ECE5554 – Computer Vision
Lecture 9a – Feature-based Alignment

Creed Jones, PhD

BRADLEY DEPARTMENT
OF ELECTRICAL & COMPUTER ENGINEERING

VIRGINIA TECH.

Course updates

- HW4 is due tonight at 11:59 PM

- HW5 (the final one) is posted
  - Due next Wednesday, August 10

- SPOT surveys on this course will open soon
  - open from August 6 through August 12
  - participation is completely anonymous and completely voluntary
  - I would appreciate your responses – especially comments that I can act on!

- Lecture 10 on Monday, August 8 will be asynchronous
  - No synchronous class session; I will be traveling
  - There will be three pre-recorded lectures, watch at your convenience
  - I will look for questions in Piazza

Final Exam will be Thursday, August 11, 8 PM to 11 PM

- The exam will be a collection of questions similar to the quiz questions, plus a few additional questions (may be a short calculation, a question requiring a few sentences in response, etc.)
- There will be a two-hour time limit (once you start) but I am designing the exam to require one hour.
- There will be a zoom session; you MUST join to get the password for the exam. I will answer any questions (for Everyone) only via the zoom chat.
- Here is the zoom link: https://virginiatech.zoom.us/j/88059665741
  Meeting ID: 880 5966 5741

Today’s Objectives

Feature-based alignment

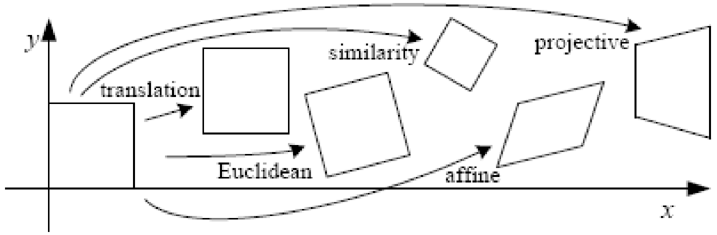
- The concept of feature-based alignment
- Relation to motion tracking
- Panography
- Review of RANSAC
- ORB
- An example

FEATURE-BASED ALIGNMENT

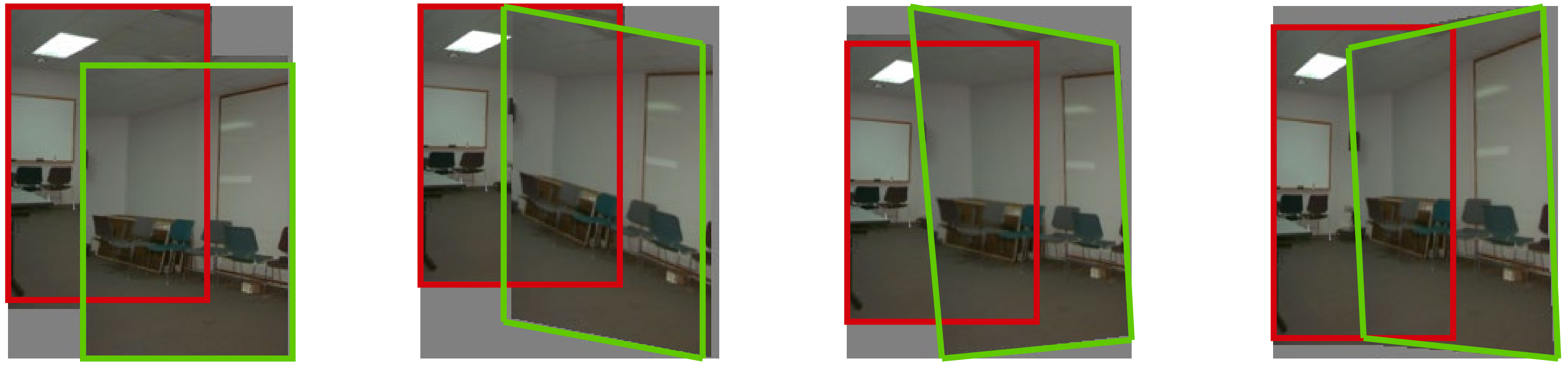
Feature-based alignment is the ability to find spatial relationships between two images of the same scene

- When we did KLT motion tracking, the assumption was that the offset between two images was small
  - But we also allowed for different points to follow different trajectories – in other words, the object could deform
- When we speak of "Feature-based alignment", we assume less deformation of the object
  - Generally only translation, rotation, scale (limited), affine and/or projective transformations
  - BUT we allow for more significant displacements from image to image

