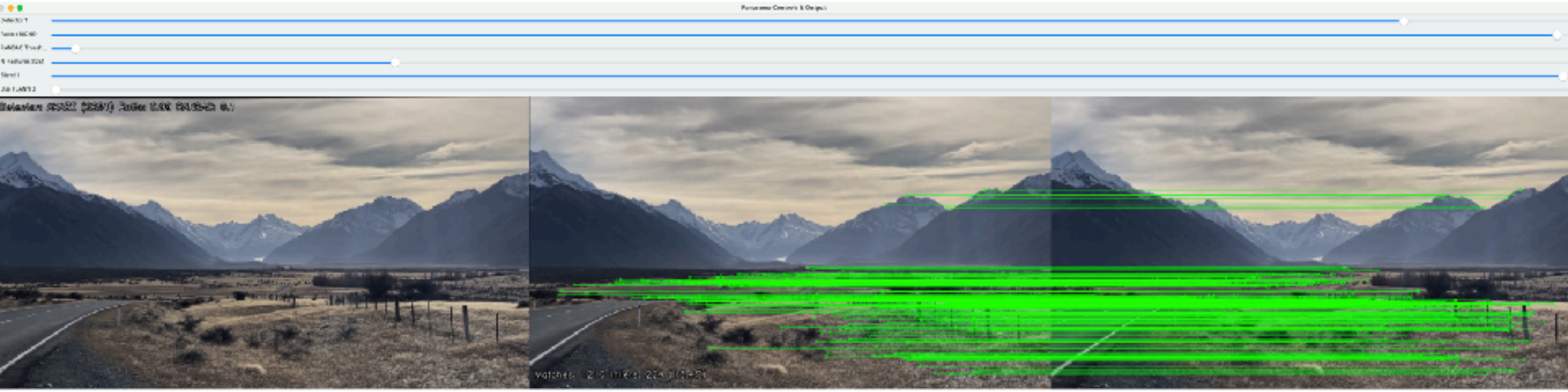
Visual Computing I:

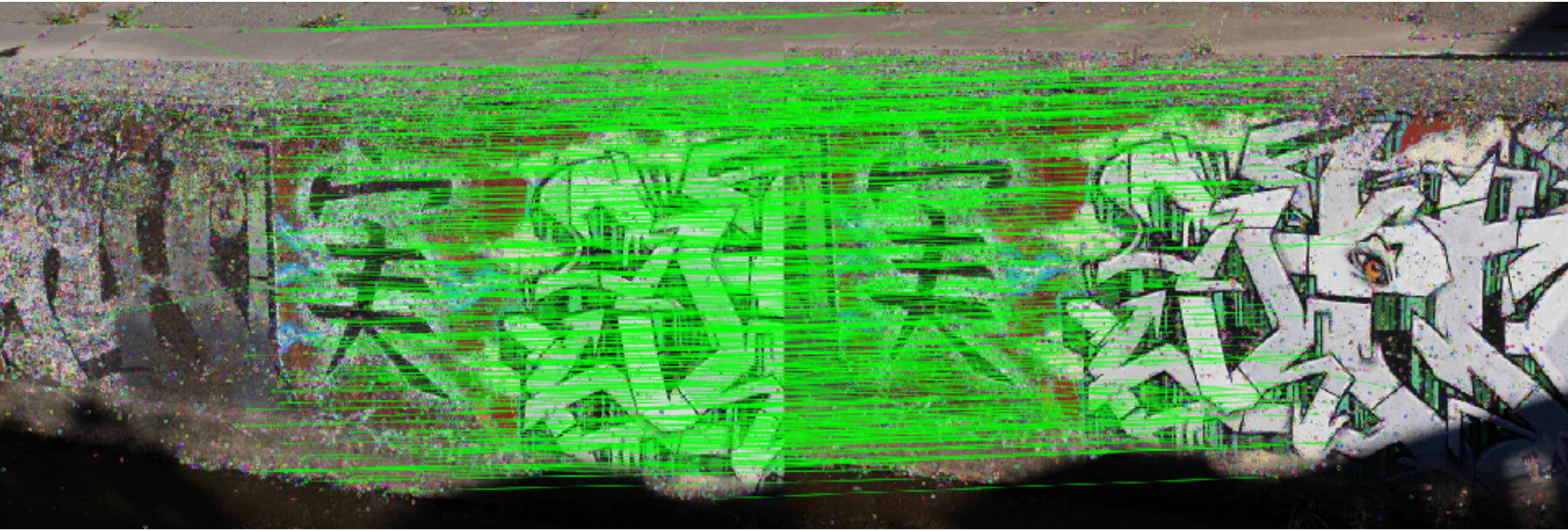
Interactive Computer Graphics and Vision



Feature Matching, Image Stitching, Homographies

Stefanie Zollmann and Tobias Langlotz

How to find correspondences?

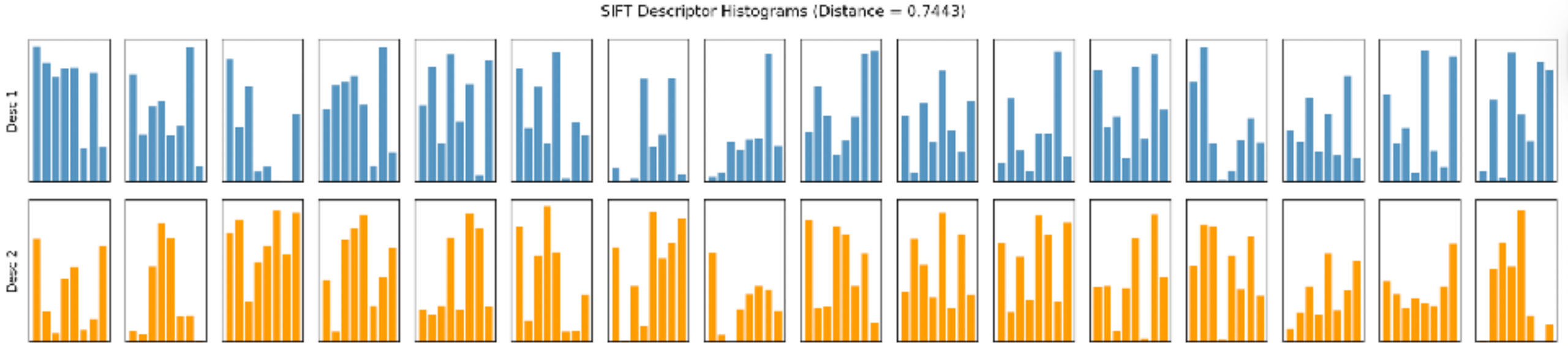


SIFT - Feature matching

SIFT - Matching

- SIFT descriptor 128 values
  - Usually these are bytes
- Computing the distance between features:

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \dots}$$



SIFT - Matching

- SIFT descriptor 128 values
  - Usually these are bytes
- Computing the distance between features:

$$d = \sqrt{\left(a_1 - b_1\right)^2 + \left(a_2 - b_2\right)^2 + \ldots}$$

  - 128 subtractions
  - 128 multiplications
  - 127 additions
  - 1 square root (optional)

SIFT - Matching

- SIFT descriptor 128 values
  - Usually these are bytes
- Computing the distance between features:

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \dots}$$

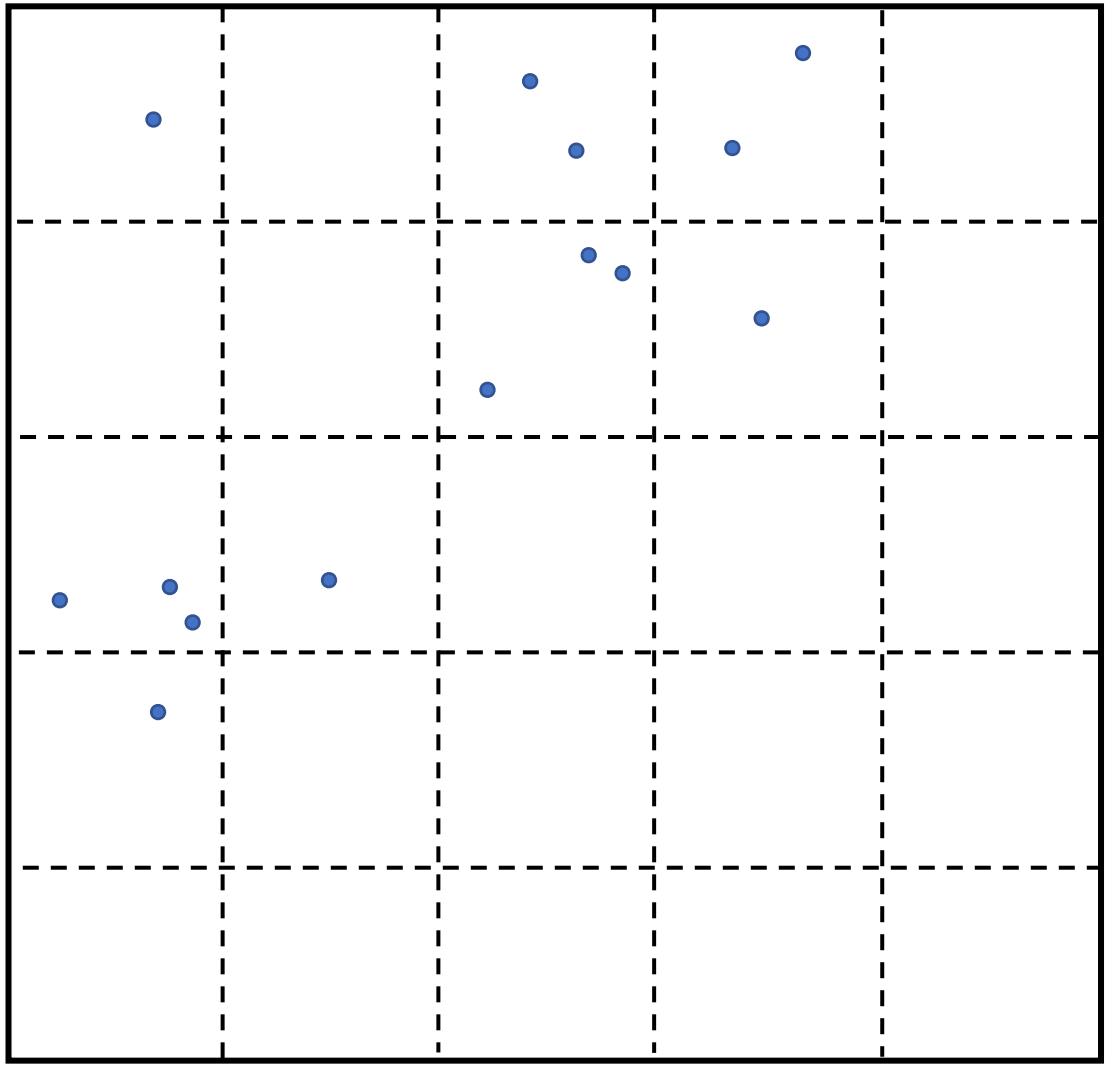
  - 128 subtractions
  - 128 multiplications
  - 127 additions
  - 1 square root (optional)

