SIFT: Introduction

Matching features across different images in a common problem in computer vision. When all images are similar in nature (same scale, orientation, etc) simple corner detectors (/tutorials/harris-corner-detector/) can work. But when you have images of different scales and rotations, you need to use the Scale Invariant Feature Transform.

Why care about SIFT

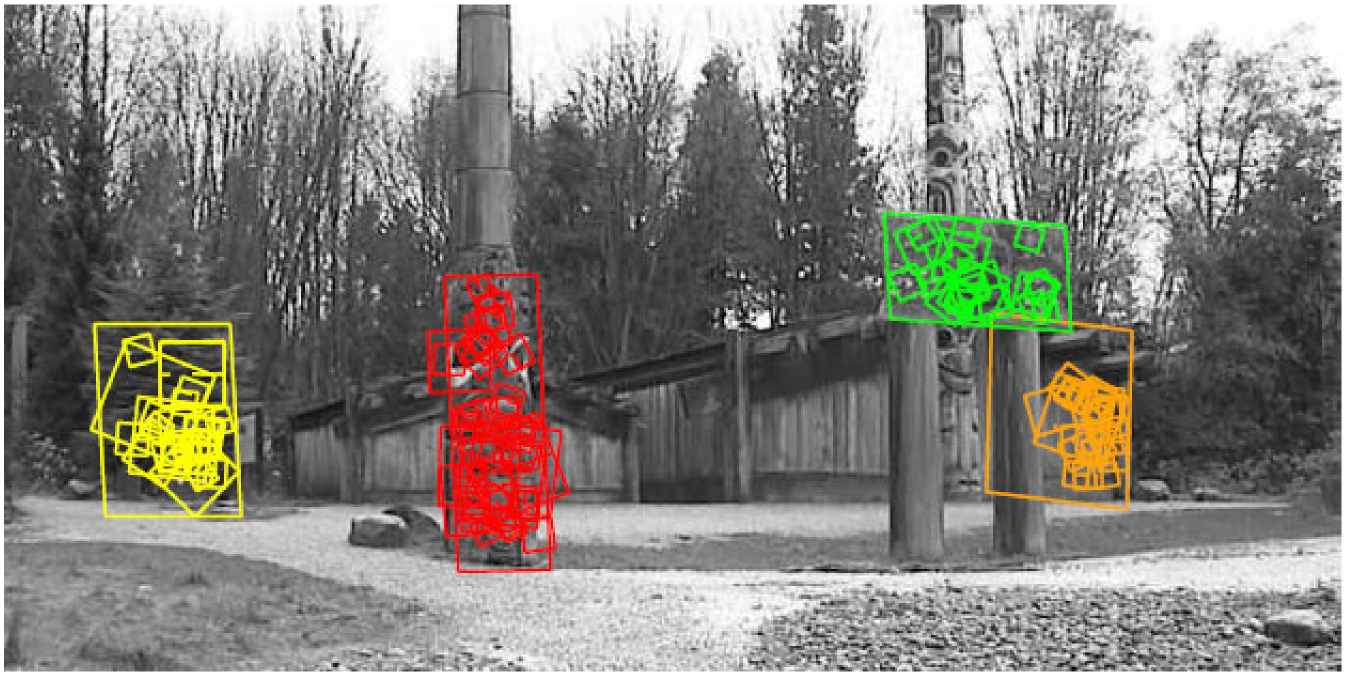
SIFT isn't just scale invariant. You can change the following, and still get good results:

- Scale (duh)
- Rotation
- Illumination
- Viewpoint

Here's an example. We're looking for these:

And we want to find these objects in this scene:

Here's the result:



Now that's some real robust image matching going on. The big rectangles mark matched images. The smaller squares are for individual features in those regions. Note how the big rectangles are skewed. They follow the orientation and perspective of the object in the scene.

