Appendix- 1

Computer Vision Techniques Using OpenCV

1. Introduction

In this report, I've outlined the various computer vision techniques that I've implemented. Link to the Github repository: https://github.com/shwetha-krishnamurthy/opencv

2. Image Processing for Computer Vision

Digital images can be two-dimensional arrays (grayscale, binary) with integral values in the range of 0-255, or three-dimensional(RGB, HSV, etc) with the third dimension containing the information about the channels (3 channels for RGB, HSV, etc). We need to process the image before it can be used to derive conclusions. Image preprocessing includes the following techniques:

- Image preprocessing
- Image enhancement
- Image segmentation
- Feature extraction
- Image classification

Image Preprocessing and Enhancement

A number of preprocessing functions are in-built in the OpenCV library.

1. *Converting an image from RGB to Grayscale:*
   An RGB image is a three-dimensional matrix with containing integral values in the range of 0-255.
   Sometimes, it is easier to work with grayscale images with just one channel, containing integral values in the range 0-255. Hence, in order to convert an RGB image to grayscale image, we take weighted average of the RGB values of the original image to generate the grayscale image.

2. *Converting to binary image:*

   The simplest segmentation method.

   Application example: Separate out regions of an image corresponding to objects which we want to analyze. This separation is based on the variation of intensity between the object pixels and the background pixels.

   To differentiate the pixels we are interested in from the rest (which will eventually be rejected), we perform a comparison of each pixel intensity value with respect to a threshold (determined according to the problem to solve).

   Once we have separated properly the important pixels, we can set them with a determined value to identify them (i.e. we can assign them a value of 0 (black), 255 (white) or any value that suits your needs).

3. *Converting to HSV:*

   An HSV (Hue, Saturation, Value) image is more useful than RGB/grayscale images when it comes to marker-based computer vision (used color detection), shadow removal, etc.

4. *Smoothing of an image:*

   Diverse linear filters are available to smooth images using OpenCV functions such as: blur, GaussianBlur, medianBlur, bilateralFilter.

   To perform a smoothing operation we will apply a *filter* to our image. The most common type of filters are *linear*, in which an output pixel's value (i.e. $g(i,j)$) is determined as a weighted sum of input pixel values (i.e. $f(i+k,j+l)$) :

   $$g(i,j) = \sum_{k,l} f(i+k,j+l)h(k,l)$$

   $h(k,l)$ is called the *kernel*, which is nothing more than the coefficients of the filter.

   It helps to visualize a *filter* as a window of coefficients sliding across the image.

5. *Image Denoising:*

   Noise is generally considered to be a random variable with zero mean.

   We need a set of similar images to average out the noise. So we take a pixel, take

small window around it, search for similar windows in the image, average all the windows and replace the pixel with the result we got. This method is Non-Local Means Denoising. It takes more time compared to blurring techniques we saw earlier, but its result is very good. More details and online demo can be found at first link in additional resources.
For color images, image is converted to CIELAB colorspace and then it separately denoise L and AB components.

6. *Improving Contrast of an image using Histogram Equalization:*
   Consider an image whose pixel values are confined to some specific range of values only. For eg, brighter image will have all pixels confined to high values. But a good image will have pixels from all regions of the image. So you need to stretch this histogram to either ends (as given in below image, from wikipedia) and that is what Histogram Equalization does (in simple words). This normally improves the contrast of the image.

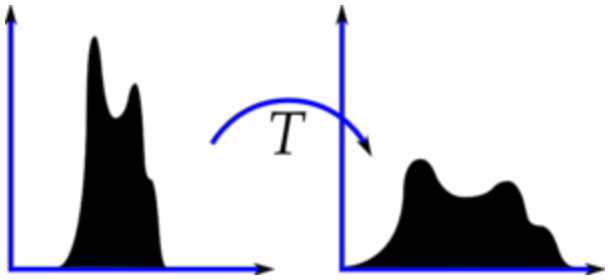

Image Segmentation

Following are the techniques used for image segmentation:

1. *Edge Detection Techniques:*
   In an edge, the pixel intensity changes in a notorious way.

   A. Naive Edge Detection: This technique takes an n x n kernel around a said pixel, and calculates the difference between the minimum and maximum intensity values (grayscale image). If the difference is greater than a given threshold, then the pixel is regarded as an edge pixel, otherwise as a background pixel.

