# Visual Understanding 1 Coursework

Garoe Dorta-Perez
CM50248: Visual Understanding 1
Unit Leader: Peter Hall
Bath University

January 15, 2015

## 1 Introduction

The main objectives of this unit are to acquire the fundamentals of Computer Vision. In order to gather this skills, we will attempt to build a stereo reconstruction system. Using a bottom-up approach the following steps will be taken:

1. Image convolution, transforming using kernel matrices.
2. Feature construction, detecting scale invariant interest points in images.
3. Matching, using the previous points to match two images.
4. Stereo reconstruction, rebuilding 3D points from 2D matched points in stereo images.

## 2 Literature review

## 3 Convolution

Transforming images has artistic and practical applications, for instance blurring certain parts in a person face to hide defects or detecting borders with differential operators. For simplicity we will restrict ourselves to gray images, defined as two-dimensional $n \times m$ matrices where each element represents the intensity, and is in the range $\{0, 1\}$ where 0 is black and 1 is white. A convolution kernel $k$, is defined as a $k_1 \times k_2$ matrix, such that $k_1 \leqslant n$ and $k_2 \leqslant m$. When an image $I_0$ is convoluted with a kernel, a new image $I_1$ is generated, such

that each element in the second image is calculated as follows WRITE IT AS EQUATION OR EXPLAIN IT OTHERWISE. The advantage in this definition is that, it gives a common framework for every possible transformation; so we only need to change the kernel to fit our needs. However, we need to specify what will happen to the borders of the matrix, leave them as they are, set them to zero, compute circularly, etc; and where does the convolution start with kernel with an even number of columns or rows.

An interesting property of convolutions is the *convolution theorem*, which states that $g * h = F^{-1}(F(g) \cdot F(h))$. This property allows to transform the images into the Fourier domain, apply the convolution, transform back and obtain the same results with less computational cost. Below we show some examples of our convolution code, as well as a table comparing the performance of our initial code, our code with the Fourier transform and Matlab's implementation.

(a) myconv. (b) myconvFFT. (c) Matlab's conv2.

Figure 1: Comparative of the different convolution codes with a horizontal sovel kernel $k = [-1, 1]$.

| | myconv | myconvFFT | Matlab's conv2 |
|---|---|---|---|
| Time | 0.658s | 0.04s | 0.002s |

Table 1: Performance comparative of the convolution methods.

In Figure 1 we see that the convolution theorem holds, and in Table 1 that performing the operations in the Fourier domain and transforming back is more efficient.

CITE FOURIER TRANSFORM AND CONVOLUTION THEOREM

# 4 Features

Being able to detect features in images that are invariant to rotations, translations, scale and changes in illumination is an useful tool, with many applications such as image matching. SIFT features CITE LOWE PAPER provide a fairly robust method to detect points in an image with the aforementioned restrictions. A broad overview of the algorithm is:

1. **Extrema detection:** maximum and minimum intensity points are detected in a difference-of-Gaussian pyramid built with the input image.
2. **Keypoint refinement:** point centrer in interpolated and weak points are rejected.
3. **Orientation assignment:** orientation and strength of each keypoint are calculated.
4. **Descriptor calculation:** a 128 feature vector is computed for every element.

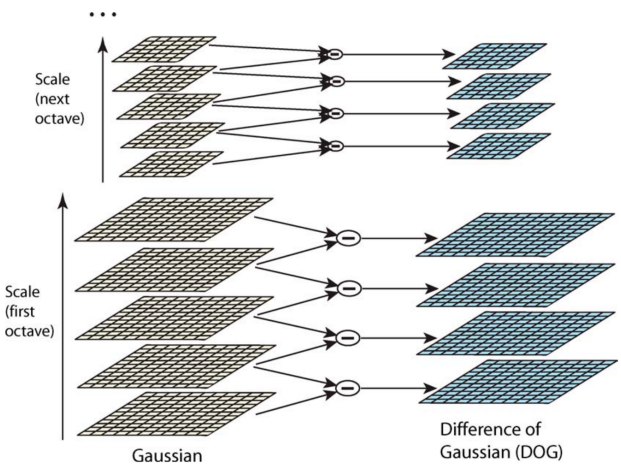



Figure 2: Pyramid of difference-of-Gaussian CITE LOWE.

