Computer Vision Note

Sagar Ojha

September 19, 2024

Contents

Chapter 1

Basic Computer Vision

1.1 Learning Resources

First Principles of Computer Vision and Introduction to Computer Vision are good places to start. The former one introduces mathematics that governs a certain idea and then develops the algorithm and its implementation in discrete space with adequate proofs and derivations while the latter one provides brief mathematical idea about the algorithms and provides some exercise and quizzes to test the knowledge. I would recommend starting with the latter one and watching some videos from the former one to get more idea about the proof of the algorithm.

1.2 Image Processing

We will use grayscaled image for our works. The idea can be expanded to RBG images as well. Image (or image intensity) is basically a function of (x, y). Grayscale image can be obtained by some transformation (or function) applied on the pixel.

1.2.1 Filters

Filters are used to modify an image and extract features from image. We will be using convolution filters. Convolution is a cross-correlation but with the function first reflected about y-axis. Convolution is linear and shift invariant operation (or system). Often, we won't know what we the input function is being convolved with. So, if we want to know the function that the input convolutes with, then we can pass in unit impulse function to convolute with the unknown function. The resulting function will be the unknown function. Thhrefore, we get to know the previously unknown convolution function.

The convolution operation over 2D-discrete functions can be implemented using matrices called convolution masks/kernel/filter represented by h in 1.1.

$$g[i,j] = \sum_{m=1}^{M} \sum_{n=1}^{N} f[m,n]h[i-m, j-n] \tag{1.1}$$

Image functions/matrices are the inputs and we convolute them with the kernel matrices. Box, fuzzy, gaussian (which is also fuzzy but is standard or formalized) filters are applied using convolution. Gaussian filters can be separated such that the convolution time complexity can be minimized.

In general, this is the roadmap that led to matrix filters: we were motivated to modify images which led to the start of developing convolution filters. We started learning the 1D continuous convolution filter and then developed 2D discrete convolution filter. In 2D, the discrete convolution filter is implemented using a matrix. Also, in 1D discrete system, the filter would be realized using an array.

1.2.2 Edge Detection

We'll develop a Canny Edge detector which is a very popular algorithm to find the edges in an image. The first thing is to blur the image (yep the grayscale image) to reduce noise in the image. This is also a type of "modification" of image which is also done using convolution. The amount of blur will definitely impact the edge detection and blurring can be modified to our need. We'll use

Gaussian blur as we assumed the noise is random and in such case Gaussian blur will perform the best. **gaussian_filter(σ)** function will take in the standard deviation as the input and return the Gaussian kernel. Size of the kernel $\approx 2\pi\sigma$. **convolution(f, h)** will take matrix **f** and convolute with another input matrix **h** and return the resulting matrix. An alternative to Gaussian filter could be a fuzzy filter.

The next step is to calculate the image gradient magnitude and direction. Sobel kernel could be used to get the first derivative of the image in horizontal and vertical directions. Sobel operation is also an edge detection method in itself. The kernel is convolved with the image which was obtained after filtering.

$$\nabla_x = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \nabla_y = \frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.2}$$

$$\nabla_{mag} = \sqrt{\nabla_x^2 + \nabla_y^2}, \quad \nabla_{dir} = \arctan 2(\nabla_y, \nabla_x) \tag{1.3}$$

Gradient images don't have sharp or thin edges. So the next step is to find the local maximum pixel value in the direction of the gradient. A full scan of the image has to be performed. The local maximum will be kept as a possible edge and other pixels are zeroed. This is called non-maximum suppression method. 1D laplacian operator in the gradient direction could be implemented as well to get the local maximum from a sharp zero crossing. We will use a naive method of comparing the neighboring pixels rather.

After the non-maximum suppression, hysteresis thresholding is applied. 2 threshold values are chosen. If a pixel is above the higher one, the pixel is marked as a strong edge, and if it is below the lower value, the pixel is marked as not an edge. If the pixel is in between the threshold values, then it gets marked as a weak edge. Finally, if the weak edge is connected to a strong edge, then the weak edge is marked to be a strong edge as well; marking the weak edge to a strong edge will help connect the weak edges which were at the neighborhood of the newly marked strong edge. After a full scan, the strong edges are considered as definite edge. This is what the algorithm outputs as the edge in an image.

1.2.3 Corner Detection

We'll develop a Harris corner detector to find the edges in an image. There can be various routes to implement the same idea. Described here is the easiest method. Check out this video and this slide to get the basic idea.