B. Sobel Edge Detection: A method to detect edges in an image can be performed by locating pixel locations where the gradient is higher than its neighbors (or to generalize, higher than a threshold).

Assuming that the image to be operated is I:

We calculate two derivatives:

1. **Horizontal changes**: This is computed by convolving I with a kernel $\mathrm{G_x}$ with odd size. For example for a kernel size of 3, $\mathrm{G_x}$ would be computed as:

$$\mathrm{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathrm{I}$$

2. **Vertical changes**: This is computed by convolving I with a kernel $\mathrm{G_y}$ with odd size. For example for a kernel size of 3, $\mathrm{G_y}$ would be computed as:

$$\mathrm{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathrm{I}$$

At each point of the image we calculate an approximation of the *gradient* in that point by combining both results above:

$$\mathrm{G} = \sqrt{\mathrm{G_x^2} + \mathrm{G_y^2}}$$

Although sometimes the following simpler equation is used:

$$\mathrm{G} = |\mathrm{G_x}| + |\mathrm{G_y}|$$

C. Prewitt Edge Detection: Prewitt operator also works same as Sobel operator. But the kernel data is little different

$$h_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad h_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Other operations are same as Sobel.

D. Canny Edge Detection: Also known to many as the optimal detector, Canny algorithm aims to satisfy three main criteria:

Low error rate: Meaning a good detection of only existent edges.

Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.

Minimal response: Only one detector response per edge.

Filter out any noise. The Gaussian filter is used for this purpose. An example of a Gaussian kernel of $size = 5$ that might be used is shown below:

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:

1. Apply a pair of convolution masks (in x and y directions:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

2. Find the gradient strength and direction with:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan(\frac{G_y}{G_x})$$

3. The direction is rounded to one of four possible angles (namely 0, 45, 90 or 135)

Non-maximum suppression is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.

Hysteresis: The final step. Canny does use two thresholds (upper and lower):

1. If a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge
2. If a pixel gradient value is below the *lower* threshold, then it is rejected.
3. If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.
   Canny recommended a *upper*:*lower* ratio between 2:1 and 3:1.

2. *Shape Detection Techniques:*

   Using contours with OpenCV, we can get a sequence of points of vertices of each white patch (White patches are considered as polygons). As example, you will get 3 points (vertices) for a triangle, and 4 points for quadrilaterals. So, we can identify any polygon by the number of vertices of that polygon. We can even identify features of polygons such as convexity, concavity, equilateral and etc by calculating and comparing distances between vertices.

3. *Blob Detection:*

   SimpleBlobDetector, as the name implies, is based on a rather simple algorithm described below. The algorithm is controlled by parameters and has the following steps. Parameters: color, size, shape.

   1. **Thresholding :** Convert the source images to several binary images by thresholding the source image with thresholds starting at **minThreshold**. These thresholds are incremented by **thresholdStep** until **maxThreshold**. So the first threshold is **minThreshold,** the second is **minThreshold** + **thresholdStep,** the third is **minThreshold** + **2 x thresholdStep**, and so on.
   2. **Grouping :** In each binary image, connected white pixels are grouped together. Let's call these binary blobs.
   3. **Merging** : The centers of the binary blobs in the binary images are computed, and blobs located closer than **minDistBetweenBlobs** are merged.

4. **Center & Radius Calculation :** The centers and radii of the new merged blobs are computed and returned.

   Apart from this, blob detection can also be performed by using Breadth-First Search.

4. *Ridge Detection:* Sometimes, we need to detect ridges instead of edges, for eg. leaf veins. Therefore, the idea behind ridge detection is that the pixel values can be (locally) approximated by a 2nd order polynomial. The second order derivative matrix is called the Hessian matrix. It describes the 2nd order structure we're interested in.

Feature Extraction and Image Classification

Features or keypoints can be extracted from an image using SIFT (Scale Invariant Feature Transform) and SURF (Speeded Up Robust Feature) extractors. These keypoints are be used for various purposes, such as, locating an object in the image of a scene, classification of images, etc.

Image Classification calls for various Machine Learning techniques, such as an SVM classifier, a Random Forest classifier, or various Deep Learning techniques.

References

1. OpenCV Documentation: http://docs.opencv.org/2.4/doc/tutorials/tutorials.html
2. Robotics - Control, Sensing, Vision, and Intelligence: *K.S. Fu, R.C. Gonzalez, C.S.G. Lee*