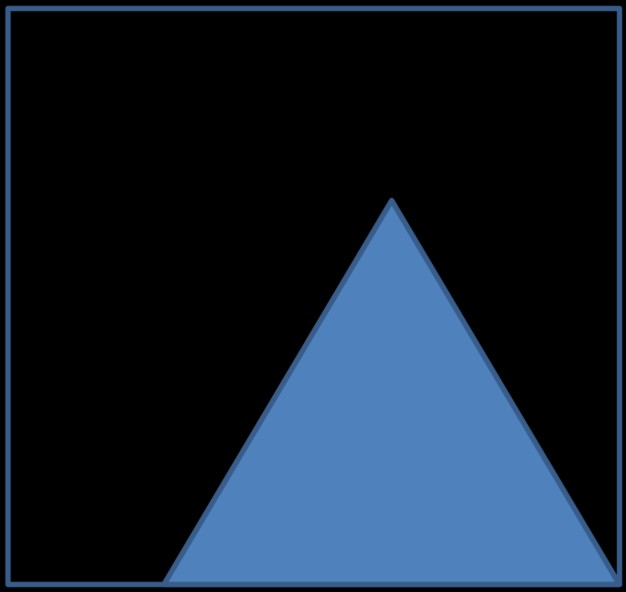
Extracting Corners

- All we have so far is a filter that measures "cornerness"
- Look for local maxima of this function to get actual corners
- Can compute sub-pixel local maxima location by fitting parabola (see LocalMaxima::apply).

Better Corner Detector Filters

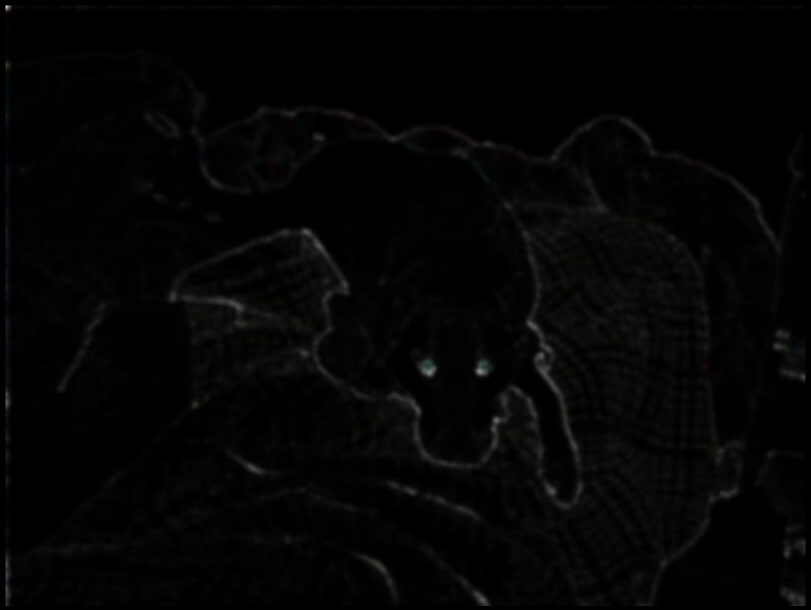
- The Moravec Corner Detector
  – Compares patch around this pixel to patch around neighboring pixels. Sums the patch distances.
- Harris Corner Detector
  – Differentiates this quantity directly with respect to direction
  – You end up with a covariance matrix of local gradients
  – High variance = wide range of local gradients = corner

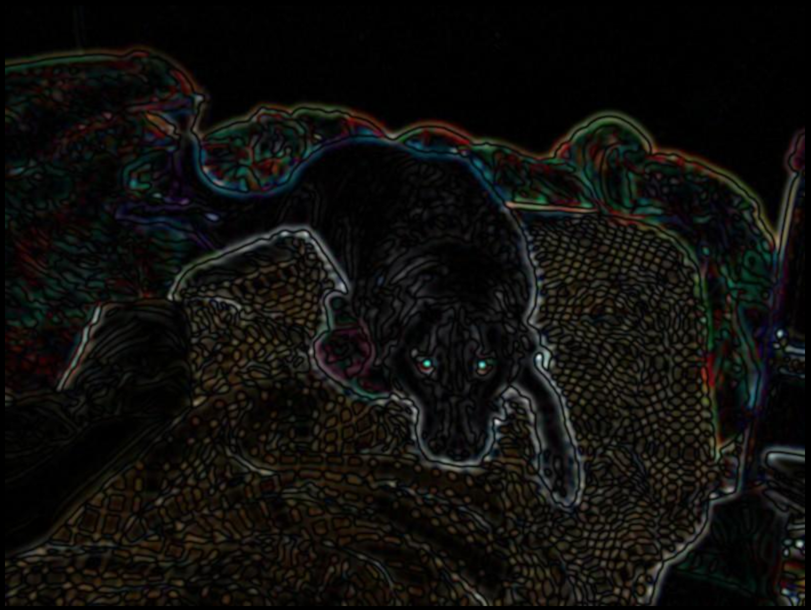
Harris Corner Detector



$$\begin{bmatrix} B*G_x^2 & B*G_xG_y \\ B*G_xG_y & B*G_y^2 \end{bmatrix}$$

Harris vs D.o.G





Corners aren’t very big

- Not much information to work with to compute a detailed descriptor
- How about looking for interesting regions instead?
- Small corners also tend to change rapidly
  - jaggies
  - highlights
  - occlusion-based

Blob Detection

- Take difference of Gaussians at various scales
- Look for “scale-space extrema”
- Demo