Figure 3: Example of a scale of a difference-of-Gaussian.

**Extrema detection** is computed using a difference-of-Gaussian (DOG) structure, as shown in Figure 2, first a scale of blurred images is constructed. The input image $I$ is convoluted with a Gaussian kernel $G$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{1}$$

where $L$ is the output image, $*$ the convolution operator, $x$ and $y$ the pixel indices, and $G$ and Gaussian kernel defined as:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{2}$$

where $\sigma$ is the standard deviation. Next, the DOG structure is computed as a difference of several $L$:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma), \tag{3}$$

where $D$ is the output image and $k$ is a constant multiplicative factor. Note that to provide an efficient implementation, the convolution and difference can be computed in the Fourier domain using the convolution theorem explained in Section 3. A example on how this DOG structure looks like is shown in Figure 3, where the sigma increases from left to right and from top to bottom. The keypoints selected will be those who are a maximum or a minimum in a $3 \times 3 \times 3$ cube region, so in their scale and the ones above and below. Selecting locations using this scheme allows to find interested points regardless of their scale, as we progress towards the top of the pyramid, the local maximum or minima will be produced by bigger objects in $I$. In our implementation only one scale of the pyramid is calculated.

Keypoint refinement is done with an optimization step which removes weak points and also improves their position. The new keypoint location $\hat{\mathbf{x}}$ will be:

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x^2}} \frac{\partial D}{\partial \mathbf{x}}, \tag{4}$$

where $\mathbf{x}$ is the previous location and $D(\mathbf{x})$ is approximated with a Taylor expansion in the form of:

$$D(\mathbf{x}) = D + \frac{\partial D'}{\partial \mathbf{x}}\mathbf{x} + 0.5\mathbf{x}' \frac{\partial^2 D}{\partial \mathbf{x^2}}\mathbf{x}. \tag{5}$$

Keypoints are rejected if they are weak or in an edge, any extrema is defined as weak if $D(\mathbf{x}') < 0.03$, where $D(\hat{\mathbf{x}})$ is:

$$D(\hat{\mathbf{x}}) = D + 0.5\frac{\partial D'}{\partial \mathbf{x}}\hat{\mathbf{x}}. \tag{6}$$

Edge elimination is performed when $R(\mathbf{H}) > f(r)$, where $\mathbf{H}$ is a $2 \times 2$ Hessian matrix, $r = 10$, $R(\mathbf{H})$ is defined as:

$$R(\mathbf{H}) = \frac{(D_{xx} + D_{yy})^2}{D_{xx}D_{yy} - (D_{xy})^2} \text{ where } D_{\alpha\beta} = \frac{\partial^2 D}{\partial \alpha \partial \beta} \text{ for } \alpha, \beta = x, y \tag{7}$$

and

$$f(r) = \frac{(r+1)^2}{r}. \tag{8}$$

**Orientation assignment** measures the point orientation using local images properties. The objective is that if a point is rotated in another image, the final descriptor will be the

same in both. A gradient magnitude $m(x, y)$ and orientation angle $\theta(x, y)$ are computed as follows:

$$\begin{aligned} m(x,y) =& \sqrt{(L(x+1,y,\sigma) - L(x-1,y,\sigma))^2 + (L(x,y+1,\sigma) - L(x,y-1,\sigma))^2}, \\ \theta(x,y) =& tan^{-1}\left((L(x,y+1,\sigma) - L(x,y-1,\sigma)) \,/\, (L(x+1,y,\sigma) - L(x-1,y,\sigma))\right), \end{aligned} \tag{9}$$

where $L$ is chosen to has the same sigma as the keypoint.

**Descriptor calculation** uses orientation histograms from sample points near the extrema. For each keypoint a $16 \times 16$ region around it is chosen and the orientations for each pixel are calculated. $4 \times 4$ histograms are computed with 8 bins each; this 8-dimensional values are placed in an array forming a 128 element feature vector for each keypoint. Normalizing the vector to unit length improves invariance to illumination changes.

