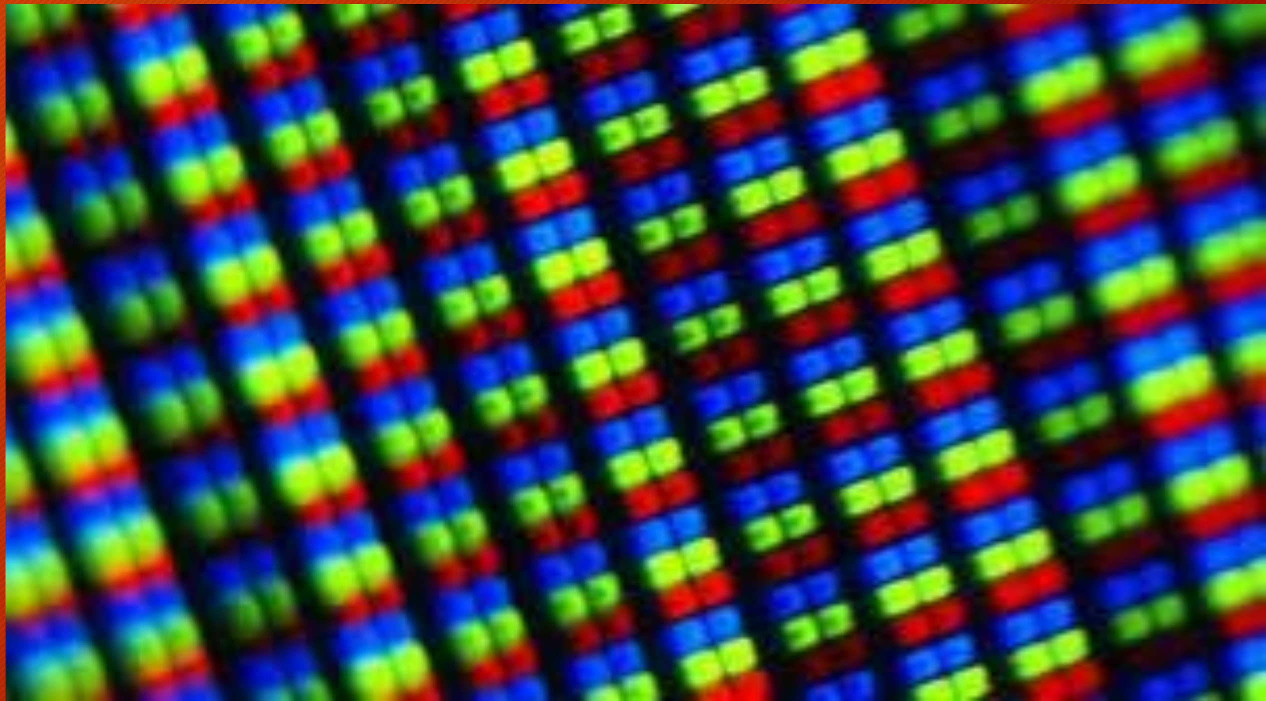
Research methods: Finding your way in computer vision

Thomas Lux

Computer vision

Using images or sequences of images to produce some valuable information

Images consist of thousands of ‘pixels’ where each pixel contains color information



Three steps

Have an idea

Research for relevant work

Create and test the idea

Step One: Having an idea

What can we see that is ‘valuable’ in this image set?

Image 1

Image 2

Image 3

Step Two: Researching for relevant work

What words describe our idea?

Object Recognition from Local Scale-Invariant Features

David G. Lowe
Computer Science Department
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada
lowe@cs.ubc.ca

Abstract

An object recognition system has been developed that uses a new class of local image features. The features are invariant to image scaling, translation, and rotation, and partially invariant to illumination changes and affine or 3D projection. These features share similar properties with neurons in inferior temporal cortex that are used for object recognition in primate vision. Features are efficiently detected through a staged filtering approach that identifies stable points in scale space. Image keys are created that allow for local geometric deformations by representing blurred image gradients in multiple orientation planes and at multiple scales. The keys are used as input to a nearest-neighbor indexing method that identifies candidate object matches. Final veri-

translation, scaling, and rotation, and partially invariant to illumination changes and affine or 3D projection. Previous approaches to local feature generation lacked invariance to scale and were more sensitive to projective distortion and illumination change. The SIFT features share a number of properties in common with the responses of neurons in inferior temporal (IT) cortex in primate vision. This paper also describes improved approaches to indexing and model verification.