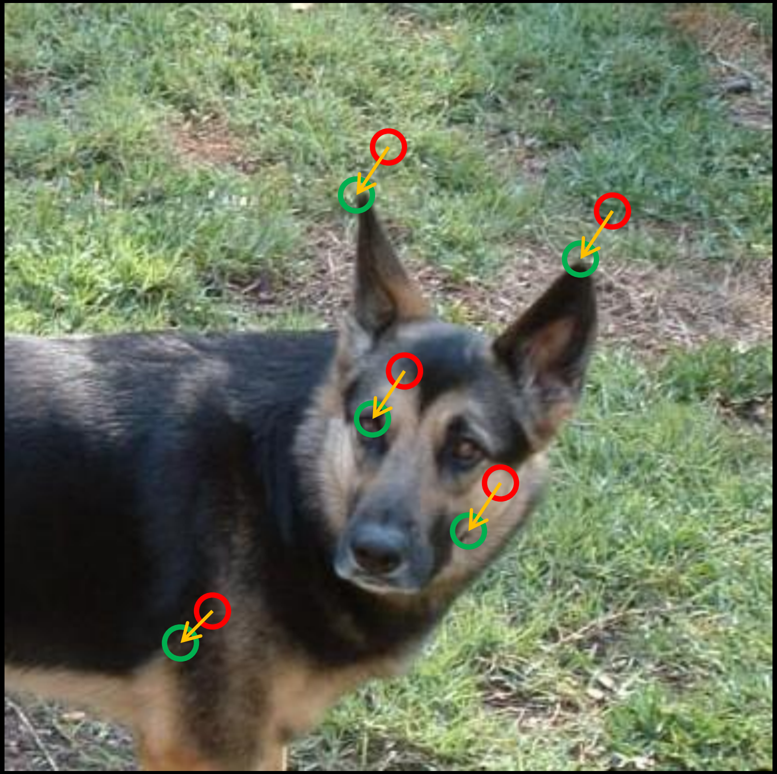
- Downside of using large blobs?

Point Tracking

- 1) Figure out what points to extract
  - Features
- 2) Figure out what other points they match
  - Correspondences
- 3) Find a warp that satisfies the correspondences