Motion models



Translation | Affine | Perspective | 3D rotation



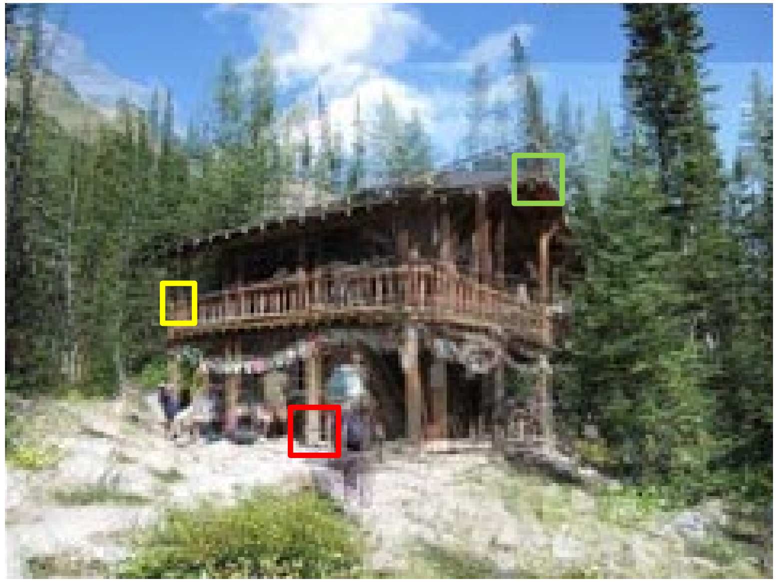
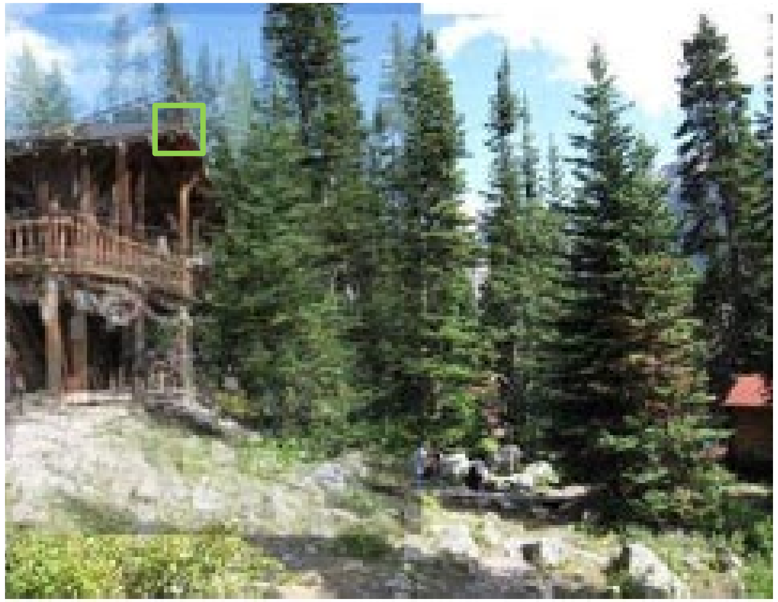
2 unknowns | 6 unknowns | 8 unknowns | 3 unknowns

One example application is stitching images together for producing a photo collage – this is called *Panography*

Figure 6.3 A simple panograph consisting of three images automatically aligned with a translational model and then averaged together.

Szeliski, p. 314

One example application is stitching images together for producing a photo collage – this is called Image stitching or *Panography*

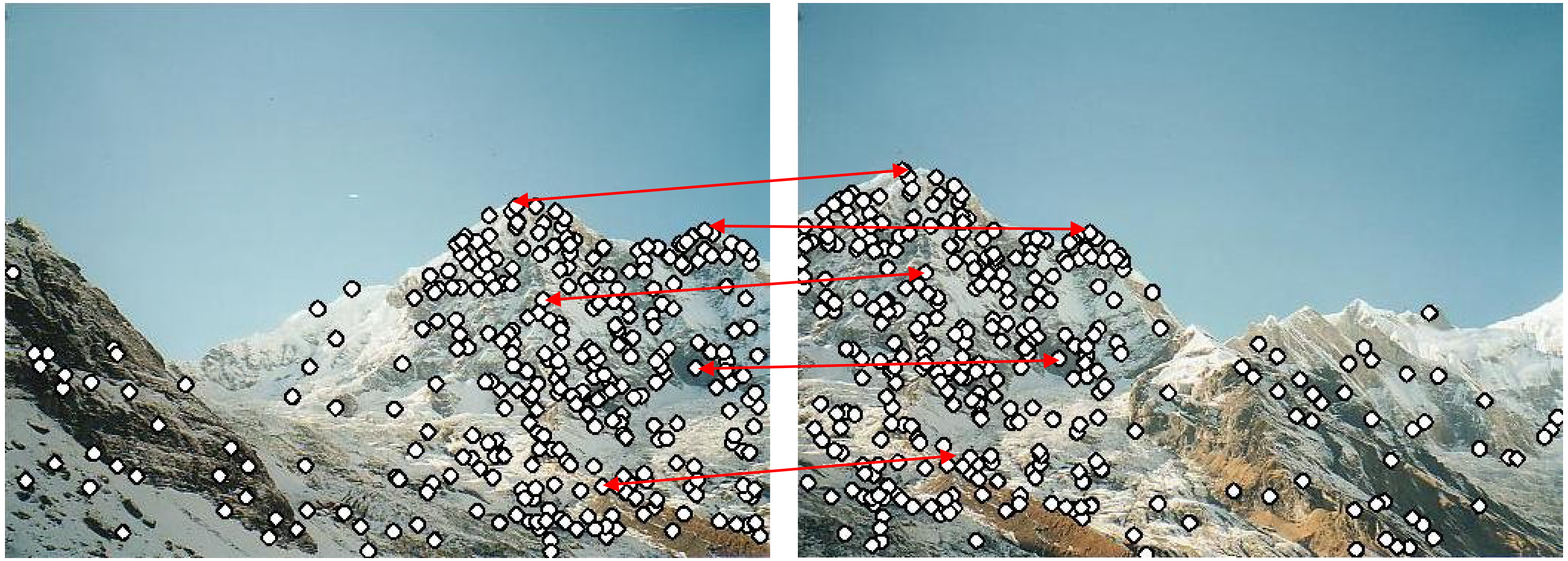
Features used to establish the positional relationship can be keypoints (corners), template match locations or output of other techniques

From Szeliski, p. 314

One example application is stitching images together for producing a photo collage – this is called Image stitching or *Panography*

More on this in the next lecture

Photo-Compositing: How would you do it manually?