The scale-invariant features are efficiently identified by using a staged filtering approach. The first stage identifies key locations in scale space by looking for locations that are maxima or minima of a difference-of-Gaussian function. Each point is used to generate a feature vector that describes the local image region sampled relative to its scale-space co-

Blob Detection

Anne Kaspers
Biomedical Image Sciences
Image Sciences Institute
UMC Utrecht

Abstract

For many high vision purposes, detecting low-level objects in an image is of great importance. These objects, which can be 2D or 3D, are called blobs. Blobs appear in different ways depending on their scale and can be detected using local operations in a multi-scale representation of the image. This paper describes several blob detection methods and applications and tries to make a fair comparison without performing experiments. It shows that blobs can be defined and localized in different ways and that each method has its own strength and shortcomings.

1 Introduction

Automatic detection of blobs from image datasets is an important step in analysis of a large-scale of scientific data. These blobs may represent organization of nuclei in a cultured colony, homogeneous regions in geophysical data, tumor locations in MRI or CT data, etc. This paper presents several approaches for blob detection and applications.

Before going into detail on blob detection, first some definitions of a blob are given. Lindeberg [10] defines a blob as being a region associated with at least one local extremum, either a maximum or a minimum for resp. a bright or a dark blob. Regarding the image intensity function, the spatial extent of a blob is limited by a saddle point, a point where the intensity stops decreasing and starts increasing for bright blobs and vice versa for dark blobs. A blob is represented as a pair consisting of one saddle point and one extremum point.

Hinz [8] just describes a blob as a rectangle with a homogeneous area, i.e. a constant contrast, which becomes a local extremum under sufficient amount of scaling.

Rosenfeld et al. [13] defines a blob as a crossing of lines perpendicular to edge tangent directions, surrounded by 6 or more directions, like in the following picture:

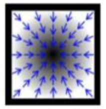


Fig. 1. A blob surrounded by 8 different directions taken from www.wikipedia.org

A third definition of a blob is given by Damerval [4], describing blobs as the largest modulus maxima of the continuous wavelet transform (CWT, see Appendix) along some maxima of interest. The CWT is able to construct a time-frequency representation, offering a good localization of frequencies and time (scale). The exact meaning of modulus maxima and maxima of interest is explained later in the section of the concerning method.

To this point only 2D blob definitions are mentioned. Yang and Parvin [17] give a definition of a 3D blob as being elliptic features in scale-space portioned by a convex hull (boundary of the minimal convex set containing a set of voxels belonging to a blob).

Blobs occur in many shapes and places. For instance, blobs can be found in an image of sunflowers, zebra fish neurons or in an image of a hand. Below, a number of example images are shown.

Fig. 2. A Sunflower field (taken from [10])

Pyramidal Implementation of the Lucas Kanade Feature Tracker
Description of the algorithm

Jean-Yves Bouguet

Intel Corporation
Microprocessor Research Labs
jean-yves.bouguet@intel.com

1 Problem Statement

Let I and J be two 2D grayscaled images. The two quantities $I(\mathbf{x}) = I(x, y)$ and $J(\mathbf{x}) = J(x, y)$ are then the grayscale value of the two images are the location $\mathbf{x} = [x \ \ y]^T$, where x and y are the two pixel coordinates of a generic image point $\mathbf{x}$. The image I will sometimes be referenced as the first image, and the image J as the second image. For practical issues, the images I and J are discret function (or arrays), and the upper left corner pixel coordinate vector is $[0 \ \ 0]^T$. Let n_x and n_y be the width and height of the two images. Then the lower right pixel coordinate vector is $[n_x - 1 \ \ n_y - 1]^T$.