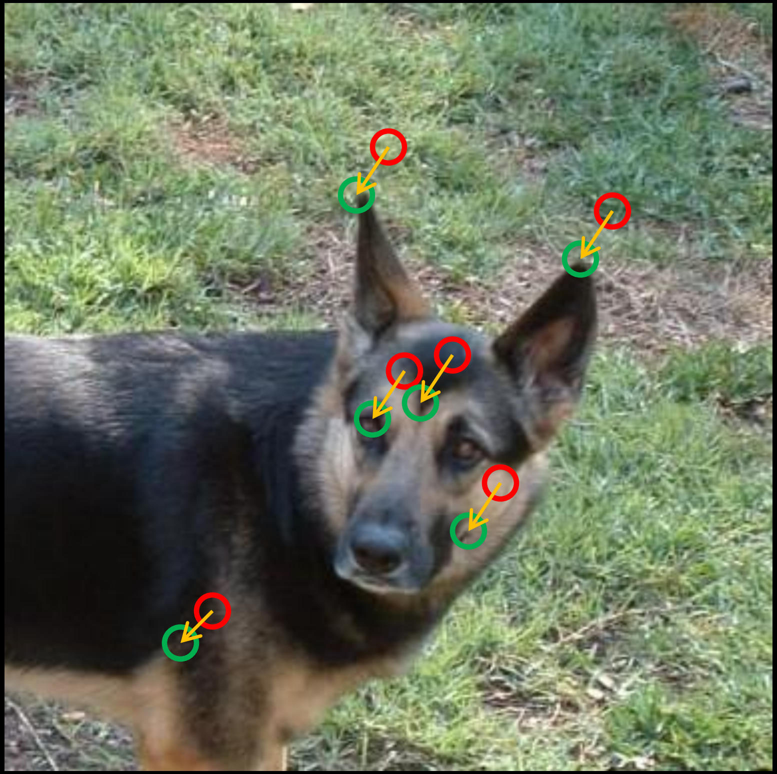
Matching Corners

- Just go for the closest point



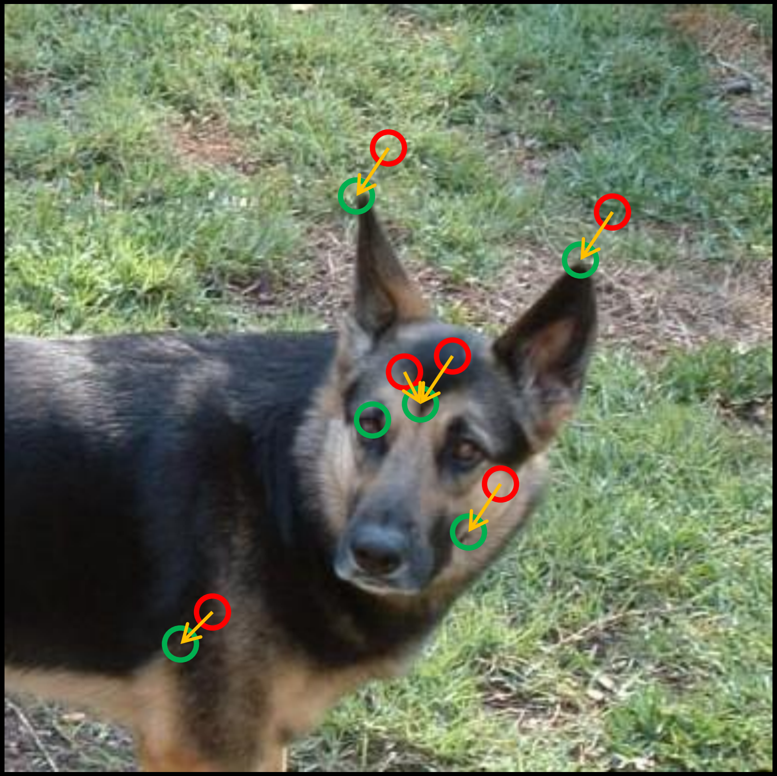
Matching Corners

- Let's introduce more corners



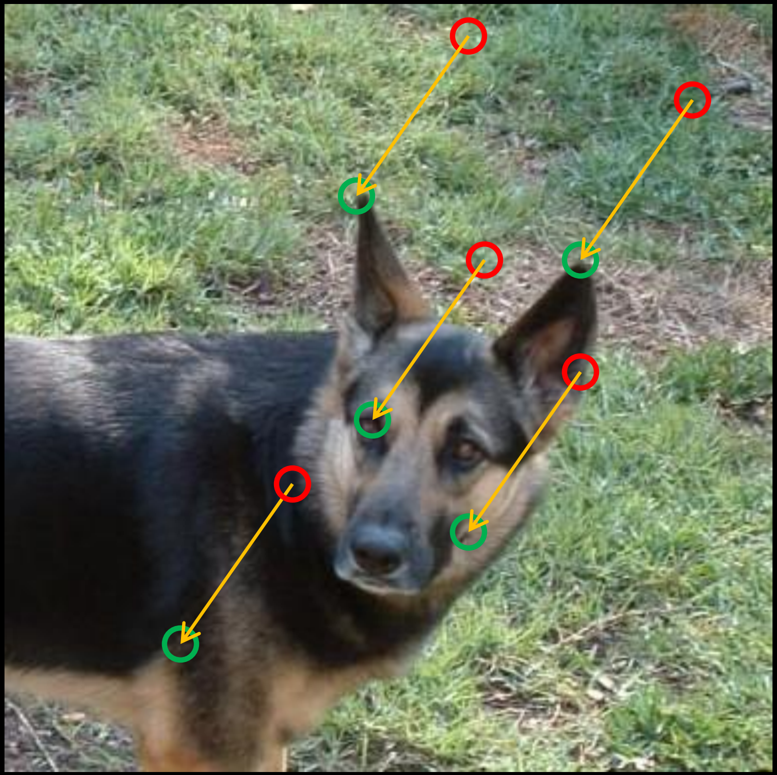
Matching Corners

- Let's introduce more corners



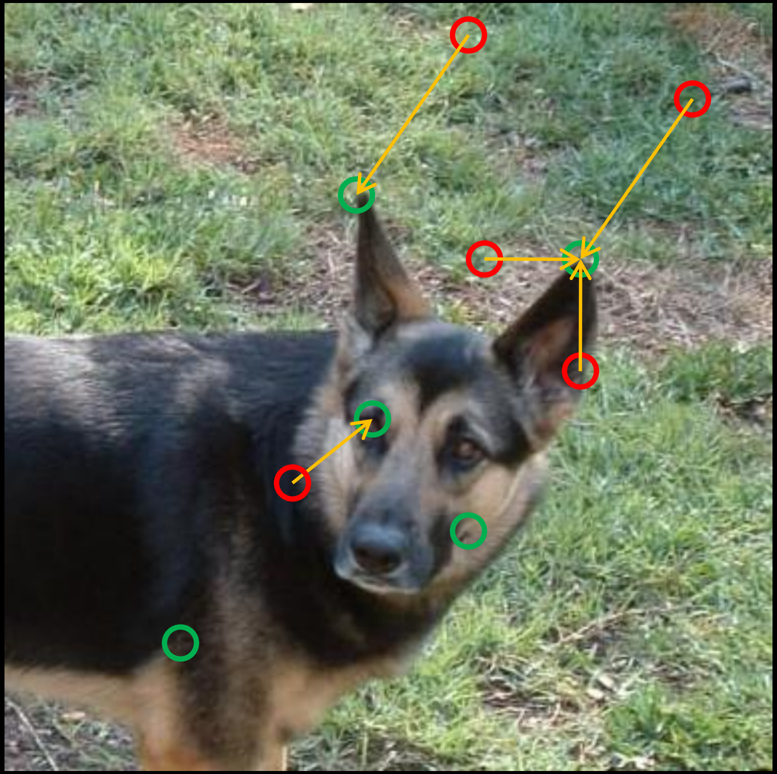
Matching Corners

- Or a larger shift



Matching Corners

- Or a larger shift



Descriptors

- Why just use distance in 2-D when we have so much more information at each patch.

- Compute an n-D "descriptor" that describes the patch.

- Use the closest point in the n-D space instead.

Descriptors

- Naive descriptor: Just read a square block of pixels out of the image around the point
  - Invariant to translation, but
  - What happens if you rotate the camera?
  - What happens if you zoom?
  - The scene gets brighter or the exposure changes?
  - The image is very noisy?
  - You view the same object from an oblique angle?
- This is what ImageStack does currently

Descriptors

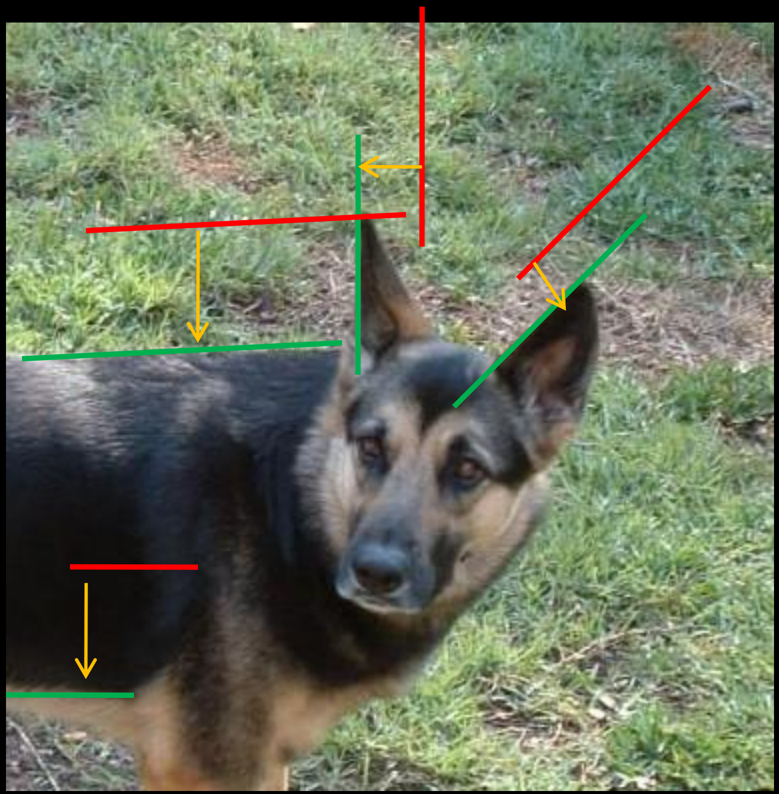
- SIFT is the gold standard descriptor
  - Check wikipedia for details of all the various filters
  - Usually massive overkill
- SURF is faster and still very good
  - Still overkill for most alignment apps
- These are both more suitable for recognition

Edge Features

- Point based matches give you 2 constraints per match
- Edge based matching can still give you 1 per match
- Edges can be more common in many environments than corners
- Harder to extract
- Harder to compute descriptors for

Matching Edges

Matching Edges



Alignment

- We've discussed means of generating correspondences between images
- How do we turn this into an actual motion model between images?
- Least Squares + RANSAC!