The algorithm

SIFT is quite an involved algorithm. It has a lot going on and can become confusing, So I've split up the entire algorithm into multiple parts. Here's an outline of what happens in SIFT.

1. **Constructing a scale space (/tutorials/sift-scale-invariant-feature-transform-scale-space/)** This is the initial preparation. You create internal representations of the original image to ensure scale invariance. This is done by generating a "scale space".

2. **LoG Approximation (/tutorials/sift-scale-invariant-feature-transform-log-approximation/)** The Laplacian of Gaussian is great for finding interesting points (or key points) in an image. But it's computationally expensive. So we cheat and approximate it using the representation created earlier.
3. **Finding keypoints (/tutorials/sift-scale-invariant-feature-transform-keypoints/)** With the super fast approximation, we now try to find key points. These are maxima and minima in the Difference of Gaussian image we calculate in step 2
4. **Get rid of bad key points (/tutorials/sift-scale-invariant-feature-transform-eliminate-low-contrast/)** Edges and low contrast regions are bad keypoints. Eliminating these makes the algorithm efficient and robust. A technique similar to the Harris Corner Detector (/tutorials/interesting-windows-in-the-harris-corner-detector/) is used here.
5. **Assigning an orientation to the keypoints (/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/)** An orientation is calculated for each key point. Any further calculations are done relative to this orientation. This effectively cancels out the effect of orientation, making it rotation invariant.
6. **Generate SIFT features (/tutorials/sift-scale-invariant-feature-transform-features/)** Finally, with scale and rotation invariance in place, one more representation is generated. This helps uniquely identify features. Lets say you have 50,000 features. With this representation, you can easily identify the feature you're looking for (say, a particular eye, or a sign board). That was an overview of the entire algorithm. Over the next few days, I'll go through each step in detail. Finally, I'll show you how to **implement SIFT in OpenCV (/tutorials/implementing-sift-in-opencv/)**!

What do I do with SIFT features?

After you run through the algorithm, you'll have SIFT features for your image. Once you have these, you can do whatever you want.

Track images, detect and identify objects (which can be partly hidden as well), or whatever you can think of. We'll get into this later as well.

But the catch is, this algorithm is patented.

```
.<
```

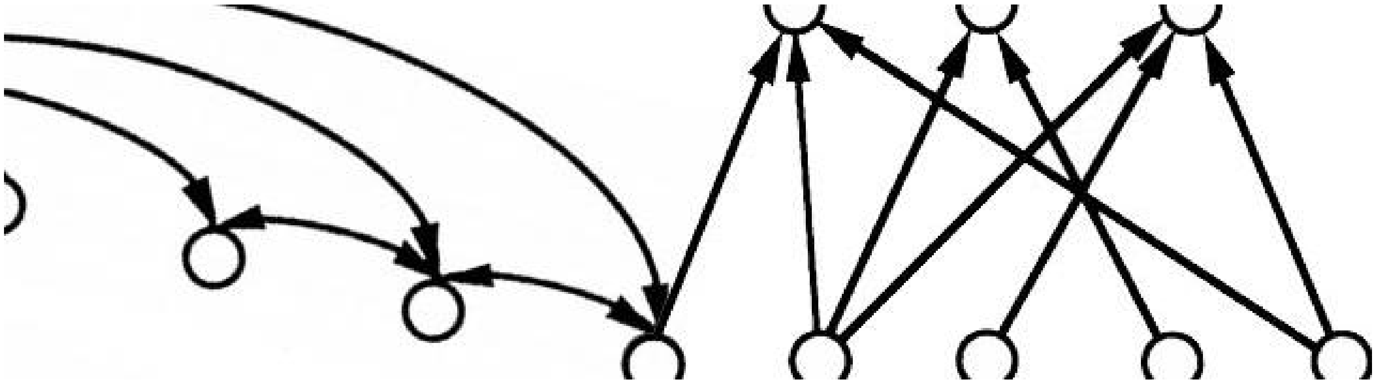
So, it's good enough for academic purposes. But if you're looking to make something commercial, look for something else! [Thanks to aLu for pointing out SURF is patented too]