One possibility:

- Find distinctive points in both images
- Identify corresponding pairs of points
- Use the correspondences to determine a 2-D geometric transformation
- Align one image to the other
- Merge them by creating a new image, averaging any pixels that overlap

The quantity that is minimized when two images are well aligned is generally a sum of squared differences in position

- If we assume only translational difference between images (no rotation), then the best alignment minimizes:

$$E_{PLS} = \sum_{i,j} \left\| \boldsymbol{t}_j + \boldsymbol{x}_{ij} - \boldsymbol{x}_i \right\|^2$$

- Where:
  - We assume a global coordinate system;
  - $\boldsymbol{t}_j$ is the position of the coordinate frame of image $\boldsymbol{j}$ in the global coordinates;
  - $\boldsymbol{x}_{ij}$ is the position of feature i in image j;
  - $\boldsymbol{x}_i$ is the average location of feature i in the global coordinates.

In the real world, not all features will be accurately located in all images; incorrect matches form *outliers* that will skew the computation of the SSE

- If a single feature is incorrectly located far from its proper position in an image, the entire found frame of reference will be skewed towards it
- Using the square of the errors makes this problem even worse

- This is a great place to use RANSAC – RANdom SAmple Consensus

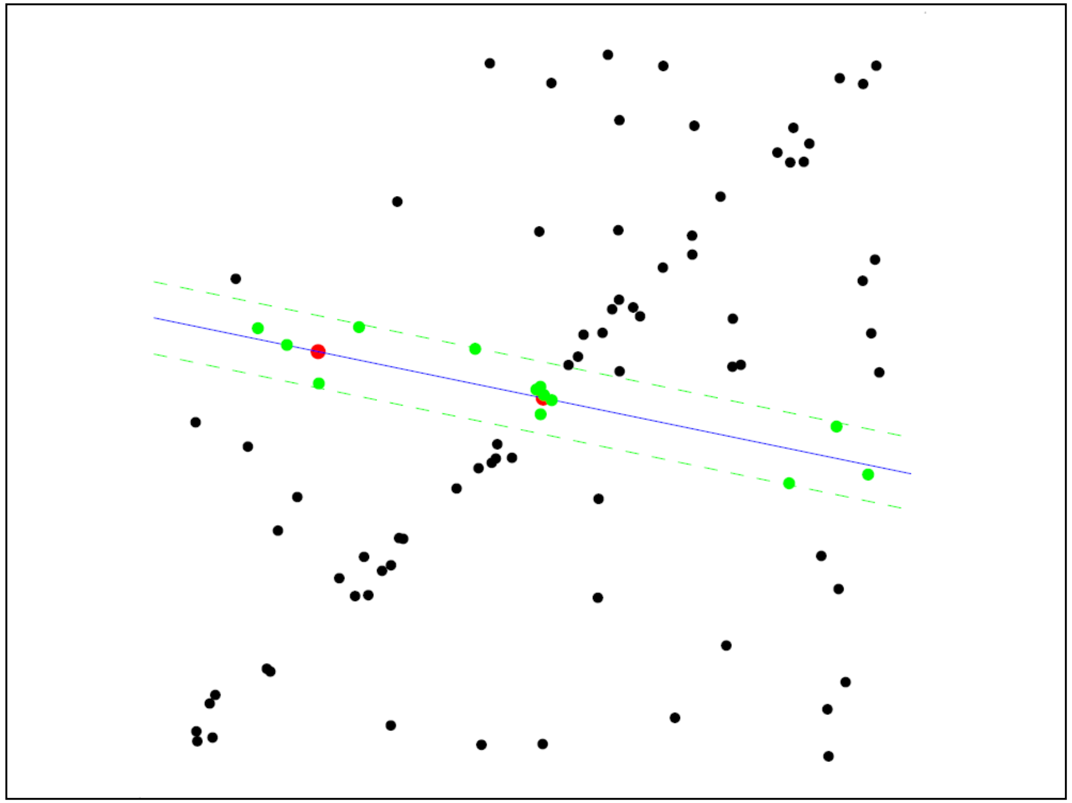
RANSAC: General form

- RANSAC loop:
    1. Randomly select a *seed group* of points on which to base our transformation estimate
       (e.g., a group of matches)
    2. Compute transformation from seed group
    3. Find *inliers* to this transformation from the remaining (non-seed) points
    4. If the number of inliers is sufficiently large, re-compute estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers

Slide credit: Grauman

RANSAC for line fitting example

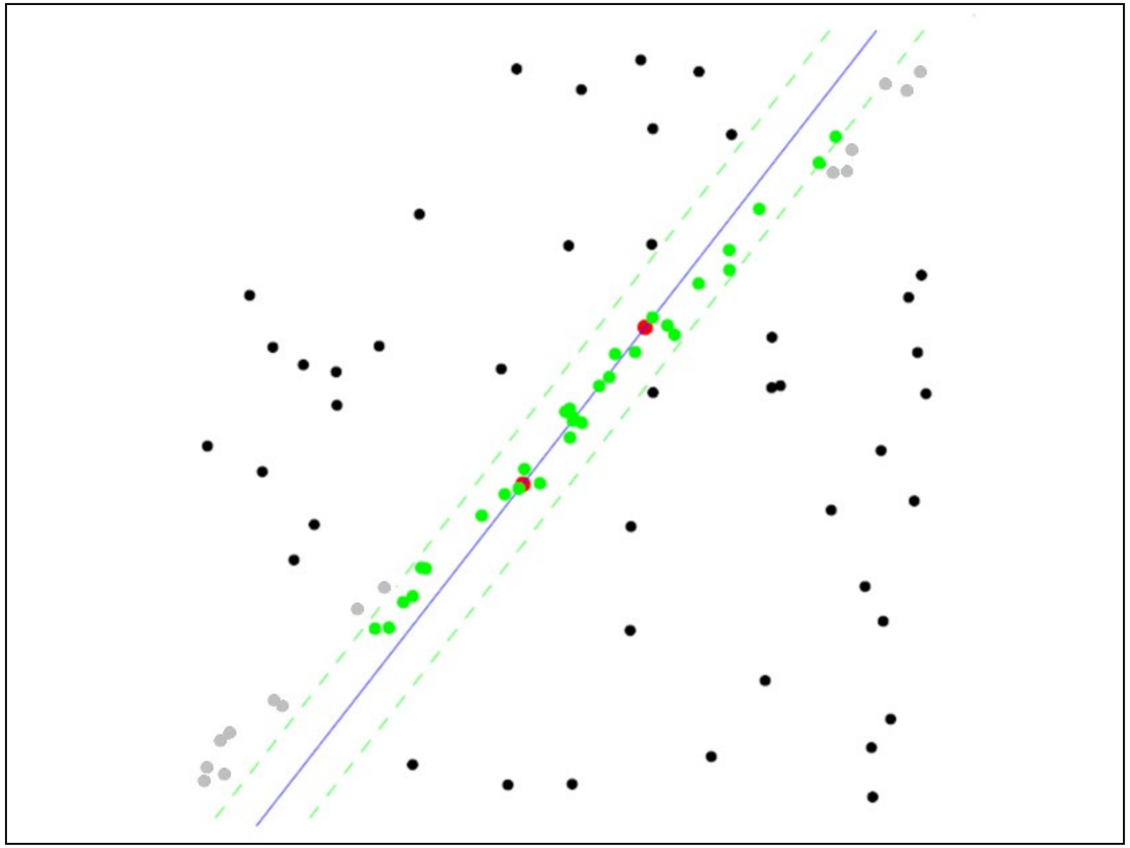


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error values
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC pros and cons

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Lots of parameters to tune
  - Doesn't work well for low inlier ratios (too many iterations, or can fail completely)
  - Can't always get a good initialization of the model based on the minimum number of samples

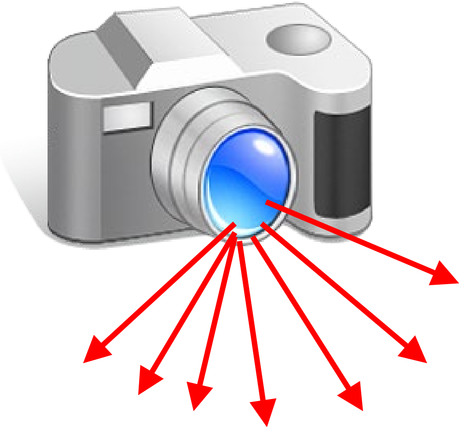


RANSAC can be used to clean up our list of correspondences prior to using the SSQ minimization to calculate frames of reference

- On the left, all feature correspondences detected
- On the right, result of removing outliers using RANSAC

Panography or Compositing: another example

. . .



Slide credit: Seitz, Grauman

We discussed the Scale-Invariant Feature Transform, SIFT, as a way of identifying robust keypoints – but it has some disadvantages

- SIFT feature extraction can be expensive, computationally
- It is patented – although the patent expired in 2020
- There is support for it in OpenCV

- Alternatives have been developed:
  - FAST
  - BRIEF

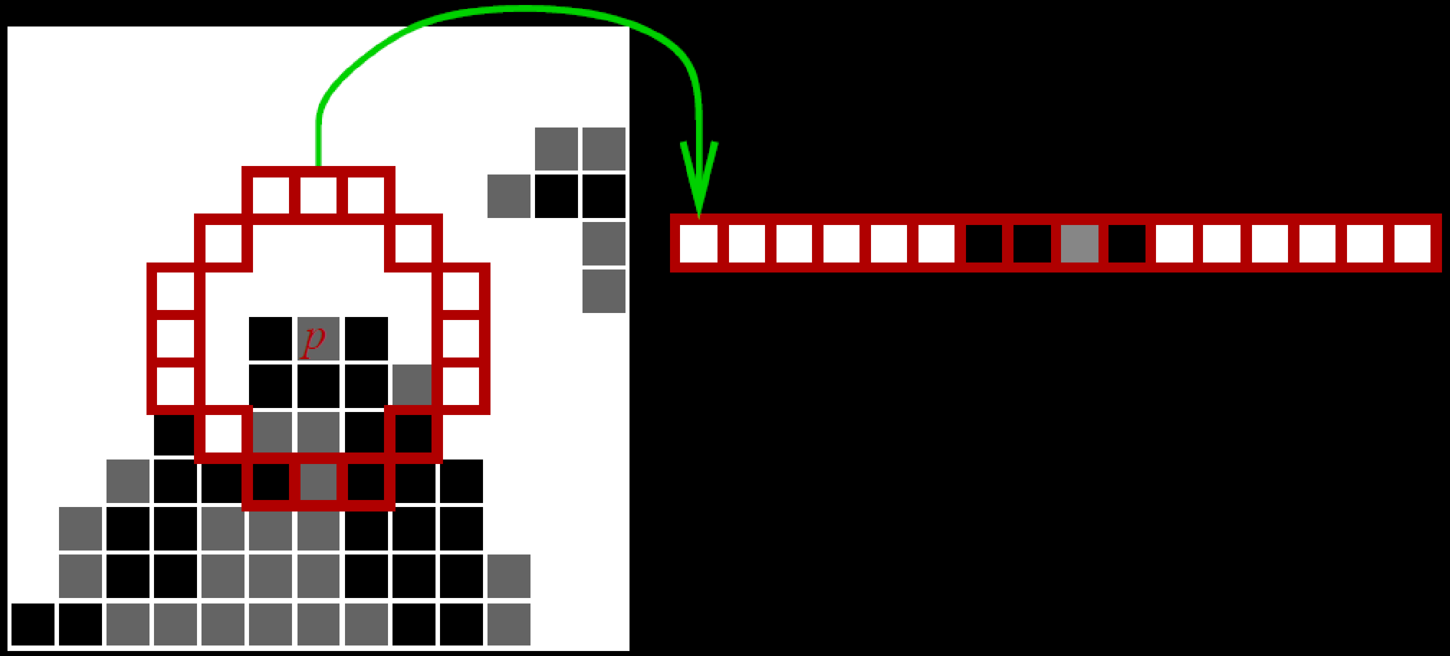
  - ORB (Oriented FAST and Rotated BRIEF)

The heart of FAST is forming a vector of rough brightness differences around a circular path

Radii of 9 and 12 are common

- From Rosten 2008 presentation
- https://www.edwardrosten.com/work/fast.html

FAST feature detection (version 2)



- Pixels are either:
  - Much brighter.
  - Much darker.
  - Similar.
- Represent ring as a ternary vector.
- Classify vectors using segment test.