Least Squares

- $\mathbf{a}_i$ are the locations of points in the first image
- $\mathbf{b}_i$ are the locations of points in the second image
- We want a function **m** that maps from **a** to **b**
- We need to calculate **m** from $\mathbf{a}_i$ and $\mathbf{b}_i$

Least Squares

- If **m** is a matrix **M**, this is easy, we can use least squares.
- $\mathbf{A} = (\mathbf{a}_0\ \mathbf{a}_1\ \mathbf{a}_2\ \mathbf{a}_3\ \ldots)$
- $\mathbf{B} = (\mathbf{b}_0\ \mathbf{b}_1\ \mathbf{b}_2\ \mathbf{b}_3\ \ldots)$
- $\mathbf{MA} = \mathbf{B}$
- $\mathbf{MAA^T} = \mathbf{BA^T}$
- $\mathbf{M} = \mathbf{BA^T(AA^T)^{-1}}$

Least Squares

- $\mathbf{M = BA^T(AA^T)^{-1}}$
- $\mathbf{BA^T}$ and $\mathbf{AA^T}$ can be computed incrementally
- 2x2 matrix inverse is easy
- O(1) memory use

- This is easy and simple to code. DO NOT:
  - Use a fancy linear algebra library
  - Do something uneccesary like an SVD
- ImageStack includes a simple header file that implements this (see LinearAlgebra.h)

Least Squares

- What kind of motion do you get from a 2x2 matrix?

- Hey wait... least squares was pretty useless...

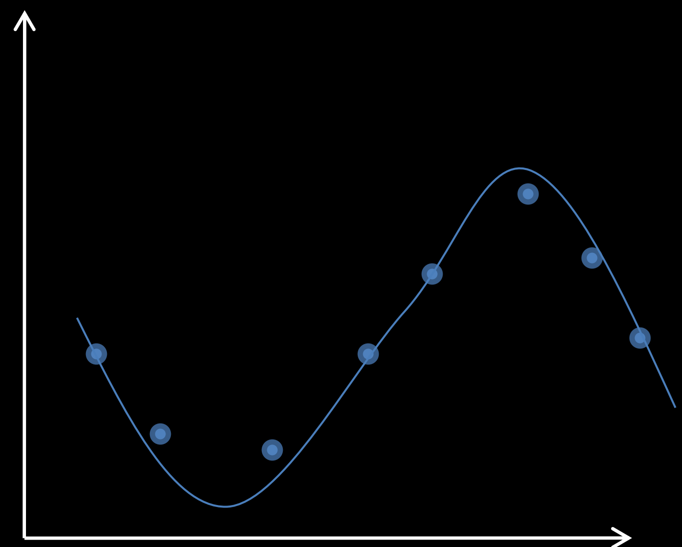
Least Squares

- How can we augment this to account for translation?
- Stick a 1 on the end of $\mathbf{a}_i$
- **M** is now 2x3 instead of 2x2

- This approach is the tip of the iceberg
- You can solve any **Mf(A) = B** where f is nonlinear the same way

Least Squares

- Eg, fitting a quintic polynomial is linear
- $\mathbf{a}_i = [1\ x_i\ x_i^2\ x_i^3\ x_i^4\ x_i^5]$
- $\mathbf{b}_i = [y_i]$
- **M** is the best 6x1 matrix
- What is f?



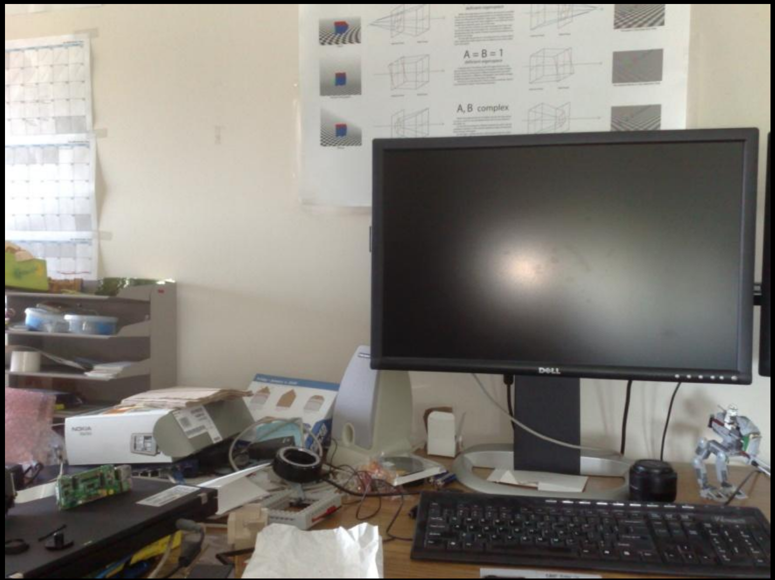
Least Squares

- So we're allowed to mess with the **a** vectors
- We can also mess with the **b** vectors
- **M**f(**A**) **=** g(**B**)

- But... error is no longer minimized in the right space
  - Often doesn't matter in practice

Least Squares

- Two different images from the same location are related by a projective transform
- ... let's work through it on the board