Consider an image point $\mathbf{u} = [u_x \ \ u_y]^T$ on the first image I. The goal of feature tracking is to find the location $\mathbf{v} = \mathbf{u} + \mathbf{d} = [u_x + d_x \ \ u_y + d_y]^T$ on the second image J such as $I(\mathbf{u})$ and $J(\mathbf{v})$ are "similar". The vector $\mathbf{d} = [d_x \ \ d_y]^T$ is the image velocity at $\mathbf{x}$, also known as the optical flow at $\mathbf{x}$. Because of the *aperture problem*, it is essential to define the notion of similarity in a 2D neighborhood sense. Let ω_x and ω_y two integers. We define the image velocity $\mathbf{d}$ as being the vector that minimizes the residual function ϵ defined as follows:

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} \left(I(x, y) - J(x + d_x, y + d_y)\right)^2 . \qquad (1)$$

Observe that following that defintion, the similarity function is measured on a image neighborhood of size $(2\omega_x + 1) \times (2\omega_y + 1)$. This neighborhood will be also called integration window. Typical values for ω_x and ω_y are 2,3,4,5,6,7 pixels.

2 Description of the tracking algorithm

The two key components to any feature tracker are accuracy and robustness. The accuracy component relates to the local sub-pixel accuracy attached to tracking. Intuitively, a small integration window would be preferable in order not to "smooth out" the details contained in the images (i.e. small values of ω_x and ω_y). That is especially required at occluding areas in the images where two patchs potentially move with very different velocities.

The robustness component relates to sensitivity of tracking with respect to changes of lighting, size of image motion,... In particular, in oder to handle large motions, it is intuively preferable to pick a large integration window. Indeed, considering only equation 1, it is preferable to have $d_x \leq \omega_x$ and $d_y \leq \omega_y$ (unless some prior matching information is available). There is therefore a natural tradeoff between local accuracy and robustness when choosing the integration window size. In provide to provide a solution to that problem, we propose a pyramidal implementation of the classical Lucas-Kanade algorithm. An iterative implementation of the Lucas-Kanade optical flow computation provides sufficient local tracking accuracy.

2.1 Image pyramid representation

Let us define the pyramid representsation of a generic image I of size $n_x \times n_y$. Let $I^0 = I$ be the "zero$^{\text{th}}$" level image. This image is essentially the highest resolution image (the raw image). The image width and height at that level are defined as $n_x^0 = n_x$ and $n_y^0 = n_y$. The pyramid representation is then built in a recursive fashion: compute I^1 from I^0, then compute I^2 from I^1, and so on... Let $L = 1, 2, \ldots$ be a generic pyramidal level, and let I^{L-1} be the image at level $L - 1$. Denote n_x^{L-1} and n_y^{L-1} the width and height of I^{L-1}. The image I^{L-1} is then defined as follows:

$$\begin{aligned} I^L(x, y) \ = \ & \frac{1}{4} I^{L-1}(2x, 2y) + \\ & \frac{1}{8}\left(I^{L-1}(2x-1, 2y) + I^{L-1}(2x+1, 2y) + I^{L-1}(2x, 2y-1) + I^{L-1}(2x, 2y+1)\right) + \\ & \frac{1}{16}\left(I^{L-1}(2x-1, 2y-1) + I^{L-1}(2x+1, 2y+1) + I^{L-1}(2x-1, 2y+1) + I^{L-1}(2x+1, 2y+1)\right). \end{aligned} \qquad (2)$$

Step Three: Creating and testing our idea

We turn our idea into words a computer understands and then run different images through it to see if it actually produces the 'valuable' information we originally wanted!

Research methods: Computer Vision

There are many more obstacles to be faced on the way, but those are the details that we all deal with in our own fields.

Step One: Have an idea

Step Two: Research relevant work

Step Three: Create and test the idea