Chapter 6 – SIFT/RANSAC Image Alignment

Author: Gianmarco Scarano

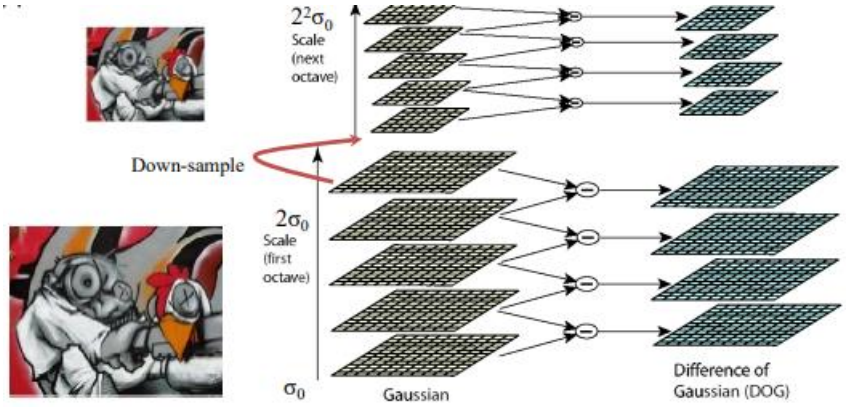
gianmarcoscarano@gmail.com

1. Scale Invariant Feature Transform (SIFT)

In SIFT, we describe both a detector and a descriptor at the same time, in the same pipeline. It maximizes the Difference of Gaussians (DoG) in scale and in space in order to find the same key points independently in each image.

1.1 Feature Detection

Here we apply DoG at different scales (doubling the value of sigma). In order to find features, we apply a Gaussian filter on the same scale (without down sample) with different sigmas.



We can clearly see how in each block, the higher we compute the Gaussian filter, the higher sigma will be. From this we compute the differences (DoG).

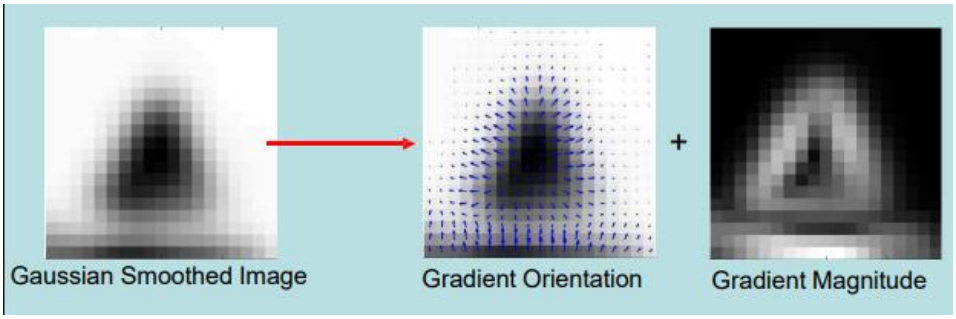
For each point in the DoG level, we compare the level below and above it, finding maxima and minima. If the point is a local extrema, it is a potential keypoint. In this way we know that that keypoint is best represented in that scale. Plus, we discard keypoints which are lower than a certain threshold.

Max/mins from DOG pyramid

Filter by illumination thresholding

Removing edge (keep only corner)

The next thing to do is to assign an orientation to each keypoint through histograms (36 bins each 10 degrees) and compute Gradient Magnitude and Orientation for all levels.

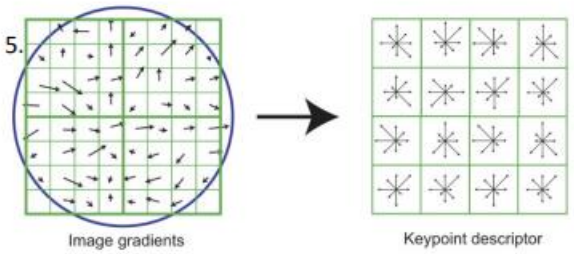


1.2 Local Image Description (SIFT Descriptor)

For each detected keypoint, a location, scale and orientation are assigned.

So, for building a SIFT Descriptor, we use the Gaussian blurred image associated with that keypoint and take image gradients over a 16x16 square. We then have to decide a reference degree (90, 180, 30, etc.) for rotating the angle of the gradient. We do this for all the gradients in the sub-patches.

Then we compute an orientation histogram with 8 orientations bins for each cell bins, as follows:



Finally, the resulting SIFT Descriptor, is explicated as:

$$SIFT = (X, Y, \sigma, \theta, d)$$

Where d is the 128-length vector, formerly known as Descriptor.

SIFT is invariant to rotation because we rotated gradients, it's also invariant to scale and invariant to illumination variation since we didn't take into consideration the magnitude of the gradient.

1.3 Matching

In matching, we need to compare descriptors, even through a simple L2 distance function, like:

$$\big||d1 - d2|\big| < \varepsilon$$

Where ε is a simple threshold value. This is good but can give small distances for ambiguous (incorrect) matches.

A better approach then, would be using the 2nd best SSD match to f1, as follows:

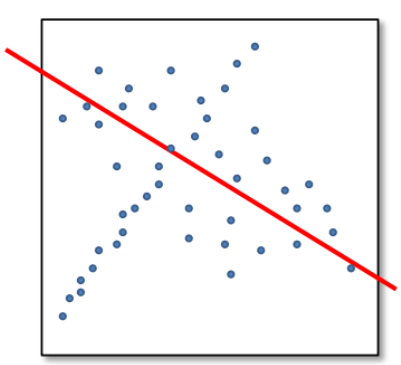
$$\frac{||f1 - f2||}{||f1 - f'2||}$$

Where $f'2$ is the 2nd best matching feature. We can always use a threshold ε in order to validate our results.