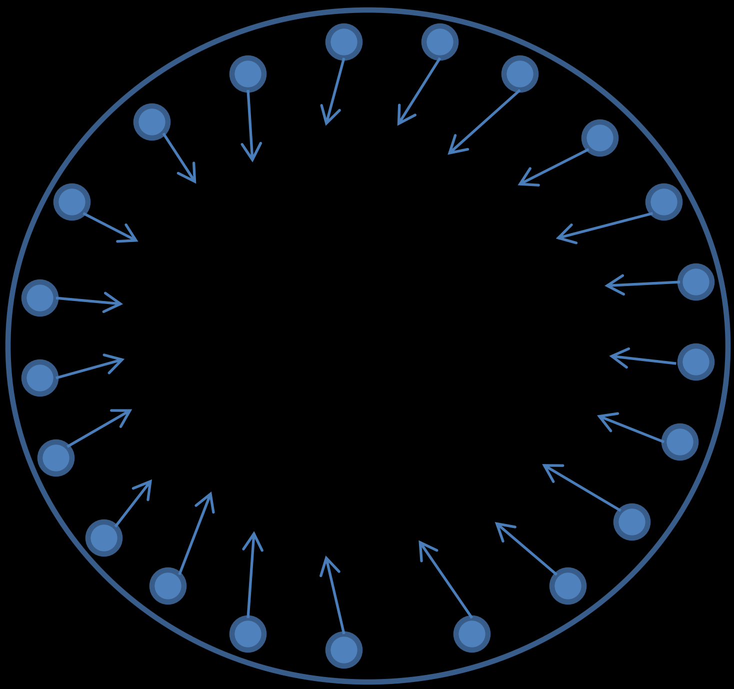


Least Squares

- Given a bunch of **a** vectors and **b** vectors...
- We can solve (for M):
  - $\mathbf{MA} = \mathbf{B}$
  - $\mathbf{M}f(\mathbf{A}) = \mathbf{B}$
  - $\mathbf{M}f(\mathbf{A}) = g(\mathbf{B})$ (but ...)
- What about:
  - $h(\mathbf{M}\, f(\mathbf{A})) = \mathbf{B}$
  - $h(\mathbf{M}_1 f(\mathbf{M}_2 \mathbf{A})) = \mathbf{B}$

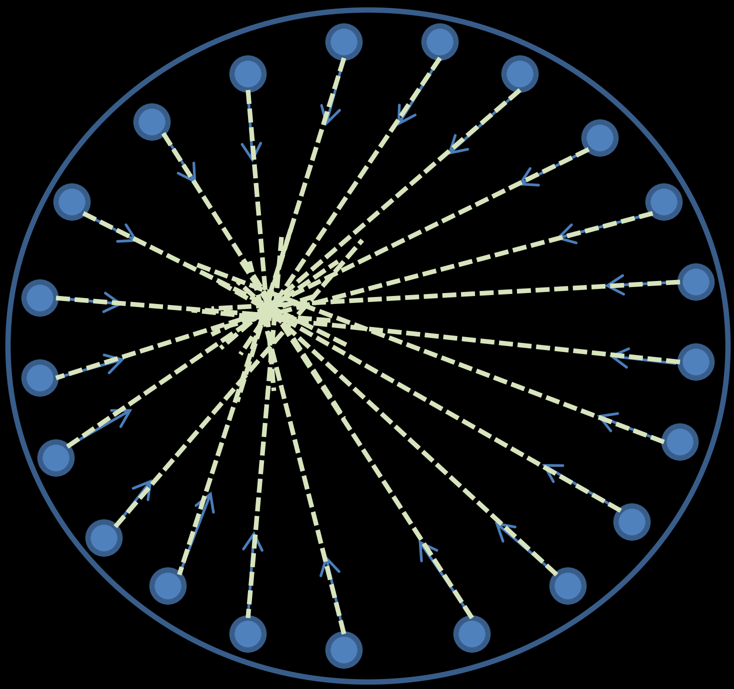
RANSAC

- What is everyone pointing at?



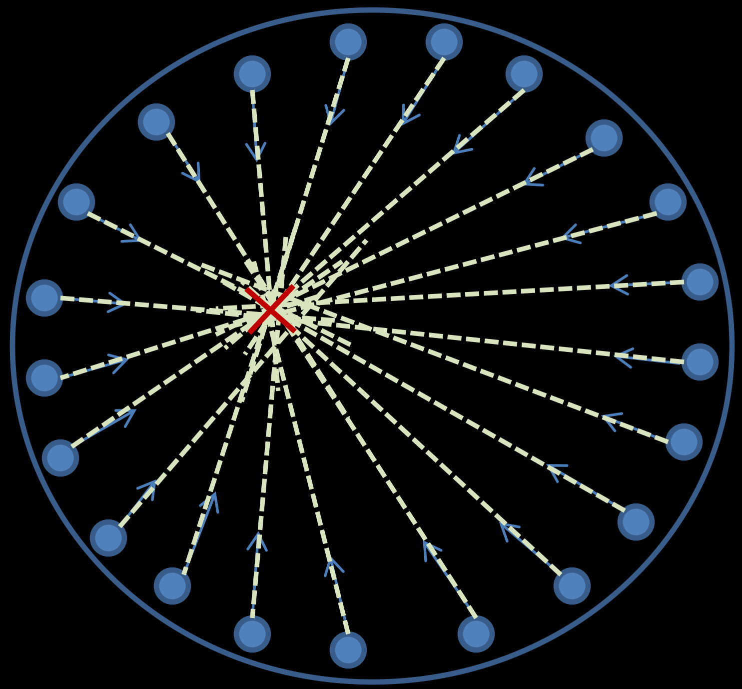
RANSAC

- Least squares will tell us!