# 5 Matching

In the image matching area an image $I_0$ is to be found under a certain affine transformation in another image $I_1$. Due to camera measurement errors, changes in illumination and other factors, a global optimal solution is computationally expensive. The RANSAC (Random Sample Consensus) CITE RANSAC algorithm is a simple and fast approach to perform this task. Despite being originally designed to estimate the parameters that fit a mathematical model to a dataset which contains outliers, it can also be used to estimate the transformation $H$ that matches $I_0$ and $I_1$. A general overview of the algorithm is given below:

**Algorithm 1:** RANSAC

**Data**: A pair of images $I_0$ and $I_1$, $k$ number of iterations, distance threshold $t$.
**Result**: Homography matrix $H$.
$f_0$, $f_1$ = detectKeypoints($I_0$, $I_1$);
$m$ = matchFeatures($f_0$, $f_1$);
$H = identity$;
**for** $i \leftarrow 0$ **to** $k$ **do**
  $rp$ = pickRandomPoints($m$);
  $newH$ = computeHomography($rp$);
  **if** *newH is better than H* **then**
    $H = newH$;
  **end**
**end**

To detect and match the keypoints, our implementation uses SURF features CITATION. We want to calculate the matrix $H$ that transforms from one image to the other, such that:

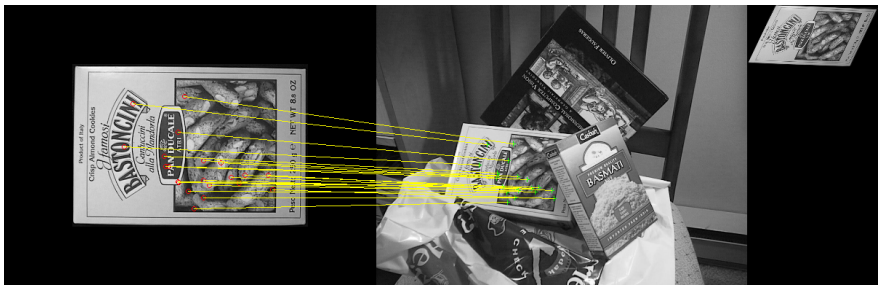
$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{10}$$

where $u$, $v$ and $w$ are the scaled pixel coordinates in $I_1$ and $x$, $y$ and $z$ the scaled pixel coordinates in $I_0$. Collecting $H$ in rows, $H_i = \begin{bmatrix} h_{i1} & h_{i2} & h_{i3} \end{bmatrix}$ for $i = \{1, 2, 3\}$ and $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}$ we can rewrite Equation 10 into:

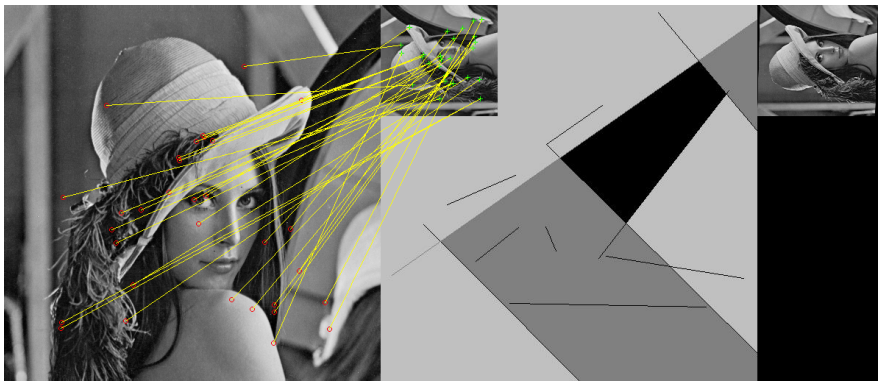
$$\begin{aligned} u = H_1\mathbf{x},\ v = H_2\mathbf{x},\ w = H_3\mathbf{x}, \\ \text{enforcing } w = 1, \\ u^* = \frac{H_1\mathbf{x}}{H_3\mathbf{x}},\ v^* = \frac{H_2\mathbf{x}}{H_3\mathbf{x}},\ \text{hence} \\ u^*H_3\mathbf{x} - H_1\mathbf{x} = 0, \\ v^*H_3\mathbf{x} - H_2\mathbf{x} = 0, thus \\ \begin{bmatrix} -\mathbf{x} & \mathbf{0} & u^*\mathbf{x} \\ \mathbf{0} & -\mathbf{x} & v^*\mathbf{x} \end{bmatrix} \begin{bmatrix} H_1^T \\ H_2^T \\ H_3^T \end{bmatrix} = \mathbf{0}^T, \end{aligned} \tag{11}$$