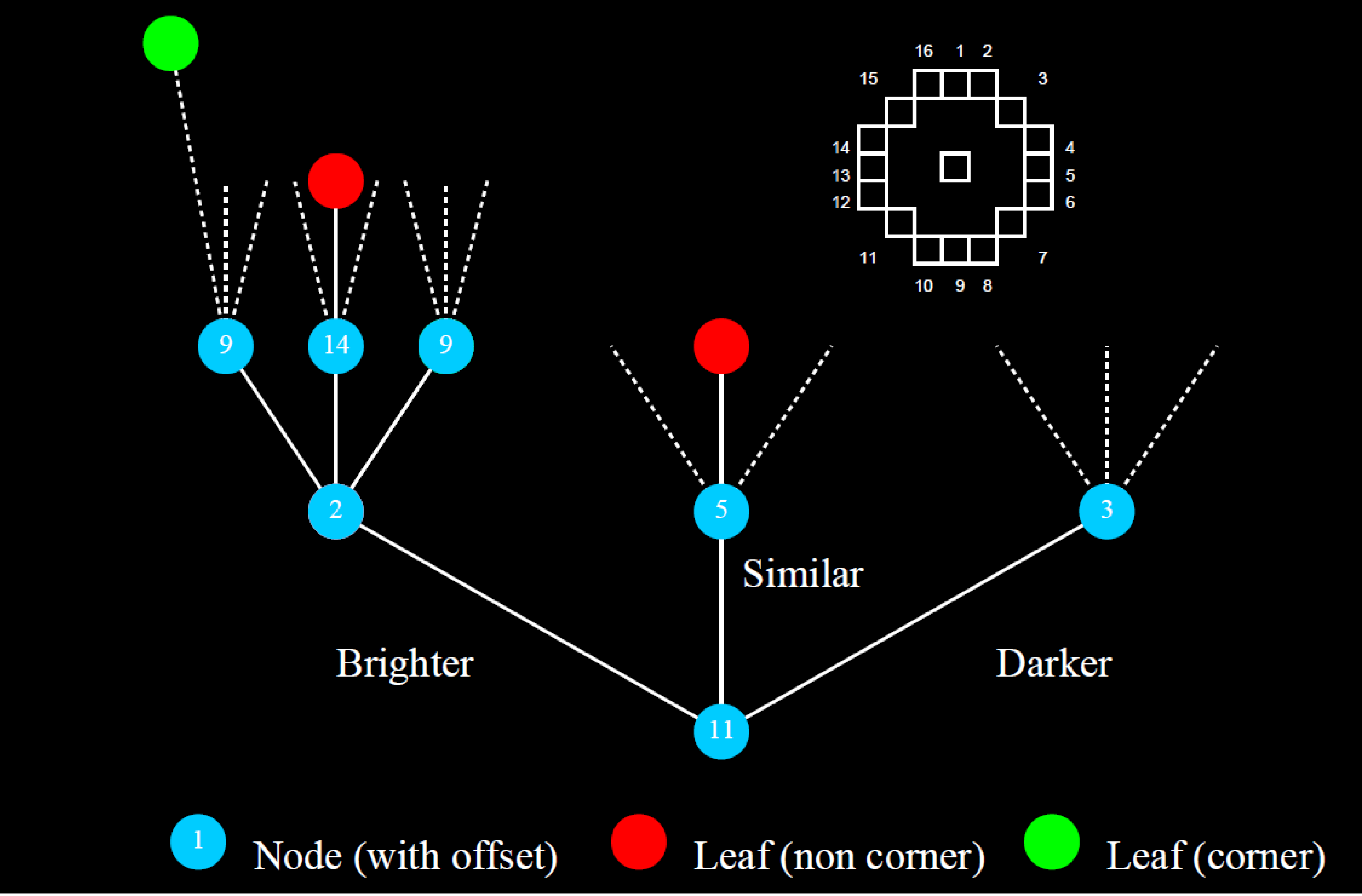
Each vector drives a decision tree to separate corners from non-corners

The corner detection is indeed fast, but not as discriminating as Harris or Shi-Tomasi

FAST will classify points as corner or non-corner but does not produce a scalar measure of corner likelihood