The goal ultimately boils down to evaluating a *response function*

$$R = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 \tag{1.4}$$

for a pixel where λ_1 & λ_2 are the eigen values of

$$M = \begin{bmatrix} \sum_{p\in P} I_x I_x & \sum_{p\in P} I_x I_y \\ \sum_{p\in P} I_x I_y & \sum_{p\in P} I_y I_y \end{bmatrix} \tag{1.5}$$

and $0.04 \leq \kappa \leq 0.06$ is a weighting term. $p \in P$ refers to the pixel p in a window P. The eigen values for the gradient covariance matrix in Eq. 1.5 captures the amount of the distribution of the gradient of image within the window P in an arbitrary directions. That is to say, if $\lambda_1 \sim \lambda_2$, and the eigen values are small, then the image is flat since the gradients in x and y would be small. If $\lambda_1 >> \lambda_2$ or $\lambda_2 >> \lambda_1$, then there is an edge in the window since one of the gradients is large. If λ_1 and λ_2 are large and λ_1 λ_2, then there exists a corner in the window P. R in Eq. 1.4 basically captures this relationship between the eigen values. The pixel is a corner if $R > T$, where T is a threshold value; T is usually selected to be 5% of the maximum R value.

We can be clever and implement this method as follows. (Note that the steps are not exactly the same steps mentioned in the method)

- Compute the gradient of the image in x and y, i.e., I_x and I_y,
- Compute the products of the gradients, i.e., I_{x^2}, I_{y^2}, and I_{xy},
- Perform a Gaussian convolution; this results in the same effect as summing the products of the gradients
  $S_{x^2} = G_\sigma I_{x^2}$, $S_{y^2} = G_\sigma I_{y^2}$, $S_{xy} = G_\sigma I_{xy}$

- Compute $det = S_{x^2}S_{y^2} - 2S_{xy}$ and $tr = S_{x^2} + S_{y^2}$
- Compute $R = det - \kappa(tr)^2$. This gives the response function for "all pixels"
- Loop over R and threshold to get the corners in the image; the loop will give corner in the image and not just the window because of this implementation

1.3 Hough Transform

1.3.1 Detecting Lines

Lines represented as

$$y = mx + c \tag{1.6}$$

can be parametrized as

$$x\cos\theta + y\sin\theta = \rho \tag{1.7}$$

where ρ is the perpendicular distance from the origin to the line and θ is the angle that the perpendicular line to Eq. 1.6 makes with the horizontal axis. In the *Hough* space, a point (x, y) is represented as a sinusoid given by Eq. 1.7.

In order to derive the parameterized representation in Eq. 1.7, consider a unit vector $[\cos\theta \ \sin\theta]^{\mathrm{T}}$ and the vector $[x \ y]^{\mathrm{T}}$ whose tail is the origin and the head is at (x, y) with the magnitude of ρ. Projecting $[x \ y]^{\mathrm{T}}$ onto $[\cos\theta \ \sin\theta]^{\mathrm{T}}$, one gets

$$\begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix} \cdot \begin{bmatrix}x \\ y\end{bmatrix} = \rho, \tag{1.8}$$

$$x\cos\theta + y\sin\theta = \rho. \tag{1.9}$$

Considering the origin of the image at the lower-bottom part, the image can be swept across with $0° \leq \theta \leq 360°$ and $0 \leq \rho \leq \ell$ where ℓ represents the diagonal length of the image; diagonal length of the image can be considered as the number of pixels along the image diagonal for the implementation. Create an accumulator bin/matrix of appropriate size depending on the resolution as well as range of both θ and ρ. Then, collect the votes from each **edge points**. The cells with the largest votes represents the parameters of the line that the edges correspond to. One may draw the line using the parameters.

1.3.2 Detecting Shapes

Prior to running the detection online, we need to construct a ϕ table using the edges of the shape that we'd like to detect. ϕ table is constructed from the gradient orientation ϕ_i for $0° \leq i \leq 360°$ of the edge and the vector $\vec{v}_n$ from the edge point (x, y) to the reference/center location (x_c, y_c) for the object. Note that it is not a scale and orientation invariant detection technique.

When online, get the edge image and for all the edge pixels, obtain the gradient orientation. At each edge pixel, use the gradient orientation info to look up to the ϕ table; there may be multiple $\vec{v}_n$'s at the ϕ_i location/index. Vote to the cell that is located at $\vec{v}_n$ displacement away from the edge pixel. The reference location for the shape will get the highest vote.

1.4 Stereo Vision

1.4.1 Correspondence & Disparity

For a coplanar stereo vision system, the 2D correspondence problem reduces to a simple 1D correspondence problem. In the code, we assume that the epipolar lines are parallel to the rows of the images; that is, search the pixel of the left image along the corresponding row of the right image. In fact, we perform template matching rather than matching just the pixel brightness. We've used normalized cross correlation for template matching.