where $\mathbf{0} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$. As shown above, each match provides two equations to solve $H$. Since, we enforce the scale to be a homogeneous coordinate, $h_{33} = 1$ is no longer an unknown. For eight unknowns, eight equations are needed, so four points in each image are to be matched. In matrix notation, we will stack all the equations in a matrix $A$, such that $A\mathbf{h} = 0$, where $\mathbf{h}$ is $H$ reshaped into a column vector. Singular value decomposition of $A$ can be computed to solve the system, $A = UZV'$, where $U$ and $V$ are unitary, $Z$ is rectangular diagonal and $\mathbf{h} = V_{i9}$ for $i = \{1, 2, \ldots, 9\}$.

The metric to define if a $newH$ is better than a previously calculated $H$ is double; each match keypoint in $I_0$ is transformed with $newH$, the euclidean distance from the matched point in $I_1$ is calculated, and if it is below a threshold $t$, we count it as a good match. $newH$ is considered an improvement over $H$, if it produces more good matches than $H$ or the same number although with lower error. Some results obtained with our code are shown in Figure 4.



(a) A synthetic test.



(b) Matching a box, images from SIFT paper CITE.

Figure 4: Samples of $H$ computation, left is $I_0$, centre is $I_1$ and right is $I_0 * H$. Keypoints matching between input images is shown for clarity.

# 6 Reconstruction

3D stereo reconstruction from a par of cameras involves three steps:

1. **Camera calibration:** compute camera model parameters.
2. **Fundamental matrix:** estimate an $F$ matrix such that $\mathbf{u_1}'F\mathbf{u_2} = 0$, where $\mathbf{u_1}$ and $\mathbf{u_2}$ are corresponding points in two images.
3. **Reconstruct 3D points:** once constrained by $F$, triangulate points position from the two images.

We will model our cameras using the pinhole camera model, in this framework a world 3D point $\mathbf{x} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}$ is projected into a 2D image point $\mathbf{u} = \begin{bmatrix} u & v & 1 \end{bmatrix}$ using the following equations:

$$\mathbf{u}' = P\mathbf{x}', \tag{12}$$

$$\mathbf{u}' = K[R|\mathbf{t}]\mathbf{x}', \tag{13}$$

where $K$ is

$$\begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

with $f_x$ and $f_y$ are focal length parameters and $p_x$ and $p_y$ is the principal point, $P$ is a $3 \times 4$ projection matrix, $R$ is a $3 \times 3$ rotation matrix, $t$ is a translation column vector, making $[R|t]$ a $3 \times 4$ matrix. Note that $\mathbf{x}$ and $\mathbf{u}$ have a homogeneous coordinate. $K$ and $[R|\mathbf{t}]$ is commonly referred as camera intrinsics and extrinsics matrix respectively, and the estimation of their elements is known as **camera calibration**.

Equation 12 can be solved following equivalent steps as to those used in Equations 10 and 11, although with an extra coordinate in $\mathbf{x}$ and regarding $P$ as equivalent to $H$ with an added column. Therefore we can again stack several points in a matrix $A$, yet the increased number of unknowns determine that six points are needed to estimate $P$:

$$A\mathbf{p} = 0. \tag{15}$$

Once we have solved the aforementioned equation with SVD, we need to decompose $P$ into $K[R|t]$. Since $K$ is an upper triangular matrix, we can use an a $RQ$ decomposition CITE on $P$, where $R$ is upper triangular and $Q$ is orthogonal. For the decomposition only the first three columns in $P$ will be used, and $\mathbf{t} = K^{-1}P_4$, where $P_4$ is the last column in $P$.

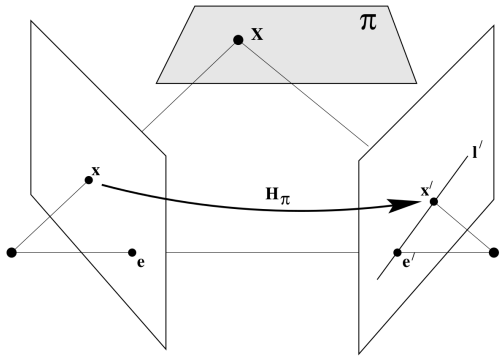



Figure 5: A point $\mathbf{x}$ in the left image is mapped via a plane $\boldsymbol{\pi}$ to a point $\mathbf{x}'$ in the right image.