- Suppose we have 10,000 features in each image
  - Brute force method – find distance for each pair
  - Match is smallest distance
  - Matching one feature: $3{,}840{,}000$ operations
  - Matching all features: $38{,}400{,}000{,}000$ ops

SIFT - Approximate Nearest Neighbours

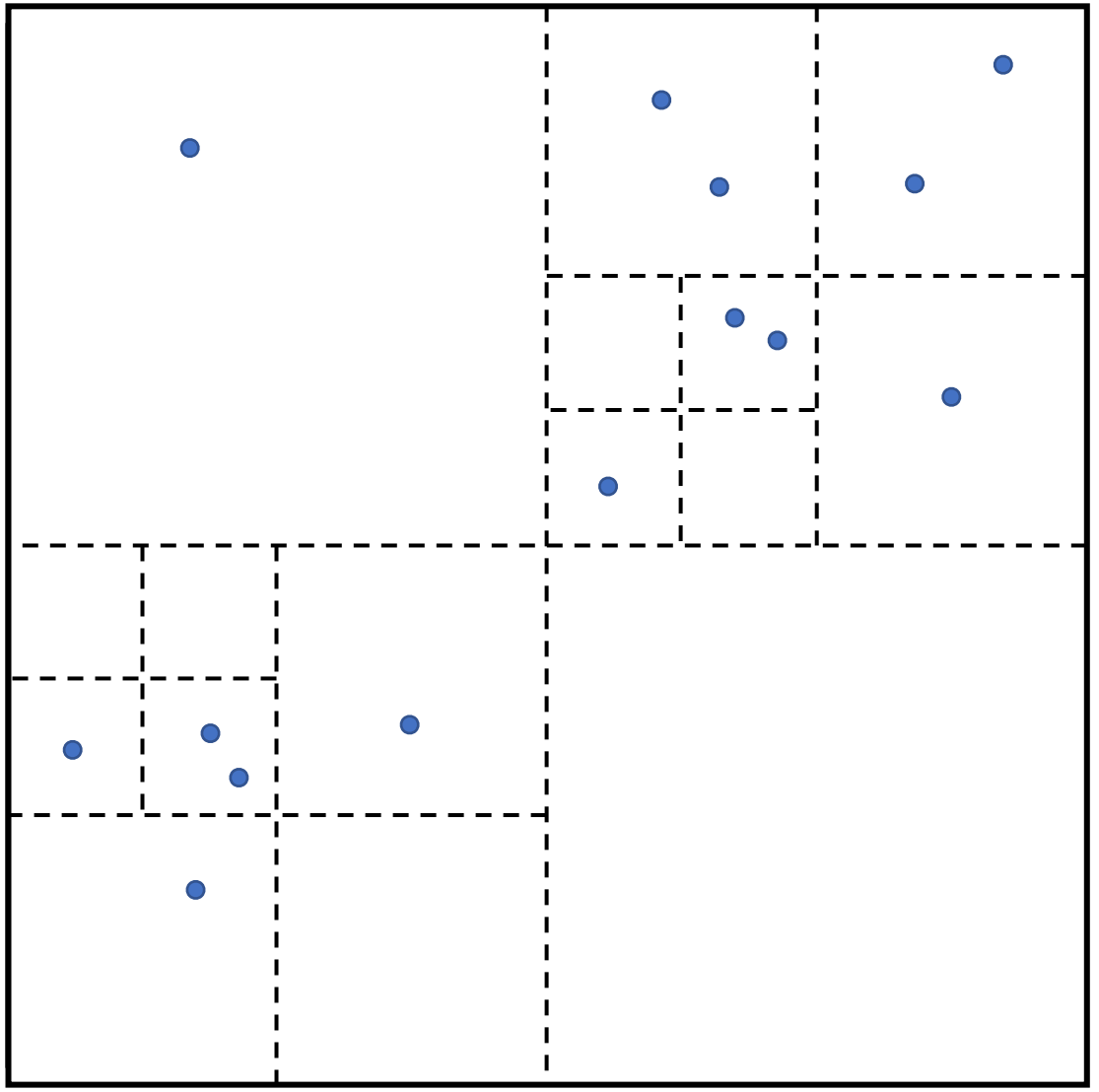
- Split space into smaller regions
  - Each feature lies in one region
  - Only look for matches in same region
  - Might not find nearest match (why?)
- Uniform subdivision:
  - Divide space into a regular grid
  - 10 divisions on 2-D grid – 100 regions
  - 10 divisions on 128-D grid – 10^{128}

Example for 2D Feature Vector



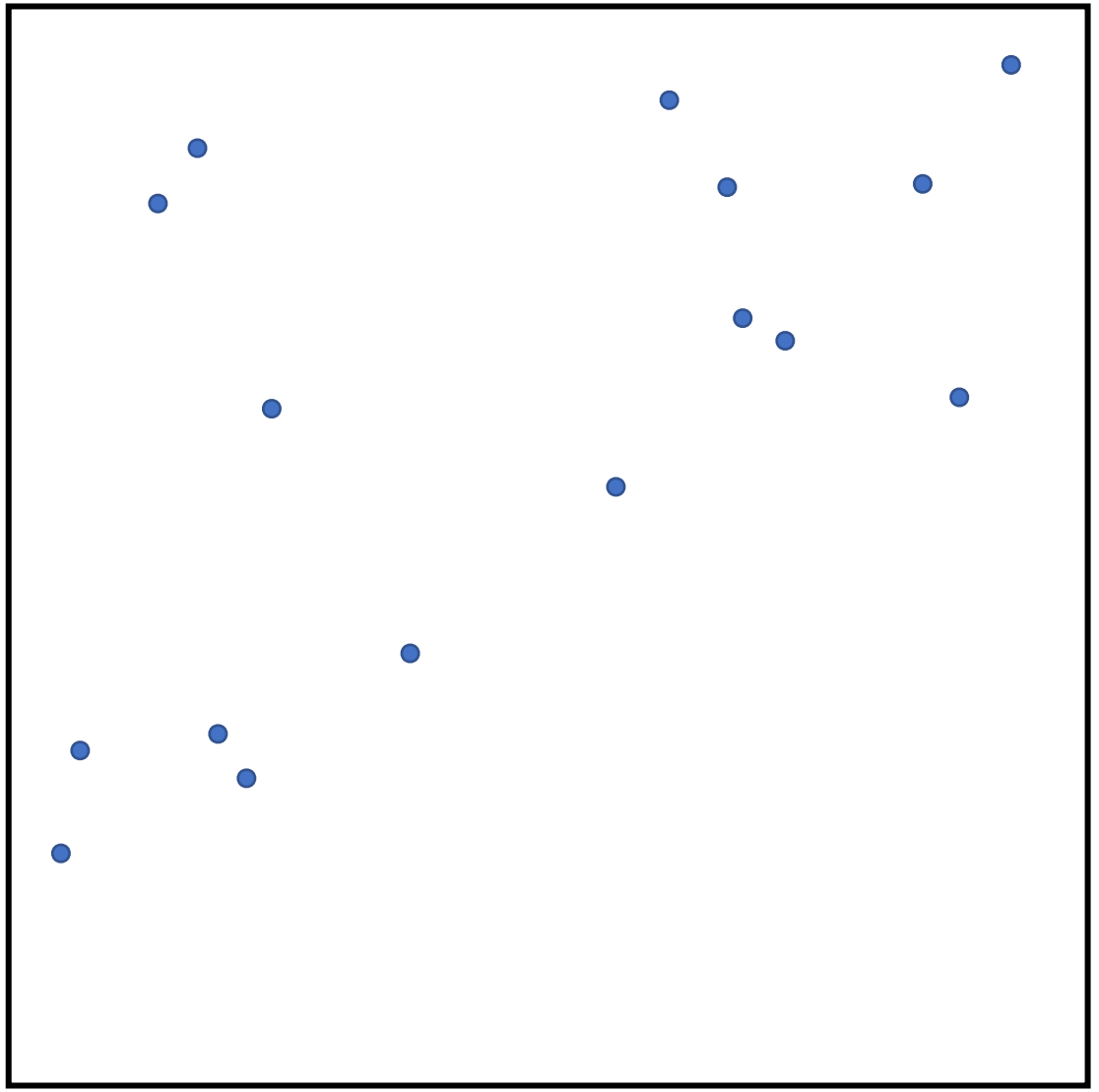
SIFT - Quadtrees, Octrees, etc.