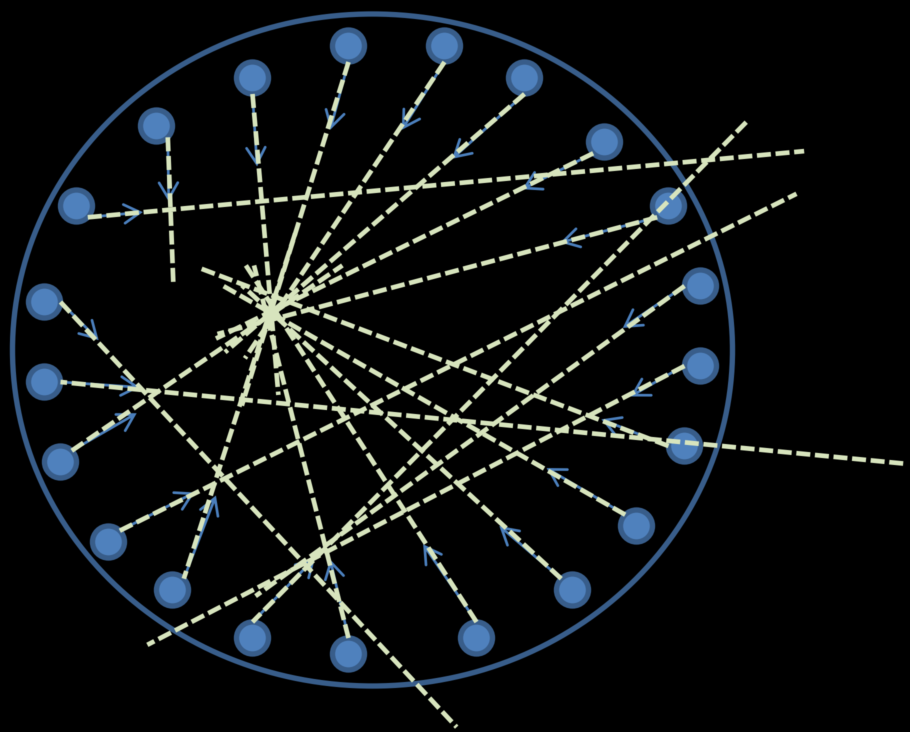
RANSAC

- X marks the spot



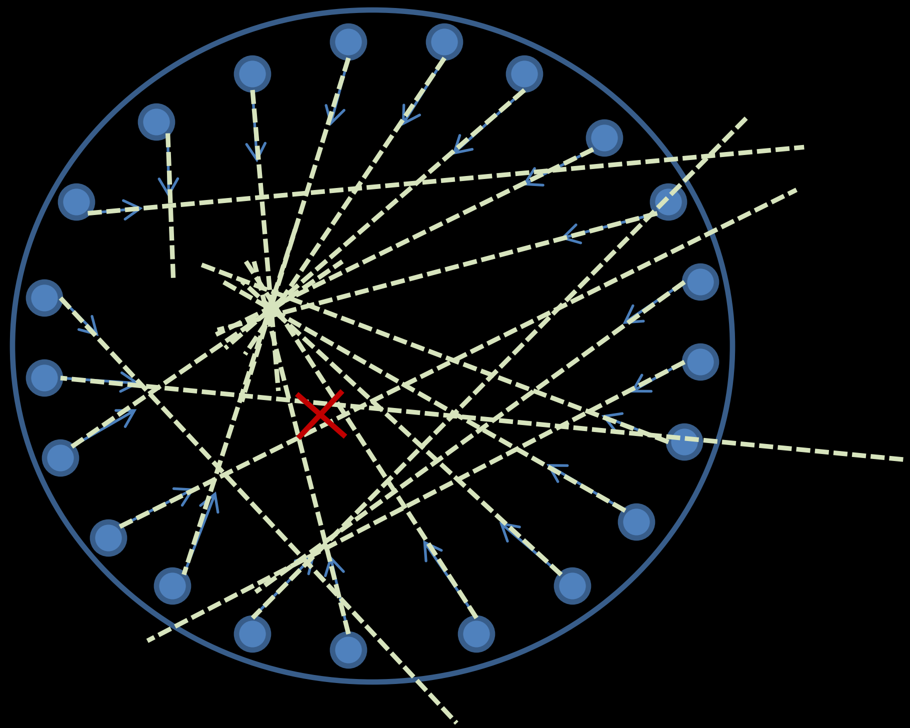
RANSAC

- What if some points are just plain wrong?



RANSAC

- Least squares can be heavily influenced by outliers.

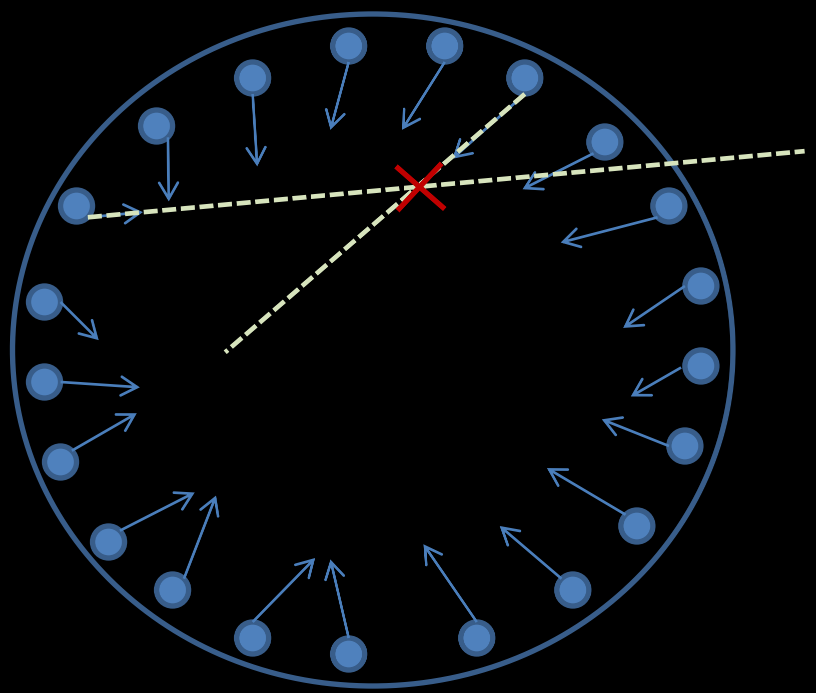


RANSAC

- Pick the **minimum** number of constraints necessary
- Hopefully none of them are bad
- Fit the model using only these
- Check how many other constraints agree
- If there aren't many inliers, we made a mistake here. Restart.
- If there are lots of inliers, fit the model again using only the inliers.

