CHAPTER 9

# Interest Points

One strategy for registering an image to another is to find *interest points* and register those. Interest points have the following important properties:

- It must be possible to find them reasonably reliably, even when image brightness changes.
- It must be possible to *localize* the point (ie tell where the point is) by looking at an image window around the point. For example, a corner can be localized; but a point along a straight edge can't, because sliding a window around the point along the edge leads to a new window that looks like the original.
- The location of the point must be *covariant* under at least some natural image transformations. This means that, if the image is transformed, the point will be found in an appropriate spot in the transformed image. Equivalently, the points "stick to" objects in the image – if the camera moves, the point stays on the object where it was, and so moves in the image. So if, for example, if $I_2$ is obtained by rotating $I_1$, then there should be an interest point at each location in $I_2$ obtained by rotating the position of an interest point in $I_1$.
- It must be possible to compute a description of the image in the neighborhood of the point, so the point can be matched. Ideally, corresponding points in different images will have similar descriptions, and different points will have different images. To compute this description, we need to be able to construct a neighborhood of the interest point that is covariant. So, for example, if the image is zoomed in, the neighborhood in the image gets bigger; and if it is zoomed out, the neighborhood gets smaller. Using a fixed size neighborhood when the image zooms won't work, because the neighborhood in the zoomed in image will contain patterns that aren't in the neighborhood in the zoomed out image.

These properties are summarized in Figure **??**. The direct constructions for interest points are worth reviewing, because they expose how these properties are achieved. Learned constructions are now competitive with direct constructions, and I describe one in section 33.2.

## 9.1 DIRECT INTEREST POINT DETECTORS

### 9.1.1 Finding Corners

Interest points are usually constructed at corners, because they can be localized and are quite easy to find with a straightforward detector. At a corner, we expect two important effects. First, there should be large gradients. Second, in a small neighborhood, the gradient orientation should swing sharply. We can identify corners by looking at variations in orientation within a window. In particular, the

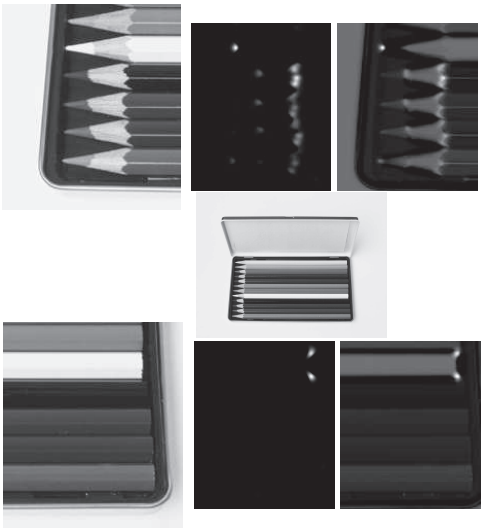

FIGURE 9.1: *The response of the Harris corner detector visualized for two detail regions of an image of a box of colored pencils (***center***). **Top left**, a detail from the pencil points; **top center**, the response of the Harris corner detector, where more positive values are lighter. The **top right** shows these overlaid on the original image. To overlay this map, we added the images, so that areas where the overlap is notably dark come from places where the Harris statistic is negative (which means that one eigenvalue of $\mathcal{H}$ is large, the other small). Note that the detector is affected by contrast, so that, for example, the point of the mid-gray pencil at the top of this figure generates a very strong corner response, but the points of the darker pencils do not, because they have little contrast with the tray. For the darker pencils, the strong, contrasty corners occur where the lead of the pencil meets the wood. The* **bottom** *sequence shows corners for a detail of pencil ends. Notice that responses are quite local, and there are a relatively small number of very strong corners.* Steve Gorton © Dorling Kindersley, used with permission.

matrix

$$\begin{aligned}\mathcal{H} &= \sum_{window} \left\{(\nabla I)(\nabla I)^T\right\} \\ &\approx \sum_{window} \left\{ \begin{array}{cc} (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial x} * *\mathcal{I}) & (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) \\ (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) & (\frac{\partial G_\sigma}{\partial y} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) \end{array} \right\}\end{aligned}$$

gives a good idea of the behavior of the orientation in a window. In a window of constant gray level, both eigenvalues of this matrix are small because all the terms are small. In an edge window, we expect to see one large eigenvalue associated with gradients at the edge and one small eigenvalue because few gradients run in other directions. But in a corner window, both eigenvalues should be large.