- From Rosten 2008 presentation

Example tree

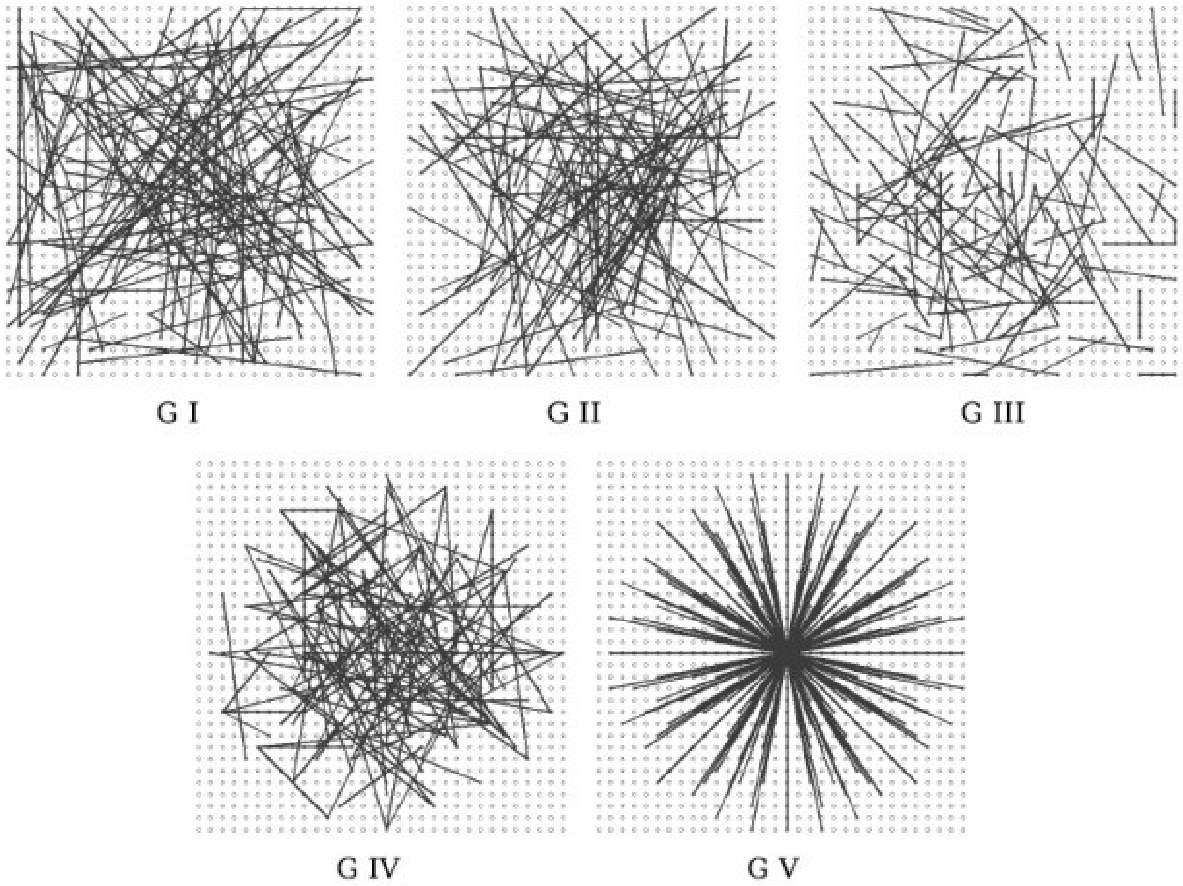


BRIEF (Binary Robust Independent Elementary Features) encodes an image patch as binary indications of order of two (selected) pixel locations in the patch

- For two pixel locations m and n, we compare $I(x_m, y_m)$ and $I(x_n, y_n)$
- The one-bit result of this comparison is:

$$b_{mn} = \begin{cases} 1, & I(x_m, y_m) < I(x_n, y_n) \\ 0, & I(x_m, y_m) \geq I(x_n, y_n) \end{cases}$$

- The bits are concatenated into a descriptor for this image patch
- Surprisingly robust
  - Insensitive to overall brightness

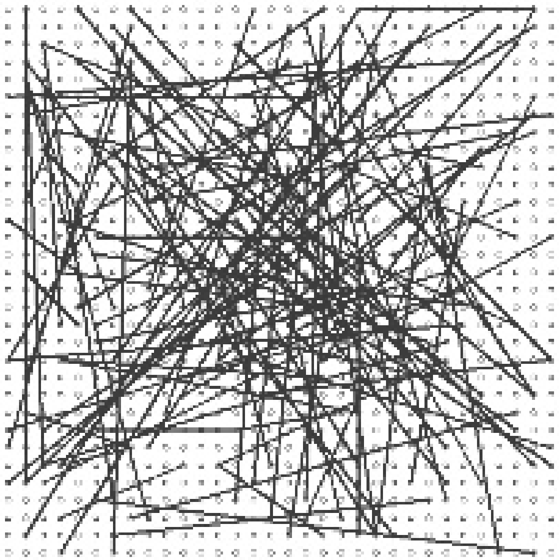


https://medium.com/software-incubator/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6

BRIEF:

Binary Robust Independent Elementary Features

- Random selection of pairs of intensity values.
- Fixed sampling pattern of 128, 256 or 512 pairs.
- Hamming distance to compare descriptors (XOR).



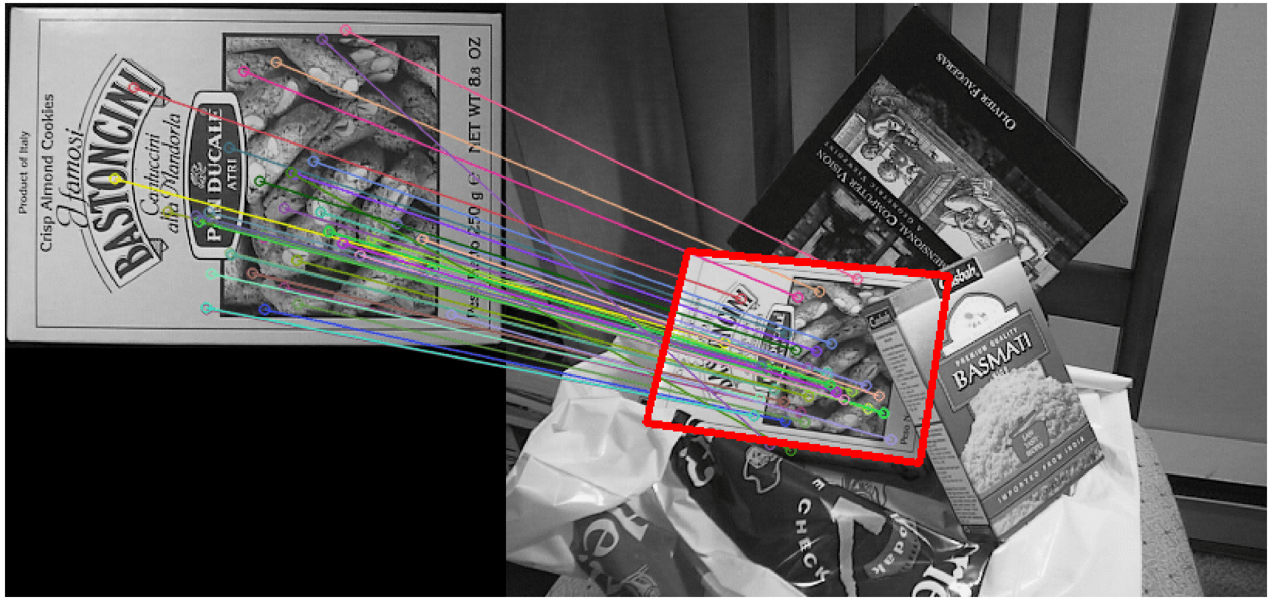
Citations:
149 (2012)