- Recursively split space in half along each dimension
- Stop splitting when:
  - Some max depth is reached
  - Only a few items in the cell
- 2-D gives a quadtree
- 3-D gives an octree
- 128-D: : still too big -> $2^{128} \sim 10^{38}$

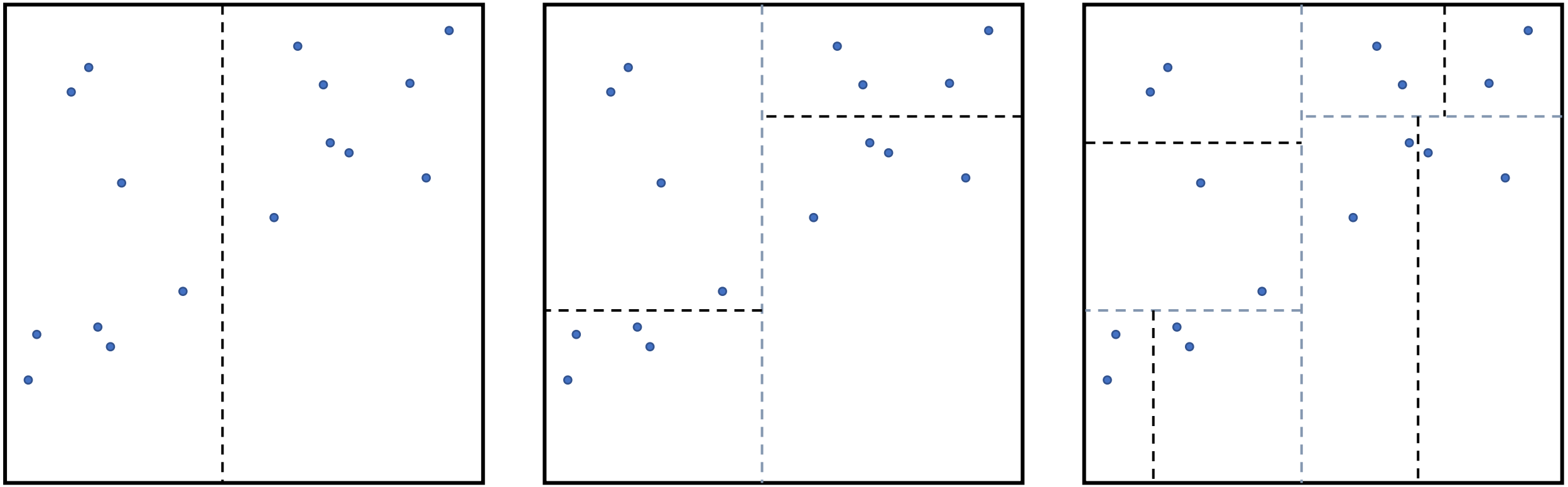


SIFT - K-D Trees

- k-dimensional tree
- Choose an axis and split along it
- Then take each half and repeat
- Stop when:
  - Only a few items in the region
  - Reached some maximum depth
- How to choose splits?
  - Axis – next, random, greatest spread?
  - Splitting point – middle or even split?



SIFT - K-D Trees



SIFT - K-D Trees and Feature Matching

- Put the feature descriptors from one image in a k-d tree
  - This has some overhead, but worth it if many features
- Given a feature from the other image:
  - Find what cell in the k-d tree contains that feature's descriptor
  - Compute the distance to all features in that cell
  - The nearest one is (probably) the best match
- For a tree with n layers and F features, this requires:
  - n comparisons to find the appropriate cell
  - $\frac{F}{2^n}$ descriptor-distance computations to find the best match

SIFT - Matching SIFT Features

- Even with brute-force, most SIFT matches are wrong
  - Lack of texture, repeating features, etc. -> ambiguous matches
  - And SIFT is the best we have
- With k-d trees this gets worse, but not much
- Solution – find best two matches
  - Ambiguous matches will have two similar distances
  - Keep only matches with a big difference between the two
  - Find two nearest matches $d\left(f, m_1\right) < \alpha d(f, m_2)$,
    - $d(f, m_i)$ is distance to ith match, $\alpha < 1$ is a parameter, e.g.: 0.8
- Makes things better, but still some wrong matches

SIFT - Why 0.8?

Distinctive Image Features from Scale-Invariant Keypoints

DAVID G. LOWE
Computer Science Department, University of British Columbia, Vancouver, B.C., Canada

Received January 30, 2003; Revised January 7, 2004; Accepted January 22, 2004

Abstract. This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images. This paper also describes an approach to using these features for object recognition. The recognition proceeds by matching individual features to a database of features from known objects using a fast nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and finally performing verification through least-squares solution for consistent pose parameters. This approach to recognition can robustly identify objects among clutter and occlusion while achieving near real-time performance.

Keywords: invariant features, object recognition, scale invariance, image matching

1. Introduction

Image matching is a fundamental aspect of many problems in computer vision, including object or scene recognition, solving for 3D structure from multiple images, stereo correspondence, and motion tracking. This paper describes image features that have many properties that make them suitable for matching differing images of an object or scene. The features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. Large numbers of features can be extracted from typical images with efficient algorithms. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition.

The cost of extracting these features is minimized by taking a cascade filtering approach, in which the more expensive operations are applied only at locations that pass an initial test. Following are the major stages of computation used to generate the set of image features:

1. *Scale-space extrema detection:* The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.
2. *Keypoint localization:* At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.
3. *Orientation assignment:* One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

"For our object recognition implementation, we reject all matches in which the distance ratio is greater than 0.8, which eliminates 90% of the false matches while discarding less than 5% of the correct matches."

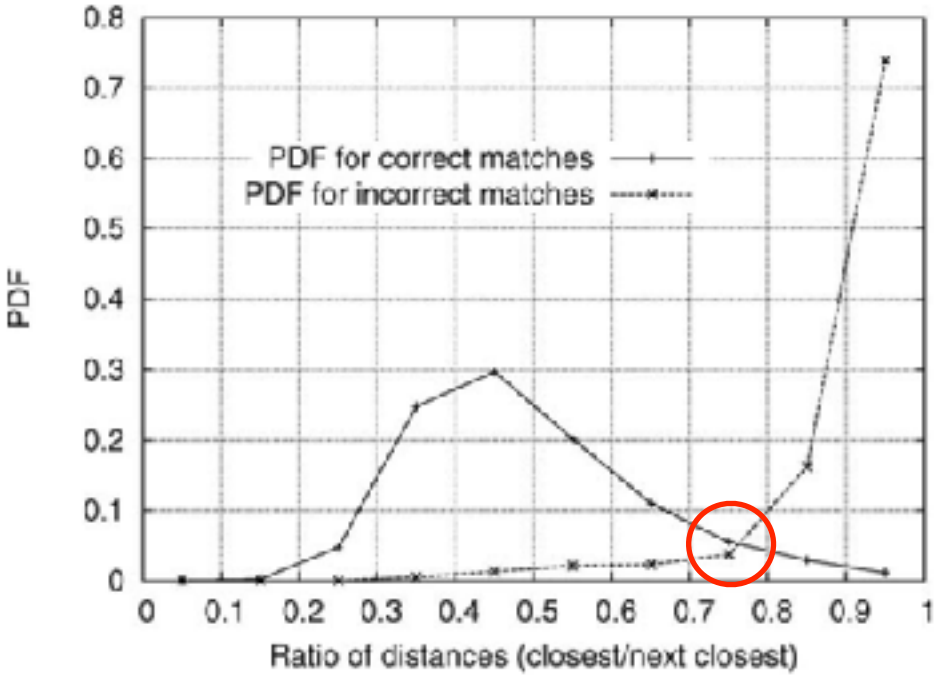
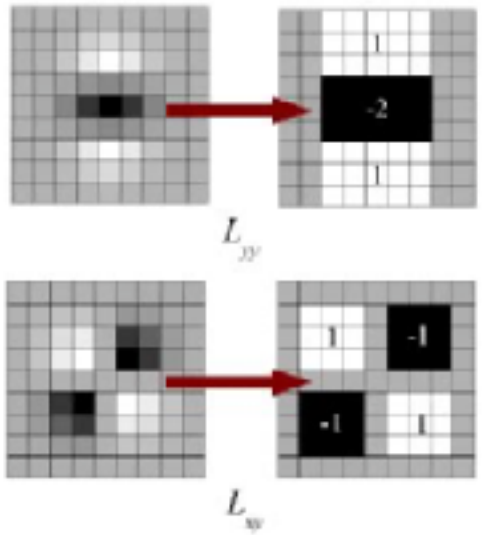


Figure 11. The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

Side Track: Other feature detectors

SURF: Speeded-Up Robust Features

- Type: Detector + descriptor (patented)
- Idea:
  - Uses Hessian matrix approximation with integral images -> fast scale-space detection
    - Hessian matrix: second derivatives of the image intensity
    - How quickly the slope changes
    - Big values: Strong Blobs
    - Box filters with integral images as approximation
  - Descriptors = Haar wavelet responses (orientation-robust)
- Advantages:
  - Faster than SIFT
  - Robust to scale, rotation, illumination changes
- Limitations:
  - Slower than binary methods (e.g., ORB)



Bay, Tuytelaars, Van Gool, " Speeded Up Robust Features ", European Conference on Computer Vision (ECCV) 2006.

ORB Features (Oriented FAST and Rotated BRIEF)

- Fast alternative to SIFT/SURF
- Key steps:
  - FAST corner detector:
    - Analyses intensities along a ring of 16 pixels centred on the pixel of interest $\boldsymbol{p}$
    - $\boldsymbol{p}$ is a FAST corner if a set of N (often =12) contiguous pixels on the ring are all brighter than the pixel intensity or all darker (very fast, but sensitive to noise)
  - BRIEF: Binary Robust Independent Elementary Features descriptors (binary strings) -> made rotation-invariant

ORB: an efficient alternative to SIFT or SURF

Ethan Rublee Vincent Rabaud Kurt Konolige Gary Bradski
Willow Garage Menlo Park, California
{erublee}{vrabaud}{konolige}{bradski}@willowgarage.com

Abstract

Feature matching is at the base of many computer vision problems, such as object recognition or structure from motion. Current methods rely on costly descriptors for detection and matching. In this paper, we propose a very fast binary descriptor based on BRIEF, called ORB, which is rotation invariant and resistant to noise. We demonstrate through experiments how ORB is at two orders of magnitude faster than SIFT, while performing as well in many situations. The efficiency is tested on several real-world applications, including object detection and patch-tracking on a smart phone.

1. Introduction

The SIFT keypoint detector and descriptor [17], although over a decade old, have proven remarkably successful in a number of applications using visual features, including object recognition [13], image stitching [20], visual mapping [23], etc. However, it imposes a large computational burden, especially for real-time systems such as visual odometry, or for low-power device such as cellphones. This was lost to an inventive search for replacements with lower computation cost, arguably the best of these is SURF [3]. There has also been research aimed at speeding up the computation of SIFT, most notably with GPU shaders[26].

In this paper, we propose a computationally efficient replacement to SIFT that has similar matching performance, is less affected by image noise, and is capable of being used for real-time performance. Our main motivation is to enhance many common image-matching applications, e.g., to enable low-power devices without GPU acceleration to perform panorama stitching and patch tracking, and to reduce the time for feature based object detection on standard PCs. Our descriptor performs as well as SIFT on these tasks (and better than SURF), while being almost two orders of magnitude faster.

Our proposed feature builds on the well-known FAST keypoint detector [19] and the recently-developed BRIEF descriptor [4]; for this reason we call it ORB (Oriented

Figure 1. Typical matching result using 500 ORB max-worst image with viewpoint change. 3mm line: are valid matches; red circles indicate unmatched points.