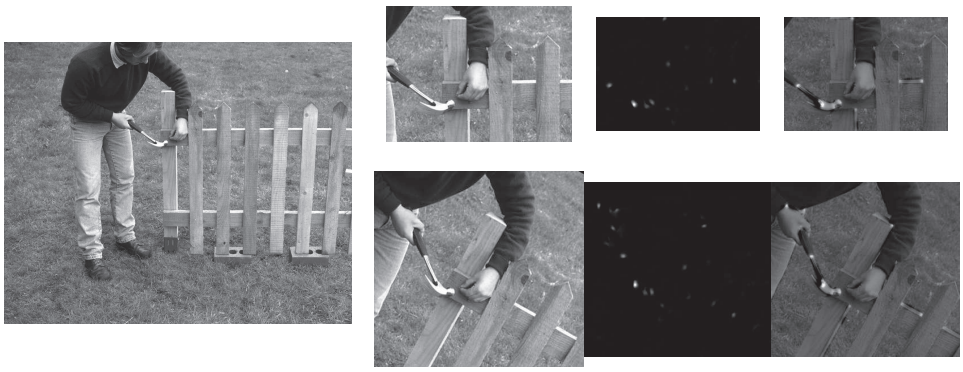

FIGURE 9.2: *The response of the Harris corner detector is unaffected by rotation and translation. The* **top row** *shows the response of the detector on a detail of the image on the* **far left**. *The* **bottom row** *shows the response of the detector on a corresponding detail from a rotated version of the image. For each row, we show the detail window (***left***); the response of the Harris corner detector, where more positive values are lighter (***center***); and the responses overlaid on the image (***right***). Notice that responses are quite local, and there are a relatively small number of very strong corners. To overlay this map, we added the images, so that areas where the overlap is notably dark come from places where the Harris statistic is negative (which means that one eigenvalue of $\mathcal{H}$ is large, the other small). The arm and hammer in the top row match those in the bottom row; notice how well the maps of Harris corner detector responses match, too.* © Dorling Kindersley, used with permission.

The *Harris corner detector* looks for local maxima of

$$\det(\mathcal{H}) - k(\frac{\text{trace}(\mathcal{H})}{2})^2$$

where $k$ is some constant [**?**]; we used 0.5 for Figure 9.1. These local maxima are then tested against a threshold. This tests whether the product of the eigenvalues (which is $\det(\mathcal{H})$) is larger than the square of the average (which is $(\text{trace}(\mathcal{H})/2)^2$). Large, locally maximal values of this test function imply the eigenvalues are both big, which is what we want. Figure 9.1 illustrates corners found with the Harris detector. This detector is unaffected by translation and rotation (Figure 9.2).

### 9.1.2 Building Neighborhoods

There are many ways of representing a neighborhood around an interesting corner. Methods vary depending on what might happen to the neighborhood. In what follows, we will assume that neighborhoods are only translated, rotated, and scaled (rather than, say, subjected to an affine or projective transformation), and so without loss of generality we can assume that the patches are circular. We must estimate the radius of this circle. There is technical machinery available for the neighborhoods that result from more complex transformations, but it is more intricate; see [].