M. Calonder, V. Lepetit, C. Strecha, P. Fua, BRIEF: Binary Robust Independent Elementary Features, 11th European Conference on Computer Vision, 2010.

ORB is based on FAST with the addition of BRIEF descriptors

- Find possible corner points using FAST
- Locate the intensity-weighted centroid of the patch
- The vector from the center of the “corner” (the patch) to the centroid is considered the “orientation” of the patch
- Rotate the patch so that the orientation is fixed
- Compute the BRIEF features at points in the patch that were developed by learning on a large set of images
- The BRIEF vector is complete

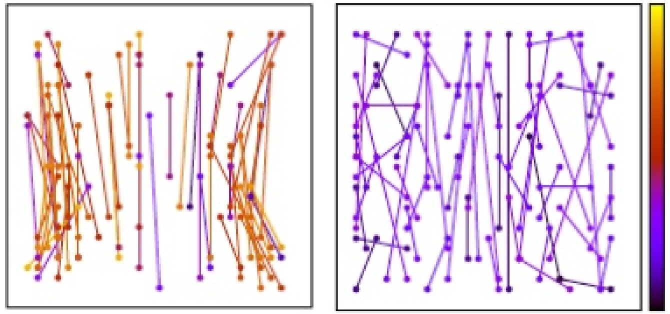
BRIEF features are good for recognition – especially as part of an overall geometric constraint strategy (think RANSAC)



ORB:

Oriented FAST and Rotated BRIEF

- Select a good set of pairwise comparisons: minimize correlation under various orientation changes



High variance   High variance + uncorrelated

A possible strategy for feature-based alignment of two images

- Find keypoints in images A and B
  - using Shi-Tomasi, FAST, Harris or other method
- For each keypoint:
  - characterize it using BRIEF, SIFT or other method
  - match each keypoint in A with its closest characteristic in B
- There will be mismatches
  - Because of changes in the images
  - Because some points just look alike
- Use a model of allowable geometric transformation from A to B
- Use RANSAC or other method to identify “inliers” – the true correspondences

- The best model for alignment is determined by the inliers

There is a really good example of feature-based alignment at this link

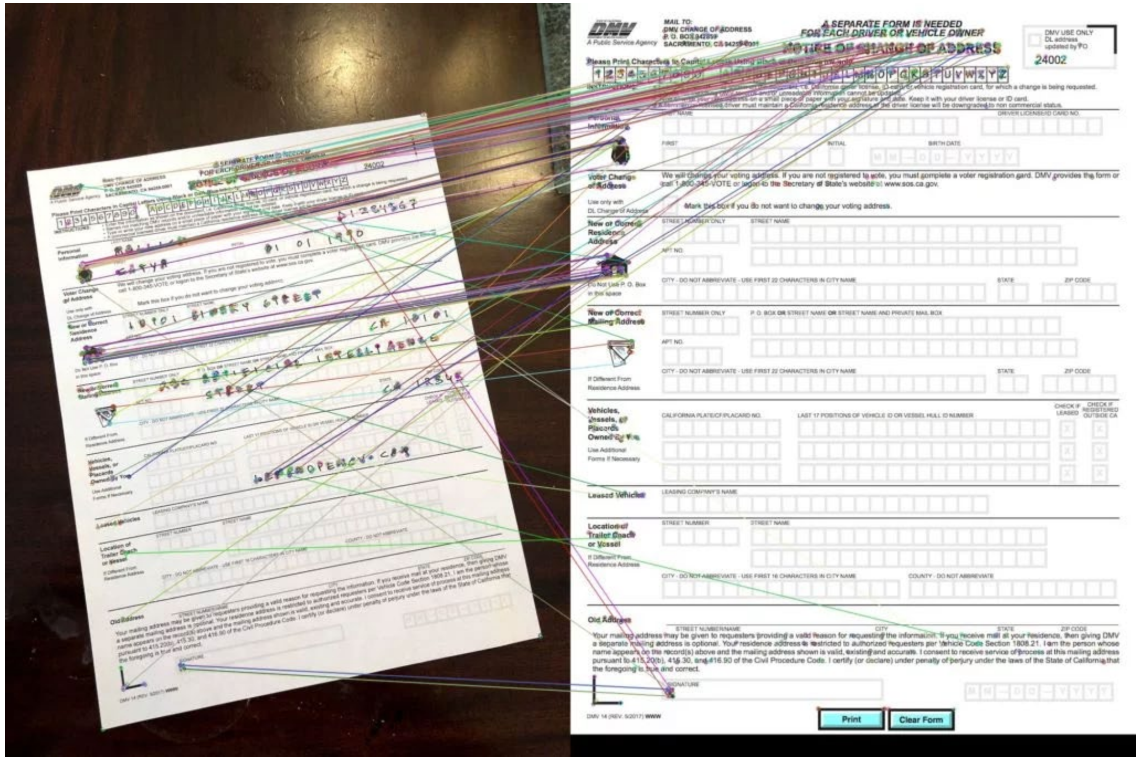
- The application is to match points on an empty form with a corresponding image of a completed form – but there is a geometric transformation between the two