Epipolar lines and the **fundamental matrix** $F$ are useful tools in stereo reconstruction. In Figure 5 the geometrical relation of the projection a 3D world point $\mathbf{X}$ in two cameras is shown. We define $\mathbf{e}$ and $\mathbf{e}'$ as the epipolar points (epipoles), $F$ is a matrix that satisfies $\mathbf{u_1}'F\mathbf{u_2} = 0$, where $\mathbf{u_1} = \begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix}$ and $\mathbf{u_2} = \begin{bmatrix} u_2 & v_2 & 1 \end{bmatrix}$ are matching points in the left and right images respectively, and $\mathbf{l}'$ is a line such that $\mathbf{l}' = F\mathbf{x}$.

To calculate $F$ an approach in Equations 10 and 11 can be used again. However, in the equation $A\mathbf{f} = 0$, this time $A$ will have following configuration:

$$\begin{bmatrix} u_{l1}u_{r1} & v_{l1}u_{r1} & u_{r1} & u_{l1}v_{r1} & v_{l1}v_{r1} & v_{r1} & u_{l1} & v_{l1} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_{l8}u_{r8} & v_{l8}u_{r8} & u_{r8} & u_{l8}v_{r8} & v_{l8}v_{r8} & v_{r8} & u_{l8} & v_{l8} & 1 \end{bmatrix} \mathbf{f} = 0, \tag{16}$$

where the $\mathbf{f}$ is $F$ reshaped in a column vector and the $l$ and $r$ indices represent points from the left and right images respectively; $F$ can be estimated using SVD.

Since we have calculated all the necessary parameters for each camera and their relationship, we can now proceed to stereo reconstruction. Below a possible implementation in pseudocode in presented:

**Algorithm 2:** Stereo Reconstruction

**Data**: A pair of images $I_0$ and $I_1$, projection matrices for two cameras $P_0$ and $P_1$.
**Result**: Cloud of 3D points $\mathbf{X}$.
$f_0$, $f_1$ = detectKeypoints($I_0$, $I_1$);
$m$ = matchFeatures($f_0$, $f_1$);
**for** $i \leftarrow 1$ **to** $numMatches$ **do**
    $A$ = constructA($m(i)$);
    $X(i)$ = SVD($A$);
**end**
displayCloud(X);

For this task we have a set of correspondences:

$$\mathbf{u_i} = P\mathbf{x}, \quad \mathbf{u_i'} = P'\mathbf{x}, \tag{17}$$

where the unknown is the 3D world object position $\mathbf{x}$. Clustering the rows of $P$, such as $P_i = \begin{bmatrix} p_{i1} & p_{i2} & p_{i3} & p_{i4} \end{bmatrix}$ for $i = \{1, 2, 3\}$, and equivalently for $P'$.

$$\begin{aligned} u = P_1\mathbf{x}, \quad v &= P_2\mathbf{x}, \quad 1 = P_3\mathbf{x}, \\ u' = P_1'\mathbf{x}, \quad v' &= P_2'\mathbf{x}, \quad 1 = P_3'\mathbf{x}, \end{aligned} \tag{18}$$

from the third equations we have $\mathbf{x} = \frac{1}{P_3}$ and $\mathbf{x} = \frac{1}{P_3'}$ respectively,

$$\begin{aligned} uP_3\mathbf{x} - P_1\mathbf{x} = 0, \quad vP_3\mathbf{x} - P_2\mathbf{x} = 0, \\ u'P_3'\mathbf{x} - P_1'\mathbf{x} = 0, \quad v'P_3'\mathbf{x} - P_2'\mathbf{x} = 0, \end{aligned} \tag{19}$$

$$\begin{bmatrix} uP_3 - P_1 \\ vP_3 - P_2 \\ u'P_3' - P_1' \\ v'P_3' - P_2' \end{bmatrix} \mathbf{x}' = \mathbf{0}^T, \tag{20}$$

which can be solved with a singular value decomposition.