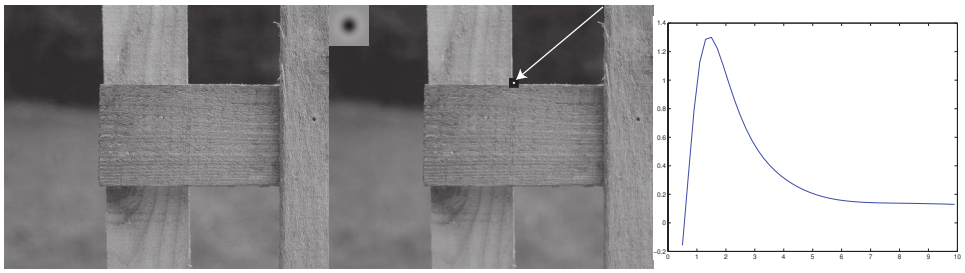

FIGURE 9.3: *The scale of a neighborhood around a corner can be estimated by finding a local extremum, in* scale *of the response at that point to a smoothed Laplacian of Gaussian kernel. On the* **left**, *a detail of a piece of fencing. In the center, a corner identified by an arrow (which points to the corner, given by a white spot surrounded by a black ring).* Overlaid *on this image is a Laplacian of Gaussian kernel, in the* **top right** *corner; dark values are negative, mid gray is zero, and light values are positive. Notice that, using the reasoning of Section* **??**, *this filter will give a strong positive response for a dark blob on a light background, and a strong negative response for a light blob on a dark background, so by searching for the strongest response at this point as a function of scale, we are looking for the size of the best-fitting blob. On the* **right**, *the response of a Laplacian of Gaussian* at the location of the corner, *as a function of the smoothing parameter (which is plotted in pixels). There is one extremal scale, at approximately 2 pixels. This means that there is one scale at which the image neighborhood looks most like a blob (some corners have more than one scale).* © Dorling Kindersley, used with permission.

To turn a corner into an image neighborhood, we must estimate the radius of the circular patch (equivalently, its scale). The radius estimate should get larger proportionally when the image gets bigger. For example, in a 2x scaled version of the original image, our method should double its estimate of the patch radius. This property helps choose a method. We could center a blob of fixed appearance (say, dark on a light background) on the corner, and then choose the scale to be the radius of the best fitting blob. An efficient way to do this is to use a Laplacian of Gaussian filter.

The *Laplacian* of a function in 2D is defined as

$$(\nabla^2 f)(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

It is natural to smooth the image before applying a Laplacian. Notice that the Laplacian is a linear operator (if you're not sure about this, you should check), meaning that we could represent taking the Laplacian as convolving the image with some kernel (which we write as $K_{\nabla^2}$). Because convolution is associative, we have that

$$(K_{\nabla^2} * *(G_\sigma * *I)) = (K_{\nabla^2} * *G_\sigma) * *I = (\nabla^2 G_\sigma) * *I.$$

The reason this is important is that, just as for first derivatives, smoothing an image and then applying the Laplacian is the same as convolving the image with

the Laplacian of the kernel used for smoothing. Figure 9.3 shows the resulting kernel for Gaussian smoothing; notice that this looks like a dark blob on a light background.

Imagine applying a smoothed Laplacian operator to the image at the center of the patch. Write $\mathcal{I}$ for the image, $\nabla^2_\sigma$ for the smoothed Laplacian operator with smoothing constant $\sigma$, $\uparrow_k \mathcal{I}$ for the the image with size scaled by $k$, $(x_c, y_c)$ for the coordinates of the patch center, and $(x_{kc}, y_{kc})$ for the coordinates of the patch center in the scaled image. Assume that upscaling is perfect, and there are no effects resulting from the image grid. This is fair because effects will be small for the scales of interest for us. Then, we have

$$(\nabla^2_{k\sigma} \uparrow_k \mathcal{I})(x_c, y_c) = (\nabla^2_\sigma \mathcal{I})(x_{kc}, y_{kc})$$

(this is most easily demonstrated by reasoning about the image as a continuous function, the operator as a convolution, and then using the change of variables formula for integrals). Now choose a radius $r$ for the circular patch centered at $(x_c, y_c)$, such that

$$r(x_c, y_c) = \underset{\sigma}{\text{argmax}} \ \nabla^2_\sigma \mathcal{I}(x_c, y_c)$$

(Figure 9.3). If the image is scaled by $k$, then this value of $r$ will be scaled by $k$ too, which is the property we wanted. This procedure looks for the scale of the best approximating blob. Notice that a Gaussian pyramid could be helpful here; we could apply the same smoothed Laplacian operator to different levels of a pyramid to get estimates of the scale.

As we have seen, orientation histograms are a natural representation of image patches. However, we cannot represent orientations in image coordinates (for example, using the angle to the horizontal image axis), because the patch we are matching to might have been rotated. We need a reference orientation so all angles can be measured with respect to that reference. A natural reference orientation is the most common orientation in the patch. We compute a histogram of the gradient orientations in this patch, and find the largest peak. This peak is the reference orientation for the patch. If there are two or more peaks of the same magnitude, we make multiple copies of the patch, one at each peak orientation.

### 9.1.3 Describing Neighborhoods with Orientations

We know the center, radius, and orientation of a set of an image patch, and must now represent it. Orientations should provide a good representation. They are unaffected by changes in image brightness, and different textures tend to have different orientation fields. The pattern of orientations in different parts of the patch is likely to be quite distinctive. Our representation should be robust to small errors in the center, radius, or orientation of the patch, because we are unlikely to estimate these exactly right.