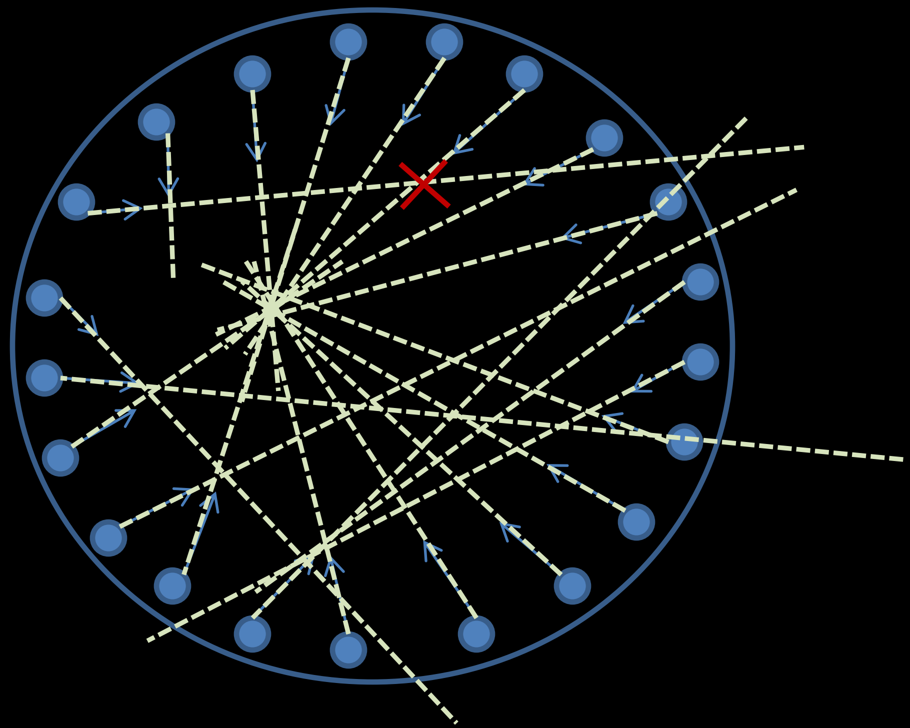
RANSAC

- Pick a the minimum number of points and hope they are inliers. Fit the model.
  - This is the **RAN**dom **SA**mple



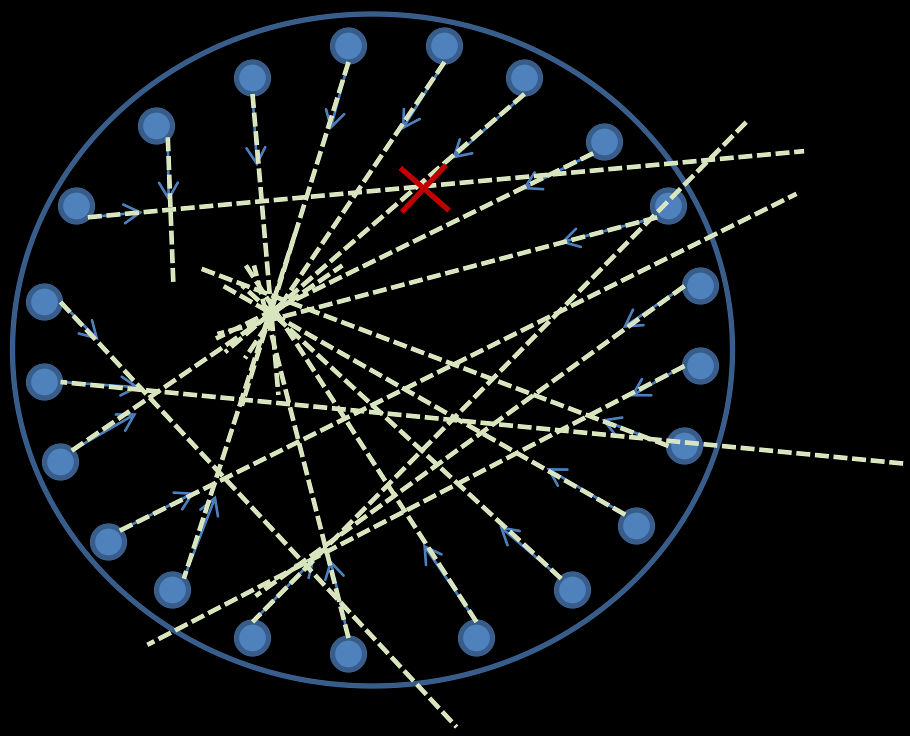
RANSAC

- Check how many other points agree.
  - This is the **C**onsensus



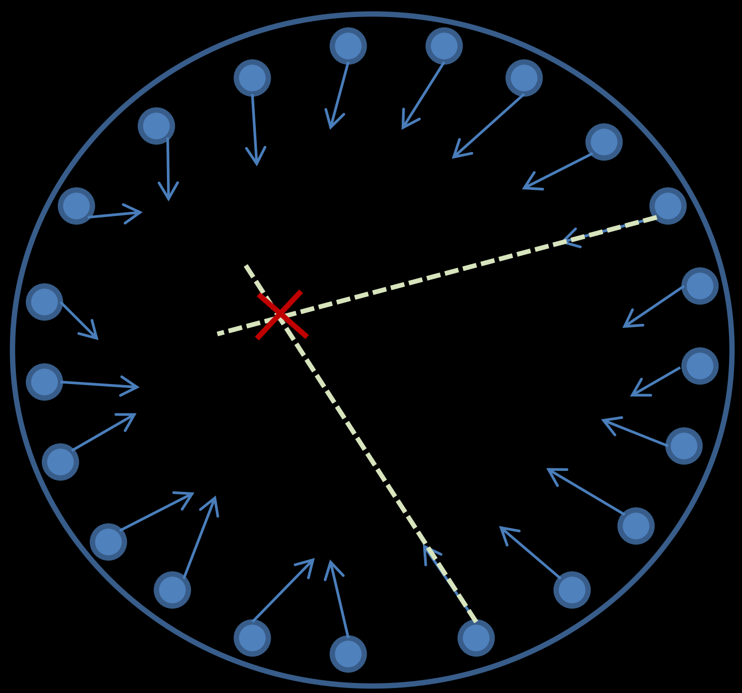
RANSAC

- Hrm, in this case nearly nobody agrees. We'd better restart with a different random pair of constraints.