FAST and Rotated BRIEF. Both these techniques are attractive because of their good performance and low cost. In this paper, we address several limitations or desirable techniques vis-a-vis SIFT, most notably the lack of rotational invariance in BRIEF. Our main contributions are:

- The addition of a fast and accurate orientation component to FAST.
- The efficient computation of oriented BRIEF features.
- Analysis of variance and correlation of oriented BRIEF features.
- A learning method for de-correlating BRIEF features under rotational invariance, leading to better performance in nearest-neighbor applications.

To validate ORB, we perform experiments that test the properties of ORB relative to SIFT and SURF, for both raw matching ability, and performance in image-matching applications. We also illustrate the efficiency of ORB by implementing a patch-tracking application on a smart phone. An additional benefit of ORB is that it is free from the licensing restrictions of SIFT and SURF.

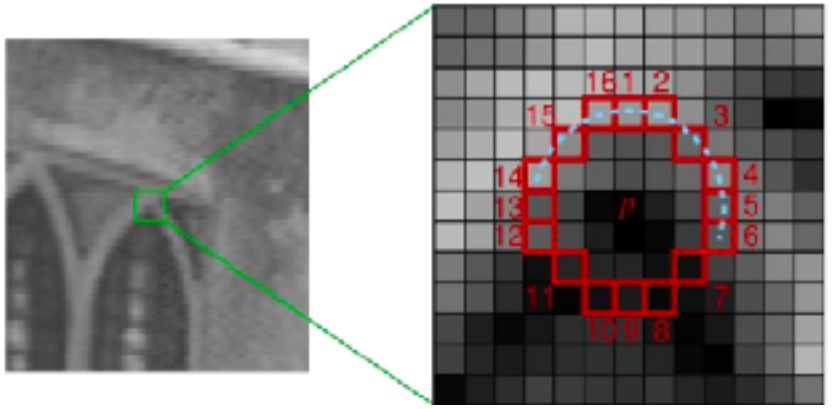
2. Related Work

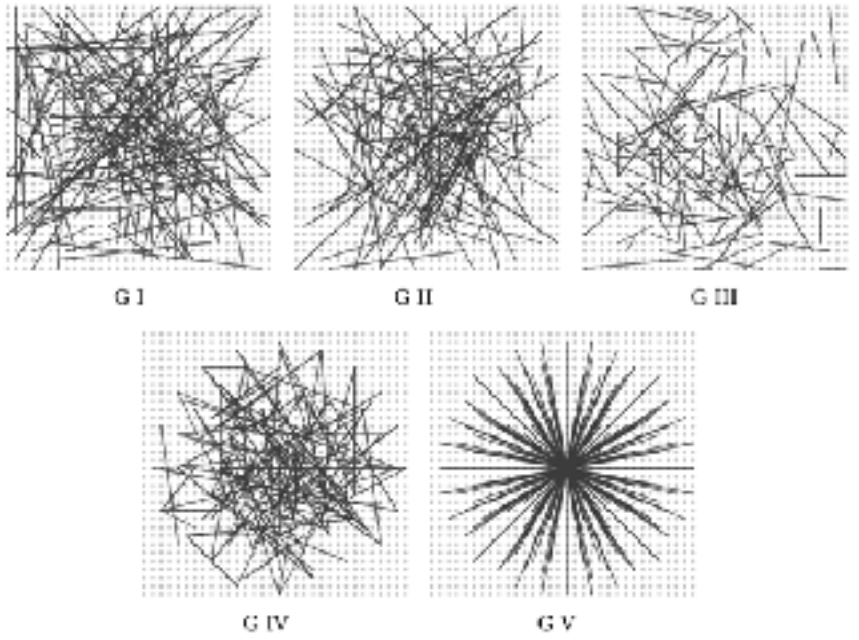
Keypoints SIFT and its variants [3, 24] are the method of choice for finding keypoints in real-time systems that match visual features, for example, Parallel Tracking and Mapping [13]. It is efficient and finds reasonably scarce keypoints, although it must be augmented with pyramid

E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *2011 International Conference on Computer Vision*, Barcelona, Spain, 2011, pp. 2564-2571, doi: 10.1109/ICCV.2011.6126544

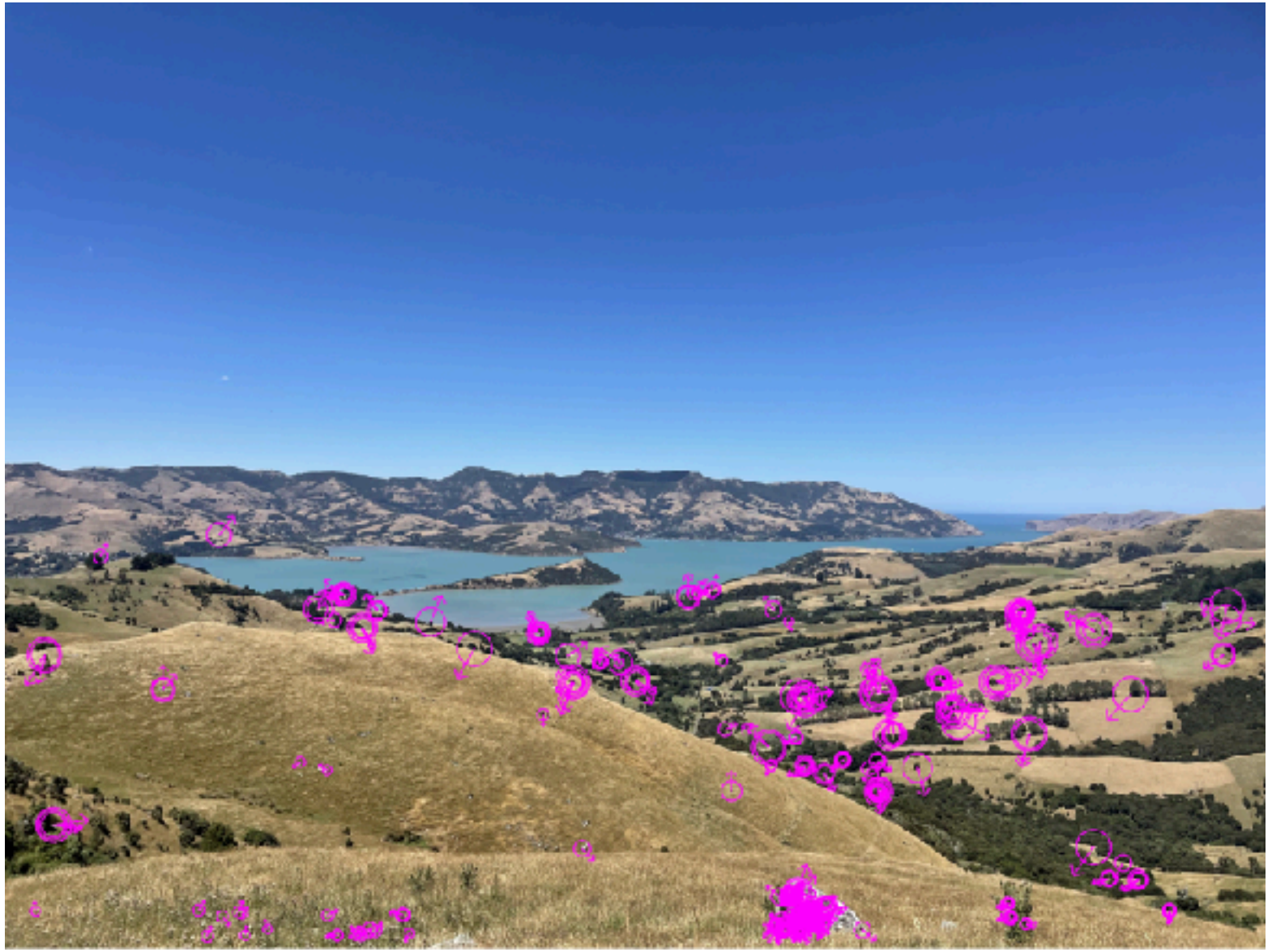
ORB Features (Oriented FAST and Rotated BRIEF)

- Fast alternative to SIFT/SURF
- Key steps:
  - FAST corner detector:
    - Analyses intensities along a ring of 16 pixels centred on the pixel of interest $\boldsymbol{p}$
    - $\boldsymbol{p}$ is a FAST corner if a set of N (often =12) contiguous pixels on the ring are all brighter than the pixel intensity or all darker (very fast, but sensitive to noise)
  - BRIEF: Binary Robust Independent Elementary Features descriptors (binary strings) -> made rotation-invariant



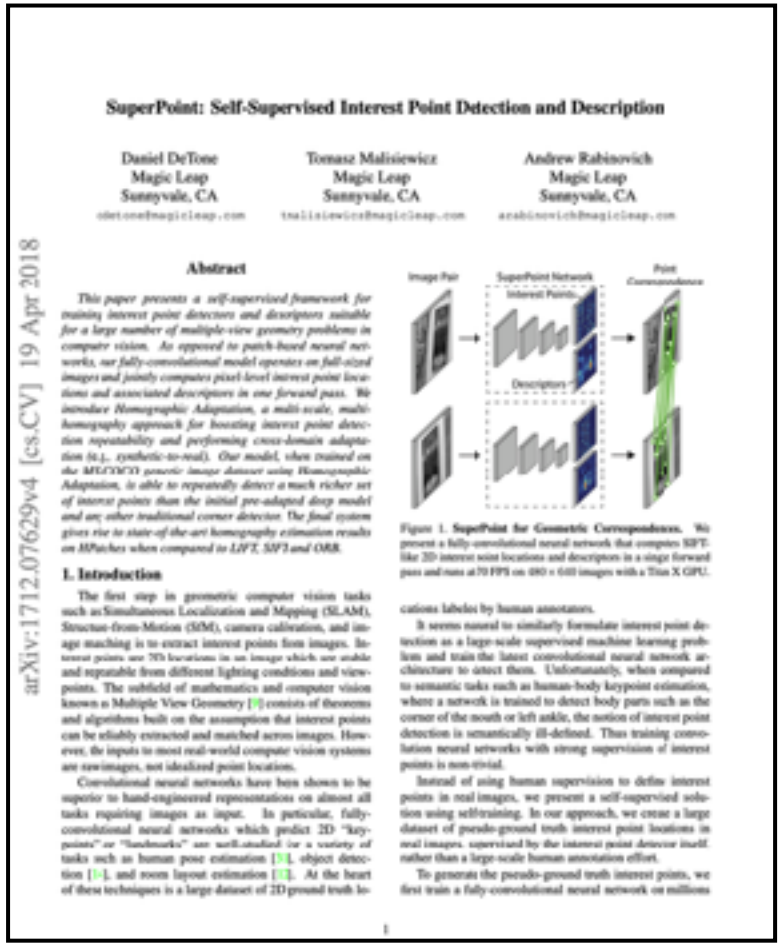


ORB Features (Oriented FAST and Rotated BRIEF)



SuperPoint

- Deep learning–based feature detector + descriptor
- Trains convolutional neural network (CNN) to directly find keypoints and compute descriptors
- Self-supervised training: synthetic “homographic adaptation” generates keypoints
- Network outputs:
    - Keypoint probability heatmap
    - Dense descriptor map
- Advantages:
    - Robust to illumination, viewpoint, texture-less regions
    - State-of-the-art for real-time vision (with GPU, 70 FPS on 480 × 640 images with a Titan X GPU)
- Limitations:
    - Requires GPU for training/inference.