We must build features that can make it obvious what orientations are present, and roughly where they are, but are robust to some rearrangement. One approach is to represent the neighborhood with a histogram of the elements that appear there. This will tell us what is present, but it confuses too many patterns with one another. For example, all neighborhoods with vertical stripes will get mixed

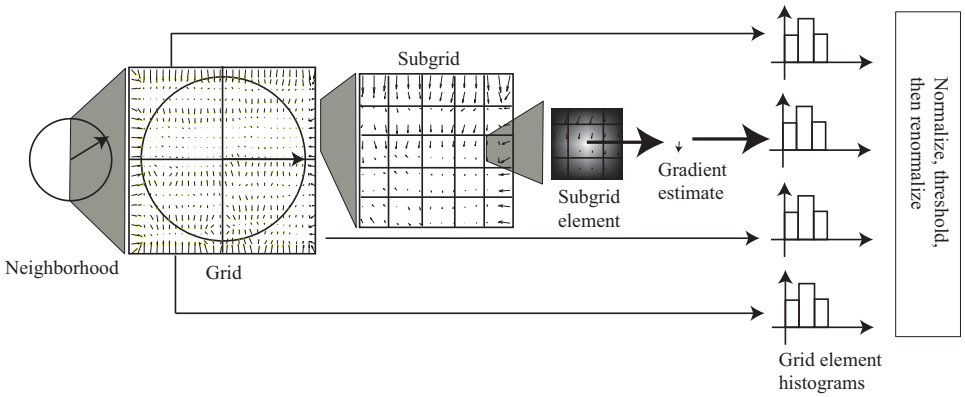


FIGURE 9.4: *To construct a SIFT descriptor for a neighborhood, we place a grid over the rectified neighborhood. Each grid is divided into a subgrid, and a gradient estimate is computed at the center of each subgrid element. This gradient estimate is a weighted average of nearby gradients, with weights chosen so that gradients outside the subgrid cell contribute. The gradient estimates in each subgrid element are accumulated into an orientation histogram. Each gradient votes for its orientation, with a vote weighted by its magnitude and by its distance to the center of the neighborhood. The resulting orientation histograms are stacked to give a single feature vector. This is normalized to have unit norm; then terms in the normalized feature vector are thresholded, and the vector is normalized again.*
**TODO:** Source, Credit, Permission: SIFTPIC

up, however wide the stripe. The natural approach is to take histograms locally, within subpatches of the neighborhood. This leads to a very important feature construction.

A *SIFT descriptor* (for Scale Invariant Feature Transform) is constructed out of image gradients, and uses both magnitude and orientation. The descriptor is normalized to suppress the effects of change in illumination intensity. The descriptor is a set of histograms of image gradients that are then normalized. These histograms expose general spatial trends in the image gradients in the patch but suppress detail. For example, if we estimate the center, scale, or orientation of the patch slightly wrong, then the rectified patch will shift slightly. As a result, simply recording the gradient at each point yields a representation that changes between instances of the patch. A histogram of gradients will be robust to these changes. Rather than histogramming the gradient at a set of sample points, we histogram local averages of image gradients; this helps avoid noise.

There is now extensive experimental evidence that image patches that match one another will have similar SIFT feature representations, and patches that do not will tend not to.

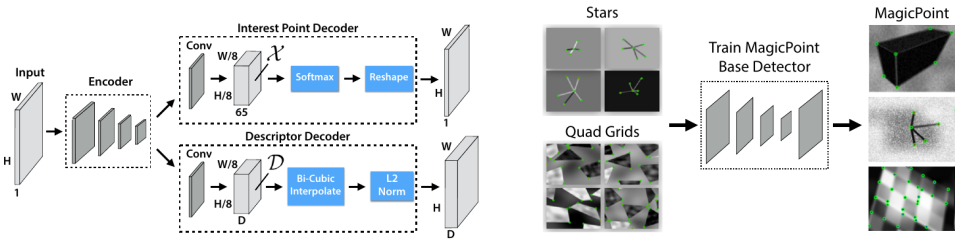


FIGURE 9.5: *SuperPoint uses an encoder with two heads (***left***), one of which predicts the locations of interest points and the other of which predicts a descriptor. The location finder assumes that there is at most one interest point per 8x8 image tile, and predicts which (if any) location is that point. A basic location finder is trained using a cross-entropy loss with a dataset of rendered images where interest point locations are known (***right***).*
**TODO:** Source, Credit, Permission