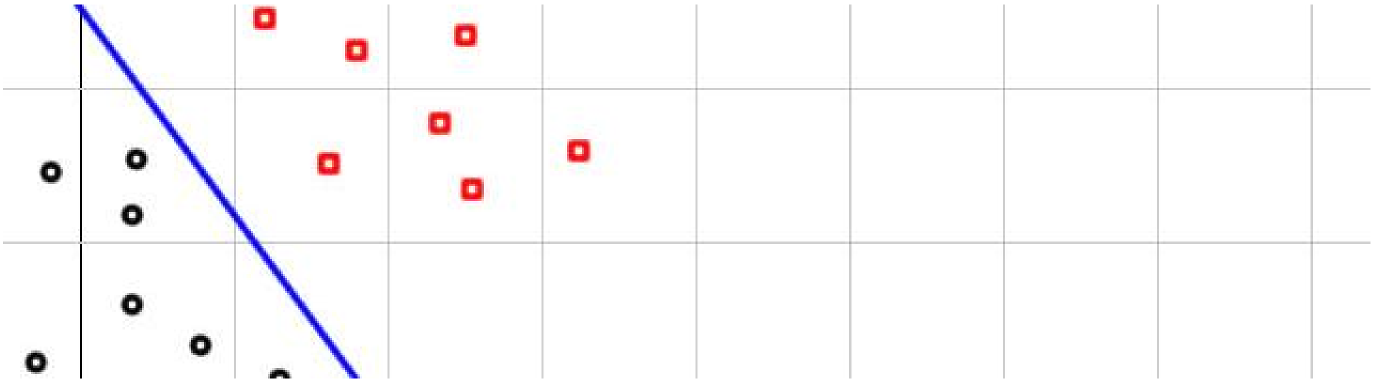
This tutorial is part of a series called **SIFT: Theory and Practice**:

1. **SIFT: Introduction**
2. SIFT: The scale space (/tutorials/sift-scale-invariant-feature-transform-scale-space/)
3. SIFT: LoG approximations (/tutorials/sift-scale-invariant-feature-transform-log-approximation/)
4. SIFT: Finding keypoints (/tutorials/sift-scale-invariant-feature-transform-keypoints/)
5. SIFT: Getting rid of low contrast keypoints (/tutorials/sift-scale-invariant-feature-transform-eliminate-low-contrast/)

6. SIFT: Keypoint orientations (/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/)
7. SIFT: Generating a feature (/tutorials/sift-scale-invariant-feature-transform-features/)

Related posts


7 unique neural network architectures (/tutorials/7-unique-neural-network-architectures/)


A single neuron Dictomizer (/tutorials/a-single-neuron-dictomizer/)

Capturing images with DirectX (/tutorials/capturing-images-with-directx/)

(http://utkarshsinha.com/)

Utkarsh Sinha created AI Shack in 2010 and has since been working on computer vision and related fields as a hobby! Currently he works as a writer of software in Bangalore, India.

The bottom sidebar

AI Shack on Facebook

AI Shack

Like

2,756 people like AI Shack.

Facebook social plugin

Contact

You can contact me through the internet over the following channels:

Facebook (http://facebook.com/aishack/)
Twitter (http://twitter.com/aishack)
Github (http://github.com/aishack)
Email (mailto:utkarsh@utkarshsinha.com)

Where to start?

So you want to get started with AI but you're not sure where. Here are some links to get you started:

- Get started with OpenCV (/tracks/opencv-basics/)
- Track a specific color on video (/tutorials/tracking-colored-objects-in-opencv/)
- Learn basic image processing algorithms (/tracks/image-processing-algorithms-level-1/)
- How to build artificial neurons? (/tutorials/a-single-neuron-dictomizer/)
- Look at some source code (http://github.com/aishack/)