SuperPoint: Self-Supervised Interest Point Detection and Description

Daniel DeTone, Magic Leap, Sunnyvale, CA
Tomasz Malisiewicz, Magic Leap, Sunnyvale, CA
Andrew Rabinovich, Magic Leap, Sunnyvale, CA

Abstract

This paper presents a self-supervised framework for training interest point detectors and descriptors suitable for a large number of multiple-view geometry problems in computer vision. As opposed to patch-based neural networks, our fully-convolutional model operates on full-sized images and jointly computes pixel-level interest point locations and associated descriptors in one forward pass. We introduce Homographic Adaptation, a multi-scale, multi-homography approach for boosting interest point detection repeatability and performing cross-domain adaptation (e.g., synthetic-to-real). Our model, when trained on the MS-COCO generic image dataset using Homographic Adaptation, is able to repeatedly detect a much richer set of interest points than the initial pre-adapted deep model and any other traditional corner detector. The final system gives rise to state-of-the-art homography estimation results on HPatches when compared to LIFT, SIFT and ORB.

1. Introduction

The first step in geometric computer vision tasks such as Simultaneous Localization and Mapping (SLAM), Structure-from-Motion (SfM), camera calibration, and image matching is to extract interest points from images. Interest points are 2D locations in an image which are stable and repeatable from different lighting conditions and viewpoints. The subfield of mathematics and computer vision known as Multiple View Geometry [9] consists of theorems and algorithms built on the assumption that interest points can be reliably extracted and matched across images. However, the inputs to most real-world computer vision systems are raw images, not idealized point locations.

Convolutional neural networks have been shown to be superior to hand-engineered representations on almost all tasks requiring images as input. In particular, fully-convolutional neural networks which predict 2D "keypoints" or "landmarks" are well-studied for a variety of tasks such as human pose estimation [31], object detection [31], and room layout estimation [31]. At the heart of these techniques is a large dataset of 2D ground truth locations labeled by human annotators.

It seems natural to similarly formulate interest point detection as a large-scale supervised machine learning problem and train the latest convolutional neural network architecture to detect them. Unfortunately, when compared to semantic tasks such as human-body keypoint estimation, where a network is trained to detect body parts such as the corner of the mouth or left ankle, the notion of interest point detection is semantically ill-defined. Thus training convolution neural networks with strong supervision of interest points is non-trivial.

Instead of using human supervision to define interest points in real images, we present a self-supervised solution using self-training. In our approach, we create a large dataset of pseudo-ground truth interest point locations in real images, supervised by the interest point detector itself, rather than a large-scale human annotation effort.

To generate the pseudo-ground truth interest points, we first train a fully-convolutional neural network on millions

Figure 1. SuperPoint for Geometric Correspondences. We present a fully-convolutional neural network that computes SIFT-like 2D interest point locations and descriptors in a single forward pass and runs at 70 FPS on 480 × 640 images with a Titan X GPU.

Detone, Malisiewicz, Rabinovich. SuperPoint: Self-Supervised Interest Point Detection and Description. CVPRW 2018

Image stitching

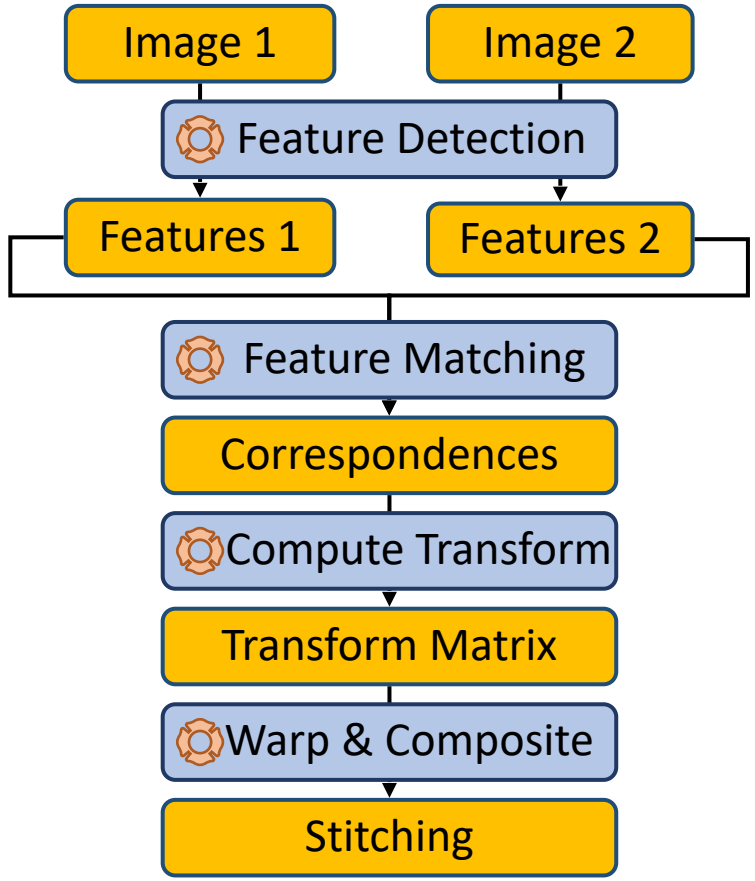
How to Create Panoramic Images?

Goal

Goal

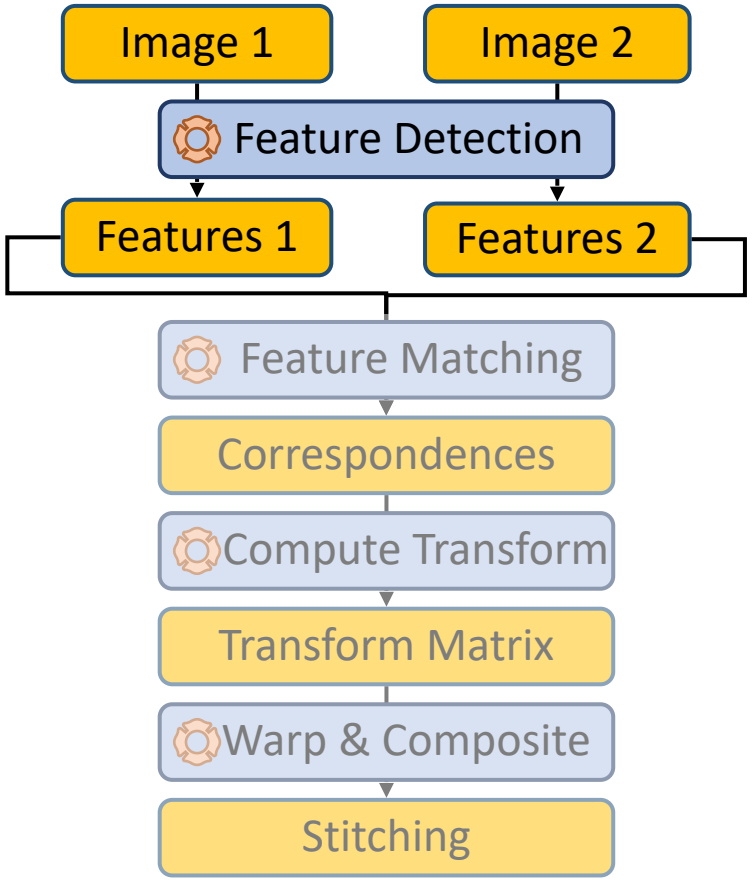
Image Stitching Process

- Four main stages:
  1. Detect features in each image
  2. Match features between images
  3. Estimate a transform
  4. Warp one image to the other
- Stitching only works if:
  - The scene is (almost) planar, or
  - The camera only (mostly) rotates



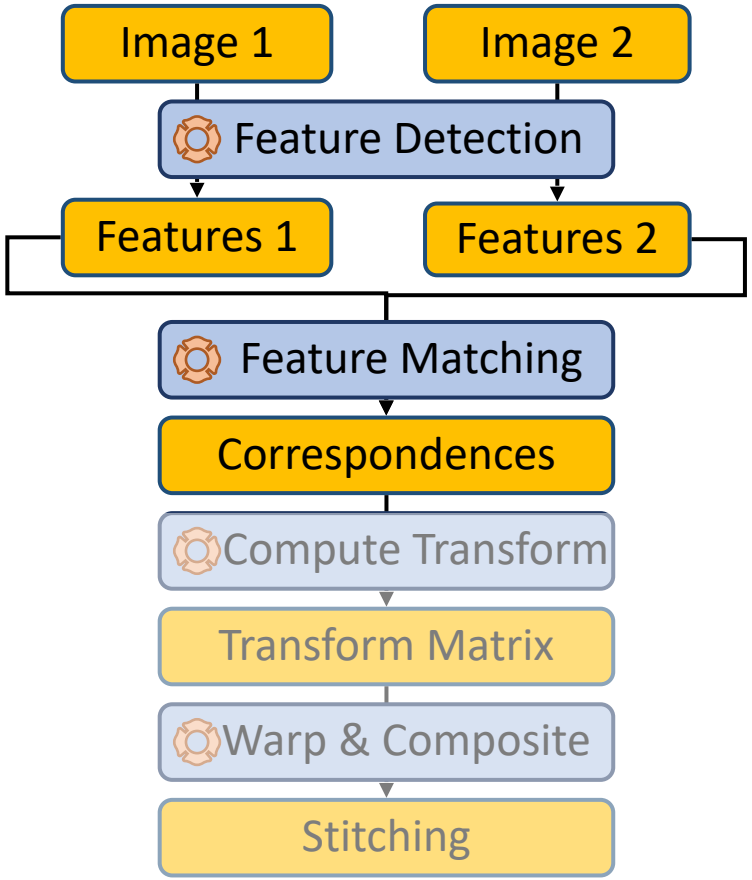
Step 1: Detect Features

- Feature points or key points
  - Can be reliably detected in different images of the same scene
  - Have precise location information
- Feature descriptors
  - Used to tell features apart
  - Ideally robust to changes in lighting, viewpoint, contrast, etc.



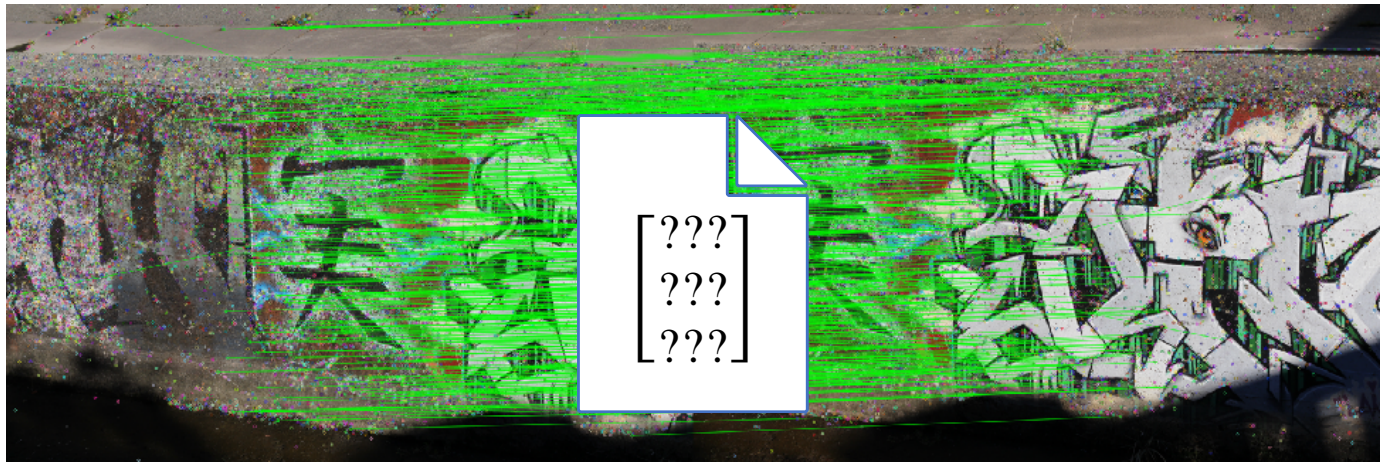
Step 2: Match Features

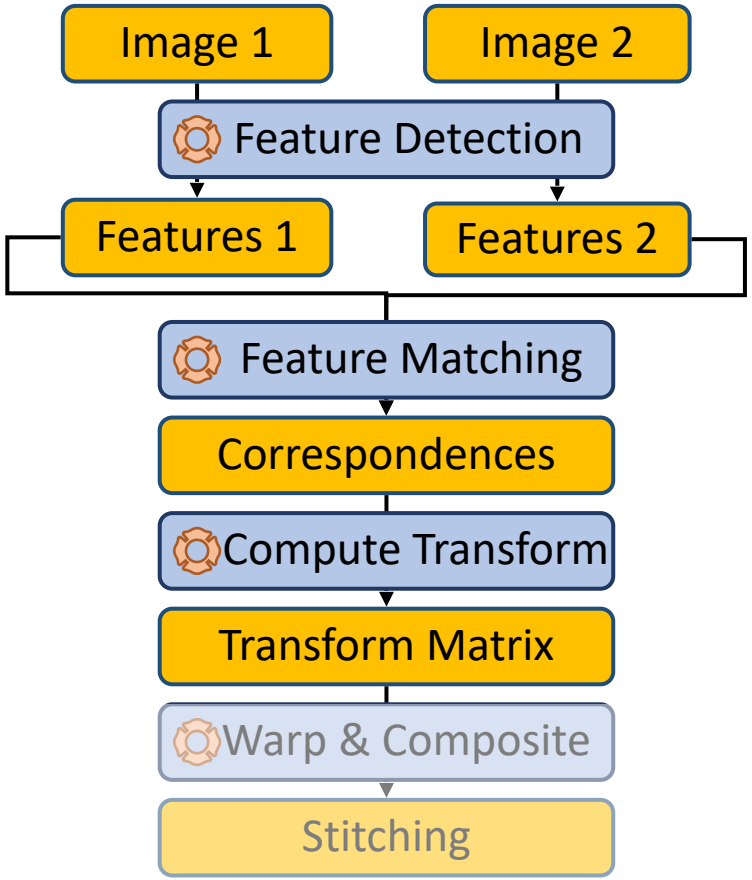
- Same scene point in both images
  - Features with similar descriptors
  - Efficient matching needs care
- Matches or correspondences
  - Capture the transform from one image to the next
  - Challenging part
  - Depends on good descriptors



Step 3: Compute Transform

- Find a general linear transform in 2D: Homography
  - Maps points between the images
  - Preserves straight lines
- Estimation needs care
  - Measurement errors
  - Uncertainty vs mistakes
  - Minimising a sensible error term





Step 4: Warping

- Warp based on the homography
  - Pick one image as the base
  - Map the other image to that coordinate frame
- Seamlines between images
  - Correct brightness, contrast, etc.
  - Blend between the images
  - Find seamlines that are hard to see

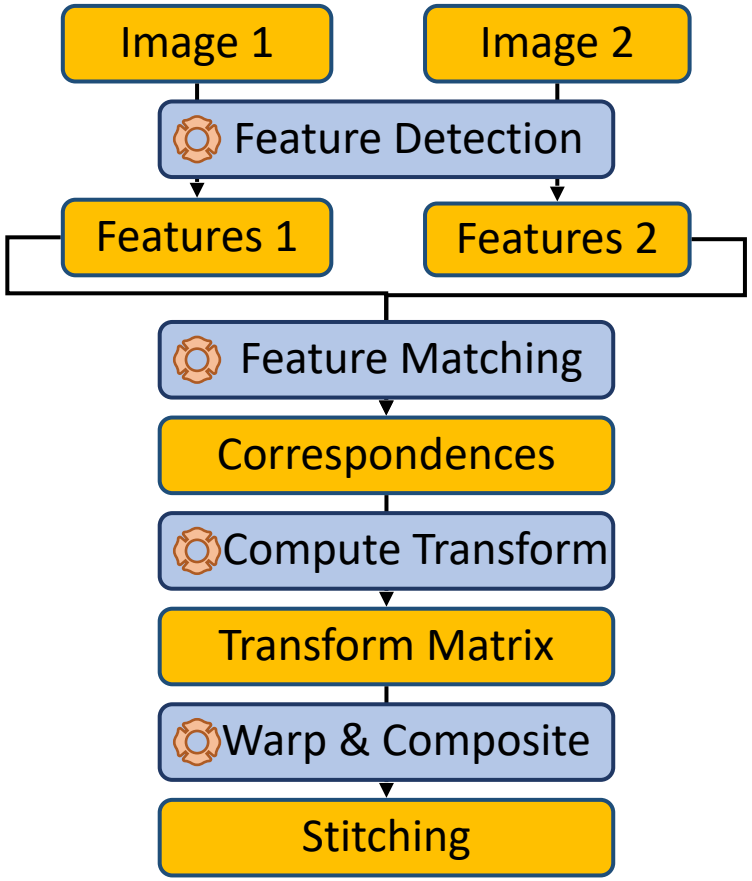
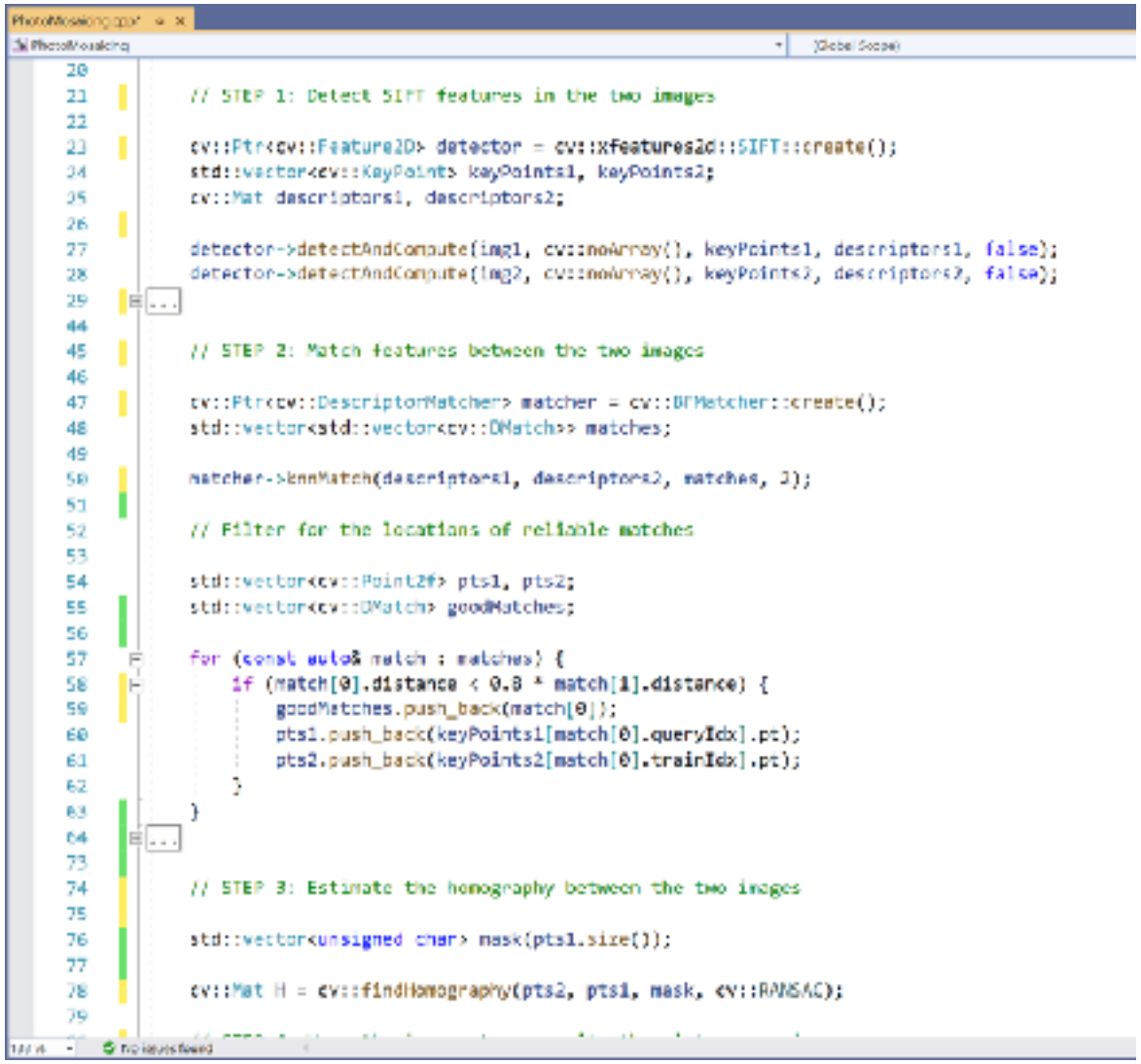


Image Stitching in Practice

- OpenCV makes this easy
  - Reading, writing, displaying images
  - Several different feature detectors
  - Different ways to match features
  - Homography estimation
  - Image warping and drawing

https://opencv.org/



```
// STEP 1: Detect SIFT features in the two images

cv::Ptr<cv::Feature2D> detector = cv::xfeatures2d::SIFT::create();
std::vector<cv::KeyPoint> keyPoints1, keyPoints2;
cv::Mat descriptors1, descriptors2;

detector->detectAndCompute(img1, cv::noArray(), keyPoints1, descriptors1, false);
detector->detectAndCompute(img2, cv::noArray(), keyPoints2, descriptors2, false);
...

// STEP 2: Match features between the two images

cv::Ptr<cv::DescriptorMatcher> matcher = cv::BFMatcher::create();
std::vector<std::vector<cv::DMatch>> matches;

matcher->knnMatch(descriptors1, descriptors2, matches, 2);

// Filter for the locations of reliable matches

std::vector<cv::Point2f> pts1, pts2;
std::vector<cv::DMatch> goodMatches;

for (const auto& match : matches) {
    if (match[0].distance < 0.8 * match[1].distance) {
        goodMatches.push_back(match[0]);
        pts1.push_back(keyPoints1[match[0].queryIdx].pt);
        pts2.push_back(keyPoints2[match[0].trainIdx].pt);
    }
}
...

// STEP 3: Estimate the homography between the two images

std::vector<unsigned char> mask(pts1.size());

cv::Mat H = cv::findHomography(pts2, pts1, mask, cv::RANSAC);
```

Homographies

Homography

$$\begin{bmatrix} u_i' \\ v_i' \\ 1 \end{bmatrix} \equiv \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

The feature location in the second image

Is equivalent to (equal up to a scale)

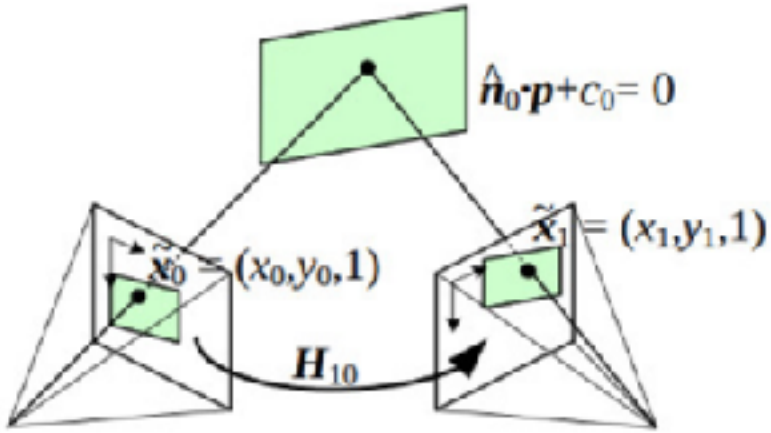
Some homography (a 3 X 3matrix)

Applied to…

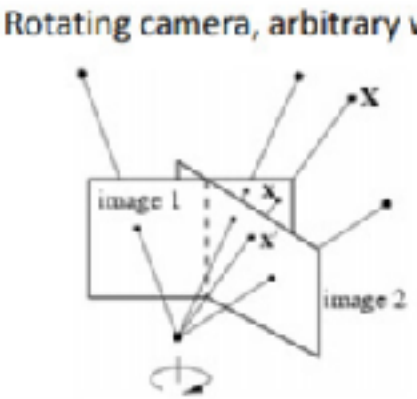
The feature location in the first image

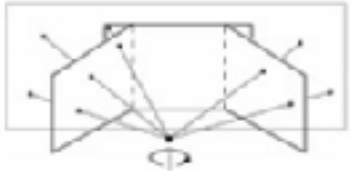
Homographies in the Stitching Process

- A homography* is:
  - A linear map between two planes (or views of a plane)
  - A 3 x 3 matrix, up to a scale
- Two images of a scene are related by a homography
  - If the scene is planar, or
  - If the camera only rotates



2D projective transformations (homographies), Christiano Gava, Gabriele Bleser





* For our purposes – this is one case of a more general concept

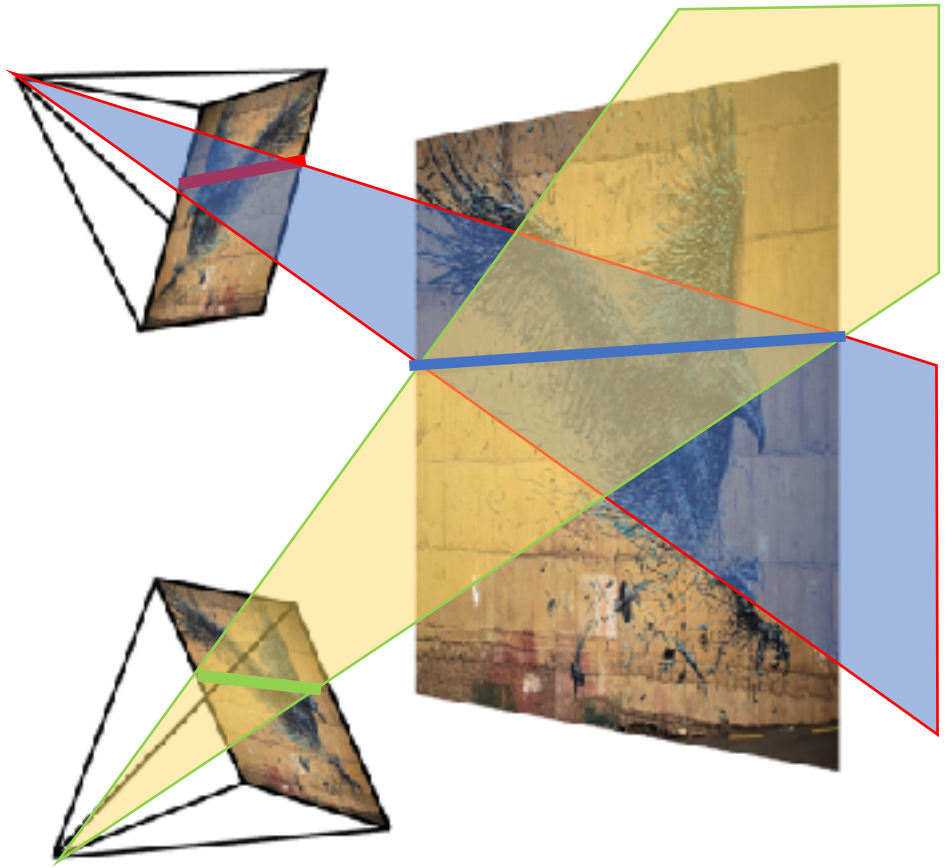
Side Track: Homographies in General

- In mathematics (projective geometry):
  - A homography (also called a projective transformation or collineation) is the most general transformation of projective space that maps lines to lines
  - It's not limited to 2D images—it applies in $\mathbb{P}^n$
  - Example: In 3D projective space, a homography is represented by a full 4×4 (up to scale).
- In computer vision practice:
  - We mostly use the 2D planar case: a 3×3

    3×3 matrix relating two images of a plane (or images from a rotating camera).

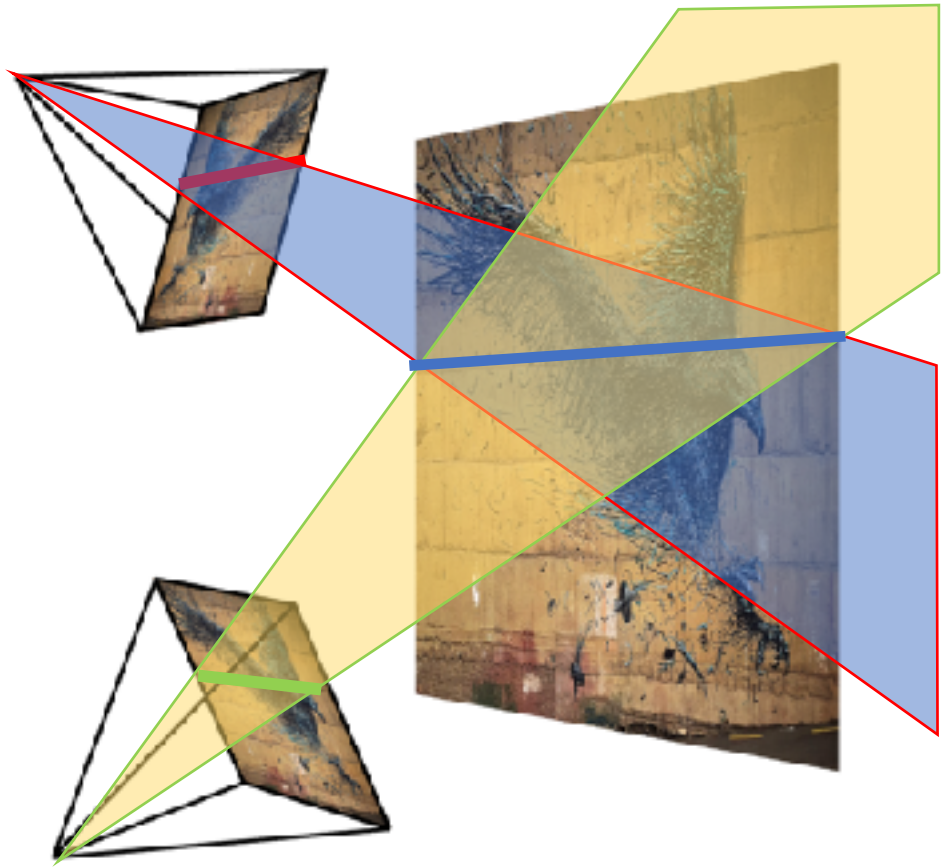
Homographies for Planar Scenes

- Take a line in one view
- This projects to a plane
- That intersects the scene (another plane) at a line
- That projects to a line in the other image
- So lines are preserved