## 9.2 SUPERPOINT: A LEARNED INTEREST POINT DETECTOR

It turns out the list of properties of interest points is crisp enough that one can learn an interest point finder, and learned interest point finders now are dominant. SuperPoint uses a network architecture that is adapted to fast computation of points and descriptors, with a mixture of learned and non-learned components. This is trained in a series of steps. The first builds an elementary interest point finder. The second uses a clever trick with image transformations to significantly improve the interest point finder. The third refines point positions and descriptors with a matching loss.

### 9.2.1 Network Architecture

First, pass the image through an encoder, which encodes the image with series of convolutional layers, non-linear layers, and three $2 \times 2$ downsampling layers, so that it takes an $H \times W$ image and produces an $H/8 \times W/8 \times 256$ feature block. This block goes to two heads. One finds interest points, the other describes them. The interest point finder is trained discriminatively, by dividing the image into a grid of $H/8 \times W/8$ tiles (each tile is $8 \times 8$ pixels). Now assume there is at most one interest point in any tile. A 65 dimensional one-hot vector encodes where the interest point is if there is one (there are 64 locations for the point, and the last component is one if there isn't a point). The interest point finder maps the original block to an $H/8 \times W/8 \times 65$ block, which is passed through a softmax. Reshaping this with a fixed reshaping procedure gives the predicted location of the interest point. The interest point describer maps the original block to an $H/8 \times W/8 \times 256$ block. This is upsampled using a bicubic interpolation procedure (Section 33.2), and the predicted vector at each location is normalized to a unit vector.

**TODO:** Brief description of bicubic interpolation somewhere

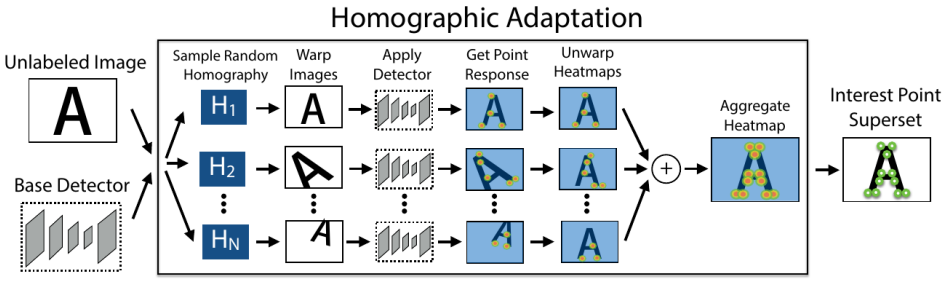


FIGURE 9.6: *The basic location finder of Figure 18 can be significantly improved by exploiting the constraint that interest point predictions should be covariant. The response of the finder to a transformed image, which is a heatmap, should be a transformed version of the response to the original image. Equivalently, apply the finder to a transformed image, and the inverse of the transformation to the resulting heatmap – that heatmap should be the same as the one the detector produces from the original image. This means that a composite finder can be built out of the the basic location finder by predicting heatmaps from images transformed with random (but carefully chosen) homographies, transforming the heatmaps back to the original image frame, then averaging them. Training this composite finder improves the original basic finder, without requiring real data.*
**TODO:** Source, Credit, Permission

### 9.2.2 Finding Interest Points

Generating a large number of relatively simple images with known interest point locations is easy. Use a simple computer graphics program to render collections of polygons; each vertex is an interest point. If any image has more than one interest points in one tile, discard all but one at random. We now have a labelled dataset of images (the labels are interest point locations), and a basic detector can be trained with this.

An interest point detector should be covariant under homographies – the interest points for a transformed image should be obtained by transforming the interest points of the original image. This likely won't be a property of the basic interest point detector, but it can be self-supervised very strongly using this idea. Write $f(\mathcal{I}, \theta)$ for the output of the interest point detector with parameters $\theta$ applied to the image $\mathcal{I}$ (this is a *heat map* – at every pixel location, there is a value giving the probability of an interest point at that location), and $\mathcal{H}(\mathcal{I})$ for the result of applying a homography to $\mathcal{I}$. The output of the detector can be thought of as an image, so a homography can be applied to it. Covariance means that the heat map $\mathcal{H}^{-1}(f(\mathcal{H}(\mathcal{I}), \theta))$ should be the same as $f(\mathcal{I}, \theta)$, at least for reasonable choices of $\mathcal{H}$. For each of the training images above, choose a collection of $N$ homographies at random (taking care with cropping, etc. – details in []), and train the detector which produces the heat map