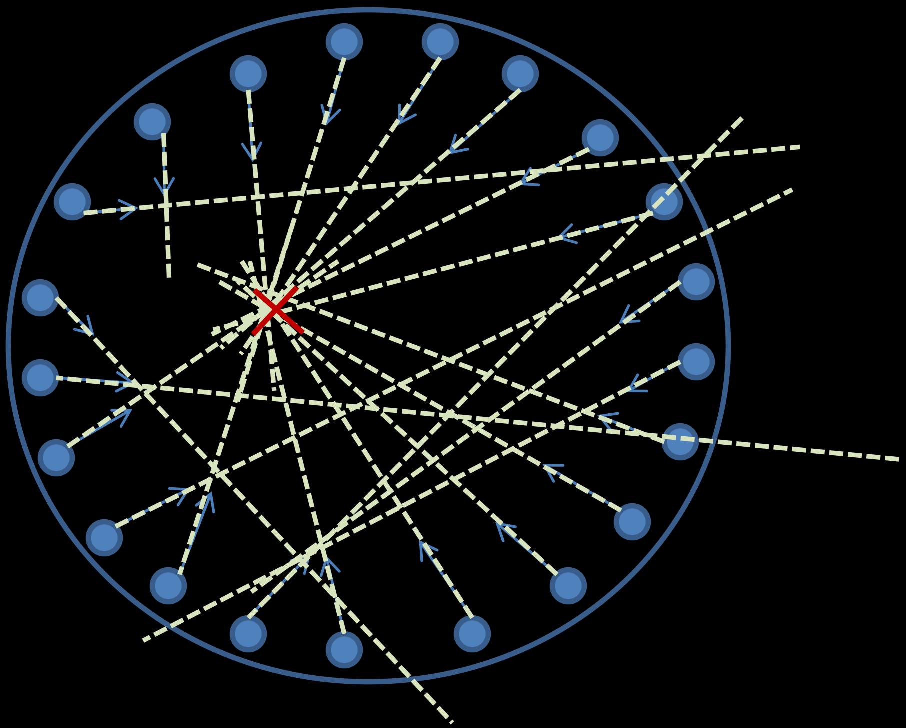
RANSAC

- **RAN**dom **SA**mple



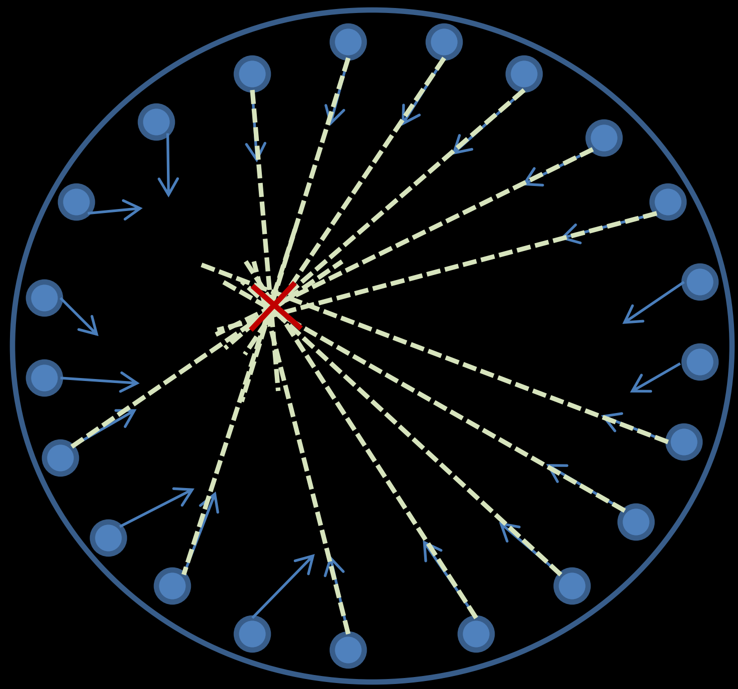
RANSAC

- **C**onsensus. This time nearly everyone agrees.



RANSAC

- Drop the outliers and refit the model for accuracy



RANSAC

- How many iterations do we need?

Computer Vision in One Slide

- Extract some features from some images
- Use these to formulate some linear constraints (even if the problem is non-linear)
- Solve a linear system of equations using RANSAC and least squares

Assignment 3

- Let’s look at ImageStack’s -align operator.

Assignment 3

- Make ImageStack’s align operator better.
- The current one doesn’t work at all (buggy)
- I’ve posted a fixed version on the assignment webpage that works for a few of the test cases.
- Graded on how many of our test cases you can successfully align + not taking too long
- Due in one week, as usual.

Alternative to RANSAC: Voting schemes

- You’re usually solving for a model with some number of parameters
- If the number of parameters is small, can discretize parameter space
- Each constraint votes for some number of parameters
- Look for local maxima in parameter space
- Known as a Hough transform, particularly good for line detection

Hough transform exercise

- Everyone think of a number from one to ten

Hough transform exercise

- If your number is even you'll be an inlier, if it was odd, you're an outlier.
- Outliers: pick a pair of random numbers from the set [0, 1, 2, 3, 4].
- Inliers: Your pair of numbers is 2, 4
- When queried, tell me some simple linear combination of your two numbers
  - e.g: 2 times my first number plus my second number is 8
- Let's compare Hough transform to RANSAC

Hough Transform v RANSAC

- Hough Transform:
  - Can detect multiple models in a single pass through the input
  - Uses lots of memory
  - Sometimes parameter space is hard to sample
- RANSAC:
  - Requires many passes through the input
  - Uses little memory
  - Computes highly accurate models

How would we align two images using a Hough Transform?

- starting from a list of correspondences