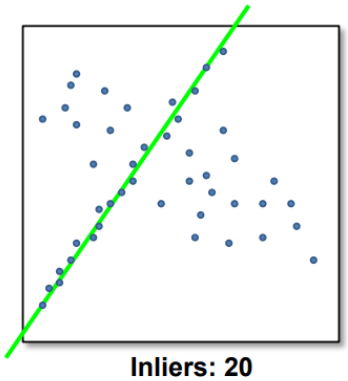
2. Image Alignment Algorithm

Given two images A and B, we compute image features for both images and try to match features.

In order to understand this concept, we can consider the problem of linear regression where we count the number of points that agree with the line (meaning that the points have a small distance from the line) -> These are called 'inliers'.

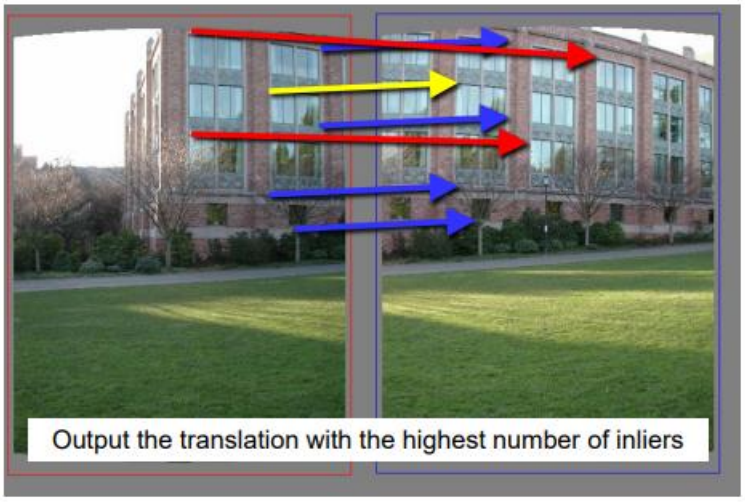


Inliers: 3



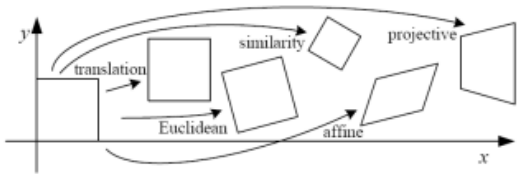
Inliers: 20

But how exactly we find these lines in two images? Well, we try out many lines and keep the best one, which is basically the main point of RANSAC (Random Sample Consensus):



As said before, the idea is that all inliers will agree with each other on the translation vector. Outliers will disagree with each other as well.

More formerly, we randomly choose s samples > We fit a model to these samples > Count the number of inliers that approx. fit the model > Repeat N times > Choose the model that has the largest set of inliers.



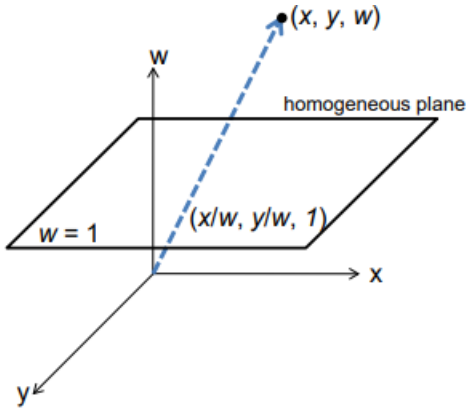
Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{2\times3}$	2	orientation + ···	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{2\times3}$	3	lengths + ···	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{2\times3}$	4	angles + ···	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times3}$	6	parallelism + ···	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times3}$	8	straight lines	

On the left, we can see a table of 2D transformations and the degrees of freedom for each transformation.

translation rotation aspect

We pass from these transformations through a vector T such that $p = (x, y)$ and $p' = (x', y')$ where $p' = T(p)$.

As for translation, for example, it is not a linear operation on 2D coordinates, so how can we shift our image with a vector of coordinates? Very simple, we pass from 2D representation to a 3D representation in order to add the shift coordinates tx and ty (how much we are moving x and y respectively).



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

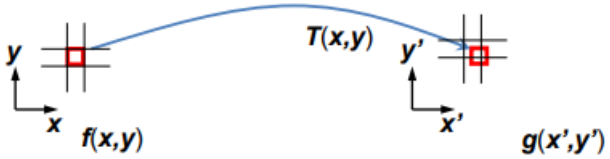
2D *in-plane* rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear/Deformation

Any transformation represented by a 3x3 matrix with last row being [0 0 1] is named ***affine*** transformation, which are essentially a combination of linear transformations and translations.

Going back to the warping problem, we want to send each pixel f(x) to its corresponding location (x', y') = T(x, y) in g(x', y'), but what happens if a pixel lands between two pixels?



Well, we basically do the opposite thing, meaning that we get each pixel g(x', y') from its corresponding location (x, y) = $T^{-1}(x, y)$ in f(x, y). In other words, we take the inverse of the transformation.

Anyways, though, a pixel could come from "between" two pixels. In that case we resample the color value from interpolated source image meaning that we can apply an interpolation filter before sending the pixel (bilinear, cubic, nearest neighbor etc).