**REAL-TIME EDGE DETECTION USING CANNY EDGE DETECTION METHOD**

Abstract :Interpretation of image contents is one of the main objectives in computer vision. Edge is a basic and important feature of an image. Image is a combination of edges. Detecting edges is one of the most important aspects in image segmentation. Edge detection is a vital step as it is a process of identifying and locating sharp discontinuities in an image.

Edge detection is a technique in which the points where image brightness changes sharply or formally are identified. These points are organized under line segments called edges. Edge detection also aims to classify and place discontinuities in an image. Noise and image both have high frequency, hence edge detection becomes difficult. The main objective of studying various edge detection techniques and analyzing their performance is due to problems such as fake edge detection, noisy images, missing edges etc.

EDGE DETECTION TECHNIQUES

Edge:The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. The edges extracted from a two-dimensional image of a three-dimensional scene can be classified as either viewpoint dependent or viewpoint independent. A viewpoint independent edge typically reflects inherent properties of the three-dimensional objects, such as surface markings and surface shape. A viewpoint dependent edge may change as the viewpoint changes, and typically reflects the geometry of the scene, such as objects occluding one another. A typical edge might for instance be the border between a block of red color and a block of yellow. In contrast a line can be a small number of pixels of a different color on an otherwise unchanging background. For a line, there may therefore usually be one edge on each side of the line.

The objective of the program given is to perform edge detection of images in real-time. The popular canny edge detection algorithm is used to detect a wide range of edges in images. OpenCV has in-built function cv2.Canny() which takes our input image as first argument and its aperture size(min value and max value) as last two arguments.

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in

* It is a multi-stage algorithm and we will go through each stages.
* **Noise Reduction**

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

* **Finding Intensity Gradient of the Image**

Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction ( Gx) and vertical direction ( Gy). From these two images, we can find edge gradient and direction for each pixel as follows:

Edge\_Gradient(G)=G2x+G2y−−−−−−−√Angle(θ)=tan−1(GyGx)

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

* **Non-maximum Suppression**

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:



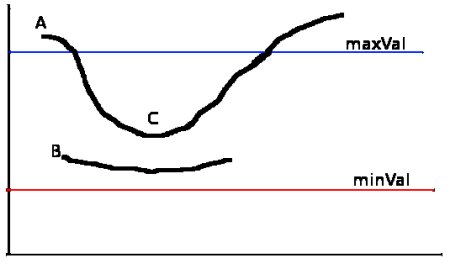
**Fig (a)**

Point A is on the edge ( in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed ( put to zero).

In short, the result you get is a binary image with "thin edges".

* **Hysteresis Thresholding**

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:



**Fig (b)**

The edge A is above the maxVal, so considered as "sure-edge". Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result.This stage also removes small pixels noises on the assumption that edges are long lines.So what we finally get is strong edges in the image.

Canny Edge Detection: Canny edge detector is one of the most commonly used image processing tools. It detects edges in a very robust manner. Unlike Roberts Cross and Sobel, the canny operation is not very susceptible to noise. It takes less time than Roberts cross. It is one of the most important methods to find the edges by separating noise from input image. The algorithm is adaptable to various environments. It is a better method because it extracts the features in an image without disturbing its features. There are certain criteria to improve current methods of edge detection. The first and most obvious is low error rate. It is important that edges occurring in images should not be missed. The second criterion is that the edge points be well localized i.e. the distance between the edge pixels as found by the detector and the actual edge should be minimum. A third criterion is to have only one response to a single edge.

SOURCE CODE

# OpenCV program to perform Edge detection in real time

# import libraries of python OpenCV

# where its functionality resides

import cv2

# np is an alias pointing to numpy library

import numpy as np

# capture frames from a camera

cap = cv2.VideoCapture(0)

# loop runs if capturing has been initialized

while(1):

# reads frames from a camera

ret, frame = cap.read()

# converting BGR to HSV

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

# define range of red color in HSV

lower\_red = np.array([30,150,50])

upper\_red = np.array([255,255,180])

# create a red HSV colour boundary and

# threshold HSV image

mask = cv2.inRange(hsv, lower\_red, upper\_red)

# Bitwise-AND mask and original image

res = cv2.bitwise\_and(frame,frame, mask= mask)

# Display an original image

cv2.imshow('Original',frame)

# finds edges in the input image image and

# marks them in the output map edges

edges = cv2.Canny(frame,100,200)

# Display edges in a frame

cv2.imshow('Edges',edges)

# Wait for Esc key to stop

k = cv2.waitKey(5) & 0xFF

if k == 27:

break

# Close the window

cap.release()

# De-allocate any associated memory usage

cv2.destroyAllWindows()

Output:

