Computer Vision Note

Sagar Ojha

September 28, 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

1.2.4 SIFT

The motivation for scale-invariance feature transform (SIFT) algorithms is tied to detecting the same *feature* across multiple images. SIFT is an algorithm to detect as well as describe local features in an image. A "feature" or an interest point has, loosely speaking, a position and size associated with it; SIFT obtains the position and the scale of the feature. In a way, the term "feature" is a region that SIFT detects; this is a circular explanation, but that is the best way I could define it as of now. But do note that a "feature" has some characteristics; these characteristics, however, are not very hard and fast. Here's the reference for the explanation of SIFT feature.

Check this out to see how SIFT works for both 1D and 2D images. To summarize, first the image is convolved with the Laplacian of the Gaussian (LoG); LoG is the second derivative of the Gaussian kernel. After convolution, one needs to normalize the output by multiplying the Laplacian convolution with σ^2 where σ is the standard deviation of the Gaussian used to construct Laplacian. In order to make the detection scale-invariant, the process is repeated for many different LoG kernels, i.e. to say, the process is repeated for different σ. Then, the σ at which the σ normalized Laplacian filter results in the extreme value is selected as the *characteristic scale* for the feature. The *characteristic scale* $\propto$ *feature size*. This proportionality means the feature is detected at various scales. In other words, whether the image is zoomed in or out, the feature can be located and the size of the feature can also be computed for both the images. The pseudocode is given below:

- For an image $I(x, y)$, convolve it with normalized Laplacian of Gaussian (NLoG); $NLoG = \sigma^2\nabla^2 n_\sigma$
  Convolution is given as $\sigma^2\nabla^2 n_\sigma * I(x, y)$
- Repeat the above step for various σ
- Find the local maximum $(x^*, y^*, \sigma^* = \max_{(x,y,\sigma)} |\sigma^2\nabla^2 n_\sigma * I(x, y)|$
- The minimization problem above gives us the locations and scales of the SIFT features; yes, it may detect multiple SIFT features.

The actual implementation of SIFT detection utilizes various tricks rather than the straightforward steps that we have discussed so far. These tricks make the implementation efficient without losing robustness. The trick is to approximate the NLoG function as the difference of Gaussian (DoG) where $DoG = n_{s\sigma} - n_\sigma$ as $DoG \approx (s-1)NLoG$. Check this video for the details about DoG.

The SIFT feature also has a principal orientation associated with it. The direction is obtained using the gradient of the feature. The direction of the gradient which is dominant is the principal orientation. Create a discretized histogram of the gradient directions and then select the principal direction as the direction that gets the most votes.

So far we can detect as well as define the position, scale, and the rotation of the SIFT feature. The data can be used to undo rotation and scaling effects when necessary. There can be many SIFT features in an image. In order to match the SIFT feature between images, we needs to make a distinction between the features. Note that the position, scale and rotation are not enough information to make distinctions between the SIFT feature between the images. What we need is a *descriptor* of the feature. The *descriptor* is a vector that is constructed from the histogram of the gradient directions; the histograms are stored in a specific order in the vector. Note that the *descrpitor* vector will belong to a very high dimensional vector space depending on the number of the histogram bins. In order to compare the SIFT feature, we compare the descriptors of the features. One way of comparing descriptors is to find the L_2 distance between the descriptor vectors; the perfect the match, the closer the vectors will be. Other methods that measure different metrics also exist; normalized correlation and intersection are some methods.

SIFT is a robust feature detection method for planar objects detection but fails for the 3D object detection. SIFT relies on local appearance of the feature in the image; so, if the image of the same scene from 2 different viewpoints are taken, then the method fails. Likewise, SIFT fails to have a unique descriptor for repeating patterns. So, SIFT will do very poorly for feature matching. Nevertheless, SIFT is very popular for finding correspondences in the images.

1.3 Image Stitching

1.3.1 RANSAC

Random sample concensus (RANSAC) is a method to estimate the mathematical model from the dataset with a high number of outliers. Least-squared method fails to estimate the model when there are many outliers as least-squared method aims to get the model that fits all the dataset, both the inliers and the outliers. In RANSAC, we come up with the model and count the number of dataset that are within a certain threshold from the model prediction. The model with the largest number of inliers is selected by RANSAC.

The *homography* matrix represents the transformation between 2 planar projections where the projections are through the same pinhole. One can estimate the *homography* matrix using a minimum of 4 corresponding points between 2 images as explained here. SIFT can be utilized to find the corresponding points. RANSAC can be utilized to come up with a better estimate of the homography. Below is the outline for the process.