Homographies for Planar Scenes

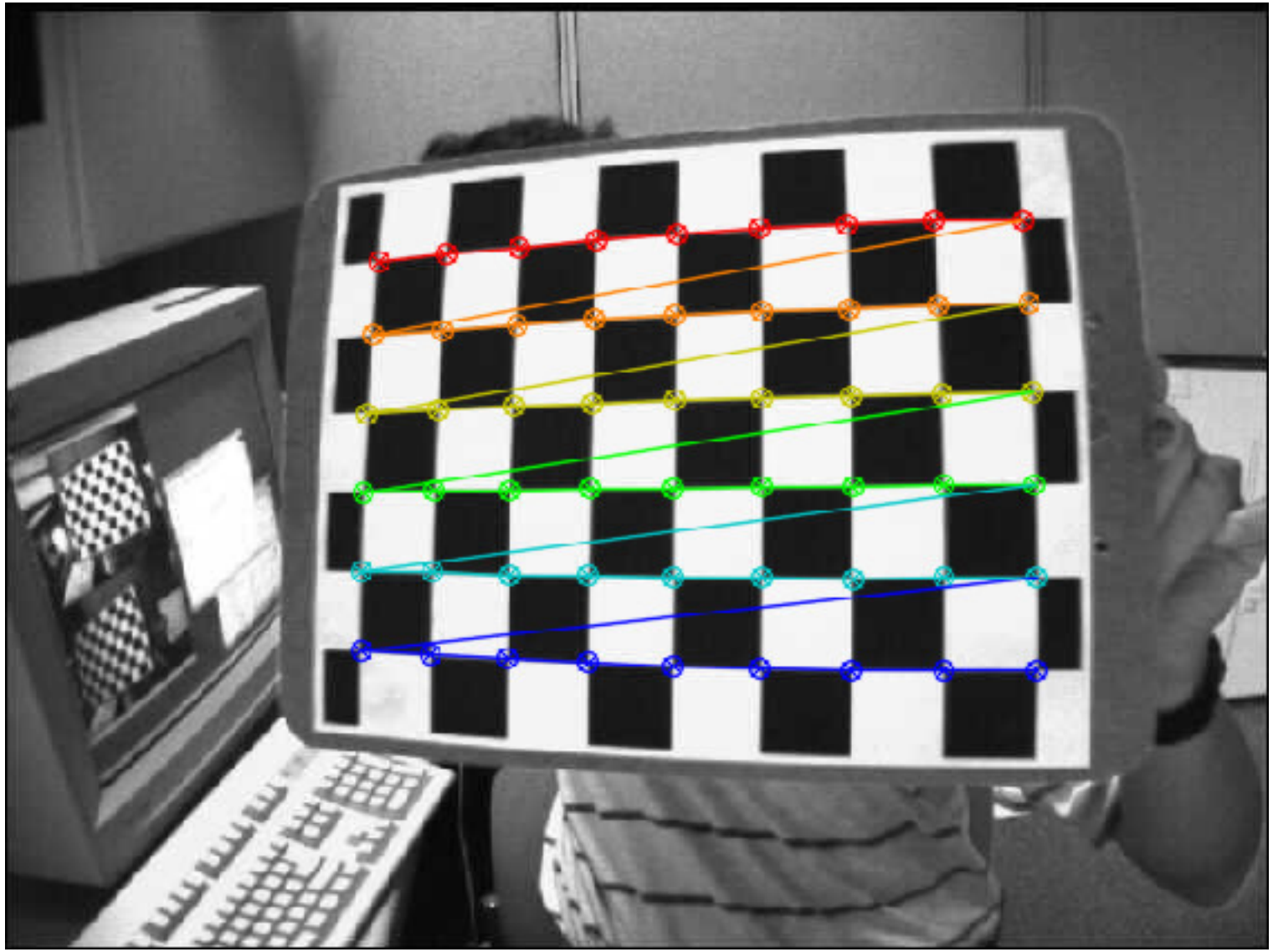
- Homographies for Planar Scenes
- Setup: The scene itself is flat (all observed points lie on a single 3D plane).
- Why a homography works:
  - The projection of a plane in 3D into two different camera views is always related by a 3×3 homography
  - Even if the camera undergoes general motion (translation + rotation), as long as all points are on the same plane, the mapping between the two image views is exactly a homography
- Use case: Texture mapping, AR marker tracking (e.g., fiducials)



Applications: AR Markers

Applications: Image Registration

Applications: Camera Calibration



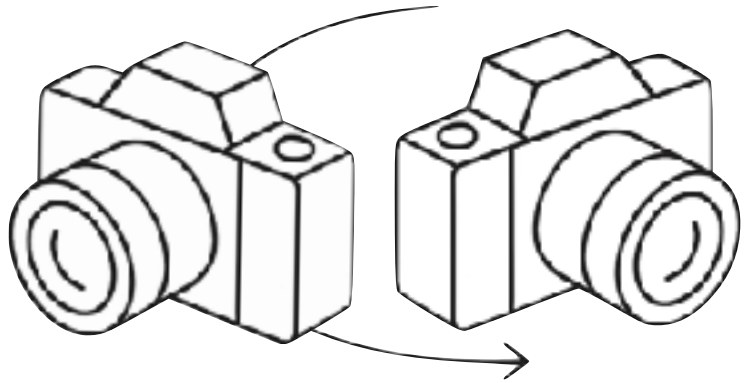
Applications: ARRephotography

Hasselman et al. ARephotography: Revisiting Historical Photographs using Augmented Reality, CHI Late-Breaking Work 2023

Homographies for Rotating Camera

- Assume cameras at origin

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \mathrm{K}\begin{bmatrix} \mathrm{R} \mid \mathbf{0} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathrm{KR} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



- If we have two cameras

$$\mathbf{u}_1 \equiv \mathrm{K}_1\mathrm{R}_1\mathbf{x}$$
$$\mathbf{u}_2 \equiv \mathrm{K}_2\mathrm{R}_2\mathbf{x}$$

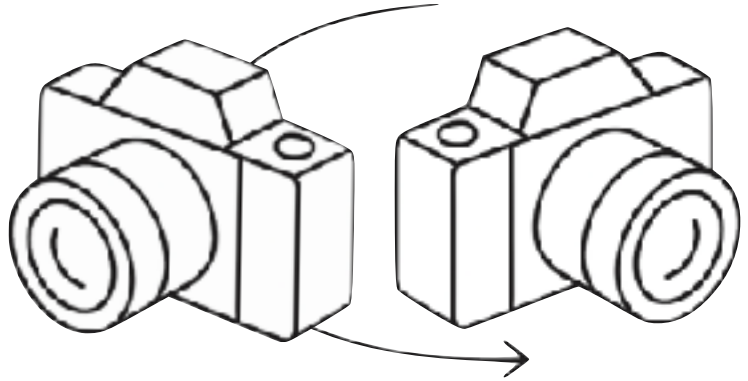
- Rearranging to get $\mathbf{x}$:

$$\mathrm{R}_1^{-1}\mathrm{K}_1^{-1}\mathbf{u}_1 \equiv \mathbf{x} \equiv \mathrm{R}_2^{-1}\mathrm{K}_2^{-1}\mathbf{u}_2$$
$$\mathbf{u}_1 \equiv \mathrm{K}_1\mathrm{R}_1\mathrm{R}_2^{-1}\mathrm{K}_2^{-1}\mathbf{u}_2$$
$$\mathbf{u}_1 = \mathrm{H}\mathbf{u}_2$$

- So the two images are related by a homography

Homographies for Rotating Camera

- Setup: The camera rotates around its optical center, but does not translate. The scene can be fully 3D (not planar)
- Why a homography works:
  - With pure rotation, the rays leaving the camera pivot but still all emanate from the same point
  - This means the transformation between the two images is equivalent to a change of coordinates on the viewing sphere
  - In homogeneous image coordinates, this is expressed as a homography (dependent only on the rotation and calibration)
- Use case: Panorama stitching, where the camera is rotated on a tripod without moving position



Not Homographies

- If we have both:
  - Non-planar scene, and
  - Translating camera
- There is no homography between the images
  - Lines may not be preserved
  - Cannot make a panorama
  - But we can do stereo (later)

Not Homographies

- If we have both:
  - Non-planar scene, and
  - Translating camera
- There is no homography between the images
  - Lines may not be preserved
  - Cannot make a panorama
  - But we can do stereo (later)

Homographies and Image Stitching

- To stitch an image:
  - Need to warp images to align
  - This warping is a homography

- How to find the homography?
  - If a feature at (u,v) in one image matches to (u',v') in the other, then

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \equiv \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Easy Option: OpenCV

```
cv::Mat H = cv::findHomography(pts1, pts2, cv::RANSAC, 3.0);
```

- pts1 – Points in first image (std::vector<cv::Point2f>)
- pts2 – Points in second image (std::vector<cv::Point2f>)
- cv::RANSAC - method to use
- 3.0 – RANSAC threshold

https://docs.opencv.org/4.4.0/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780

Solving from Point Correspondenes

- Suppose we have n matching points between two images
  $(u_1, v_1) \leftrightarrow (u_1', v_1'), (u_2, v_2) \leftrightarrow (u_2', v_2'), \ldots (u_n, v_n) \leftrightarrow (u_n', v_n')$
- Each pair gives us information about the homography, H
  $x_i' \equiv \mathrm{H} x_i$
- The equivalence makes things difficult:
  - Everything has an unknown scale factor associated with it
  - Can't solve things uniquely – if H is a solution, so is
    $k\mathrm{H}, \ \forall k \neq 0$

Solving from Point Correspondenes

- Remember: We want to use the relationship: $x' = \lambda Hx$ where
  - x,x′ are homogeneous 2D points (3×1),
  - H is the homography (3×3),
  - λ is an unknown scale factor
- The problem: λ is different for each correspondence, so it's not something we can just solve for globally
- The trick: Equation $x' = \lambda Hx$ means the two vectors x′ and Hx are parallel in 3D (homogeneous coordinate space) -> In other words, one is a scaled version of the other
  - This means their cross product is 0: $x' \times (Hx) = 0$
  - No λ -> no worry -> Each correspondence provides 2 independent linear equations in the entries of H
  - Need at least 4 correspondences to solve for H (8 dof)

The end!