$$\frac{1}{N} \sum_i \mathcal{H}_i^{-1}(f(\mathcal{H}(\mathcal{I}), \theta)).$$

Training like this has quite strong effects on $\theta$ because the detector receives gradient if (say) an interest point is detected in the wrong place in a (say) rotated version of the original image. It is also an extremely efficient use of data.

### 9.2.3 Refining Detection and Learning to Describe

The refined detector can now be trained to improve interest point detections and to produce descriptions. Take a synthetic image with interest points known and apply a homography. The interest points in the result should be close to those predicted by applying the homography to the interest points in the original image. This property can be imposed with a cross-entropy loss between $\mathcal{H}f(\mathcal{I}, \theta)$ and $f(\mathcal{H}(\mathcal{I}), \theta)$.

Corresponding points in $\mathcal{I}$ and $\mathcal{H}(\mathcal{I})$ should have similar descriptors and pairs of points that don't correspond should have different descriptors. It is easier to impose this on tiles than points. For every pair of tiles, where one comes from $\mathcal{I}$ and the other from $\mathcal{H}(\mathcal{I})$, say the pair corresponds if there is some interest point in the first that maps to a point in the second. Otherwise, the pair does not correspond. Recall that the descriptors are computed on a coarse grid where each location corresponds to a tile (and are then upsampled). Write $\mathbf{d}(t)$ for the descriptor of a tile, and so on. The matching loss is a hinge loss that ensures that, if tiles $t$ and $t'$ correspond, then $\mathbf{d}^T(t)\mathbf{d}(t')$ is positive and greater than some margin, and if they do not, it is negative and less than some margin.

**Resources: Interest Points** *A pretrained version of SuperPoint can be found at* `https://github.com/magicleap/SuperPointPretrainedNetwork`. *There are implementations for TensorFlow () and PyTorch (). HPatches is an evaluation dataset (at* `https://github.com/hpatches/hpatches-dataset`*) which comes with evaluation protocols and benchmarks (at* `https://github.com/hpatches/hpatches-benchmark`*). OpenCV provides an implementation of the Harris corner detector, and procedures to compute SIFT descriptors.*

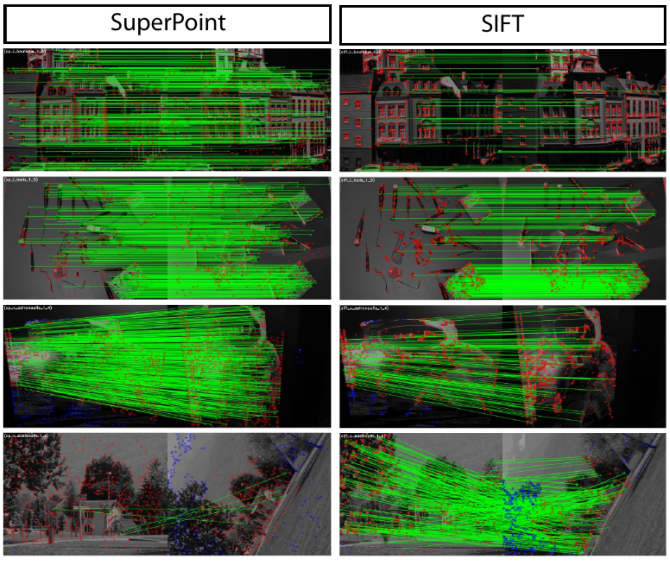


FIGURE 9.7: *SuperPoint produces many good interest point locations together with descriptors that are distinctive.* **Left** *shows SuperPoint detections and matches for four image pairs, and* **right** *for a SIFT based matcher. The images are transformed with a known homography (red dots are detected interest points; blue dots are detected interest points that are outside the field of view of the corresponding image, and so could not have a match; green lines indicate matches). Generally, SuperPoint produces large numbers of interest points that match well. The original SuperPoint is trained with relatively small image rotations, because big rotations are less common in practice, and so handles large image rotations poorly compared to a SIFT based matcher (fourth row).*
**TODO:** Source, Credit, Permission