- Before performin RANSAC, it is extremely important to have only good SIFT matches
- Estimate the *homography* using a minimum of 4 corresponding points selected at random; if more points are used, then least squared solution for the estimate should be taken
- Using the estimated model of the *homography*, compute where the SIFT features of one image land on the other image
- Compute the error between the actual positions of the SIFT features in the second image and the predictions made by the estimated model
- If the error is within the specified threshold, then the position of the feature is considered to be an inlier
- Estimate another *homography* and repeat the above processess
- Pick the model with the most number of inliers
- The inliers could be used as well to compute the final *homography* using least-squared method

1.3.2 Warping & Stitching

Warping refers to mapping the image onto another plane. Using the *homography*, warp the image onto the reference plane or image and stitch the images. In practice, one first needs to create a canvas to stitch both images. After that, using backwards mapping, fill up the canvas. In other words, for each pixel in the canvas, find the coordinates of the pixels in unwarped image using the inverse of homography and fill up the canvas. For the reference image, only the translation of the origin has to be account for while filling up the canvas. Blending could be utilized to get rid of any seam produced while stitching.

1.4 Hough Transform

1.4.1 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.

1.4.2 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.

1.5 Stereo Vision

1.5.1 Camera Model

The perspective transformation is given as

$$u = f_x\frac{x_c}{z_c} + o_x, \quad v = f_y\frac{y_c}{z_c} + o_y \tag{1.10}$$

where (u, v) is the pixel coordinates (top-left of the image is the origin). $f_x = m_x f, f_y = m_y f$ where m_x, m_y are the pixel densities measured in *pixels/mm* in x, y directions, respectively, and f is the focal length of the camera. (x_c, y_c, z_c) is the coordinates of the particle expressed in camera coordinate frame $\{c\}$. (o_x, o_y) is the coordinate of the *principle point* which is point that optical axis pierces the image plane. Note that f_x, f_y, o_x, o_y are the *intrinsic* parameters of the camera.

The nonlinear transformation expressed in Eq. 1.10 can be written as a iinear transformation using homogeneous coordinates as

$$\begin{bmatrix}u \\ v \\ 1\end{bmatrix} \equiv \begin{bmatrix}\tilde{u} \\ \tilde{v} \\ \tilde{w}\end{bmatrix} \equiv \begin{bmatrix}z_c u \\ z_c v \\ z_c\end{bmatrix} = \begin{bmatrix}f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0\end{bmatrix} \begin{bmatrix}x_c \\ y_c \\ z_c \\ 1\end{bmatrix}. \tag{1.11}$$

After considering the following transformation

$$\begin{bmatrix}x_c \\ y_c \\ z_c \\ 1\end{bmatrix} = \begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1\end{bmatrix} \begin{bmatrix}x_w \\ y_w \\ z_w \\ 1\end{bmatrix} \tag{1.12}$$

where (x_w, y_w, z_w) are the coordinates of a point in world coordinate frame $\{w\}$ and (x_c, y_c, z_c) are the coordinates of the point in camera frame $\{c\}$, the "full" projective transformation, i.e., from world to image plane coordinate frame is

$$\begin{bmatrix}\tilde{u} \\ \tilde{v} \\ \tilde{w}\end{bmatrix} = \begin{bmatrix}p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44}\end{bmatrix} \begin{bmatrix}x_w \\ y_w \\ z_w \\ 1\end{bmatrix}. \tag{1.13}$$

1.5.2 Calibration

Camera calibration refers to finding the projection matrix in Eq. 1.13. The projection matrices P and kP produce the same homogeneous pixel coordinates, hence, we say that the projection is defined only up to a scale. One can set the scale arbitrarily by either setting $p_{34} = 1$ or setting $||p||^2 = 1$. This implies that there are 11 unknown parameters in the projection matrix. Hence, a minimum of 6 points are necessary to solve for the parameters. Unlike homography, find the points in the image plane and obtain the world coordinates of the points. A simple checkered pattern can be utilized for the calibration. Also, take multiple images and obtain the perspective transformation matrix. Intrinsic paramters are not affected by the changing views.

The projection matrix can be decoupled into *intrinsic* and *extrinsic* matrices. In order to obtain the *intrinsic* paramters as well as the rotation matrix, one can use "QR" factorization method as

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{1.14}$$

and the translation vector is obtained by solving

$$\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}. \tag{1.15}$$

1.5.3 Simple Stereo

Simple stereo or horizontal stereo could be utilized to estimate depth using 2 images taken from each camera in the stereo setup. The 3d coordinates of the point expressed in camera frame are obtained as

$$x = \frac{b(u_l - o_x)}{u_l - u_r}, y = \frac{bf_x(v_l - o_y)}{f_y(u_l - u_r)}, z = \frac{bf_x}{u_l - u_r} \tag{1.16}$$

where b is the horizontal baseline, displacement between the camera, (u_l, v_l) are the coordinates of the point in the left image and (u_r, v_r) are the coordinates of the point in the right image.

1.5.4 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.

1.5.5 Uncalibrated Stereo

We assume that the intrinsic parameters are known and the extrinsics are the one we need to work for. The intrinsic parameters are often available as meta-tags in the image file.