- https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/

ORB is used to find keypoints in both images; correspondences are filtered by the required geometric relationship

- We allow scale, translation and rotation
  - Not skew; we assume that the paper is on a flat surface both times

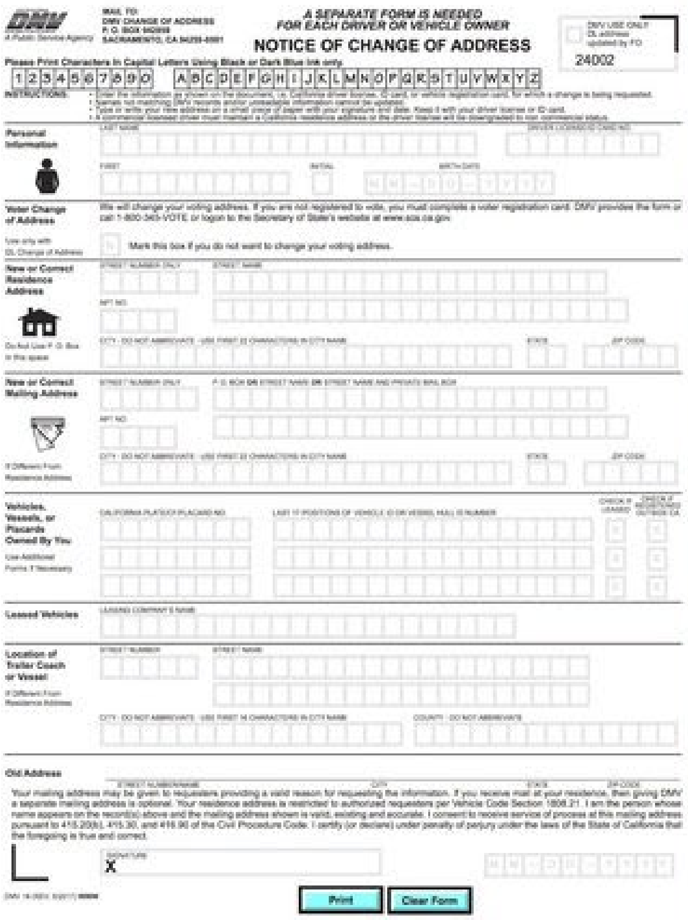
- Using several functions from OpenCV:
  - The ORB class
  - DescriptorMatcher
  - findHomography



```python
from __future__ import print_function
import cv2
import numpy as np

MAX_FEATURES = 500
GOOD_MATCH_PERCENT = 0.15

def alignImages(im1, im2):

  # Convert images to grayscale
  im1Gray = cv2.cvtColor(im1, cv2.COLOR_BGR2GRAY)
  im2Gray = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY)

  # Detect ORB features and compute descriptors.
  orb = cv2.ORB_create(MAX_FEATURES)
  keypoints1, descriptors1 = orb.detectAndCompute(im1Gray, None)
  keypoints2, descriptors2 = orb.detectAndCompute(im2Gray, None)

  # Match features.
  matcher =
cv2.DescriptorMatcher_create(cv2.DESCRIPTOR_MATCHER_BRUTEFORCE_HAMMING)
  matches = matcher.match(descriptors1, descriptors2, None)

  # Sort matches by score
  matches.sort(key=lambda x: x.distance, reverse=False)

  # Remove not so good matches
  numGoodMatches = int(len(matches) * GOOD_MATCH_PERCENT)
  matches = matches[:numGoodMatches]

  # Draw top matches
  imMatches = cv2.drawMatches(im1, keypoints1, im2, keypoints2, matches, None)
  cv2.imwrite("matches.jpg", imMatches)

  # Extract location of good matches
  points1 = np.zeros((len(matches), 2), dtype=np.float32)
  points2 = np.zeros((len(matches), 2), dtype=np.float32)

  for i, match in enumerate(matches):
    points1[i, :] = keypoints1[match.queryIdx].pt
    points2[i, :] = keypoints2[match.trainIdx].pt

  # Find homography
  h, mask = cv2.findHomography(points1, points2, cv2.RANSAC)

  # Use homography
  height, width, channels = im2.shape
  im1Reg = cv2.warpPerspective(im1, h, (width, height))

  return im1Reg, h


if __name__ == '__main__':

  # Read reference image
  refFilename = "form.jpg"
  print("Reading reference image : ", refFilename)
  imReference = cv2.imread(refFilename, cv2.IMREAD_COLOR)

  # Read image to be aligned
  imFilename = "scanned-form.jpg"
  print("Reading image to align : ", imFilename);
  im = cv2.imread(imFilename, cv2.IMREAD_COLOR)

  print("Aligning images ...")
  # Registered image will be resotred in imReg.
  # The estimated homography will be stored in h.
  imReg, h = alignImages(im, imReference)

  # Write aligned image to disk.
  outFilename = "aligned.jpg"
  print("Saving aligned image : ", outFilename);
  cv2.imwrite(outFilename, imReg)

  # Print estimated homography
  print("Estimated homography : \n",  h)
```

https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/

As a result, the completed form is geometrically transformed to match the layout of the blank form



Today’s Objectives

Feature-based alignment

- The concept of feature-based alignment
- Relation to motion tracking
- Panography
- Review of RANSAC
- ORB
- An example