Snow Level Detection Using Image Processing.

**Algorithm Development.**

# Phase 1: Algorithm Testing for simple Images.

**First, let us try and develop an Algorithm for simple test cases and see its efficiency.**

Problem: To detect the lines in an image. To give changes in line count, line space and line length.

Difficulties:

* Minimal processing power consumption. So, efficient Algorithm is required to reduce processing time.
* Choosing the most suitable operator in terms of efficiency and at the same time noise reduction is required.
* Most difficult challenge is to implement it in the Development platform.

**Step 1:** Importing all the necessary modules. Importing Data- images (bit map).

**Step 2:** Converting the image into a grayscale image and assigning it to a variable.

**Step 3:** Apply a filter using kernel (3x3 or 5x5) for smoothening the images. Like a gaussian filter/gaussian blur.

**Step 4:** Calculate the Gradient magnitude and direction for every pixel (or simply for every block of the matrix of image) or by using a convolution kernel as a nearest approximation.

**Step 5:** Noise cancellation/suppression by using the maxima condition. For the actual edges the gradient value will be local maxima but not for small noises.

**Step 6:** Apply Thresholding to convert the gray scale image into a binary image with the edges being assigned white (255) and rest as black (0).

**Steps 3** **to 6 are done using a single in-built canny function which uses Gaussian filter and Sobel operator (for kernel) like Sobel function but additionally helps us in thinning the edges and performs Hysteresis Thresholding (which is very much efficient). For reference- (**<https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html>**).**

**Step 7:** Use dilation or erosion techniques to reduce noise or thicken edges if required by using dilate and erode inbuilt functions.

**Step 8:** Store the dimensions of the image in variables.

**Step 9:** To detect any lines that are present in the image we can use Hough Transformation method. Store the output in a variable.

This method uses the representation of a line equation by . All the possible values of are plotted in an array for every possible point (pixel) in the image. So, for every point satisfying the values the count in the corresponding box of the array increases by 1- finally representing the total no. of points on the corresponding line equation.

By passing in a threshold value of count in the function, we get all the values that passed the limit. Indicating that there is a high chance that an edge with that line equation exists. All the lengths are calculated in no. of pixels unit and all the angles are calculated using radians. For reference- (<https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html>).

**Step 10:** HoughLines function gives an output of an array of all plausible values. Using this we can get the no. of lines in the image.

**Step 11:** Create 2 lists variables to store the lengths (initially zeroes) and line spacing of the lines in the image.

**Step 12:** Loop over all the pixels in the binary image stored and check for the values that each of the pixel satisfies. For this check if the corresponding x and y values satisfy the line equation of those values. We need to round off the cosine and sine values as the ρ values are all integers and measured in number of pixels. For every test pass, increment the corresponding value in the list. Finally, we will get the list with no. of pixels on the edge which is nothing but the length of the edge in terms of no. of pixels unit.

**Step 13:** Now to find the line spacing. Just iterate over the array given by HoughLines and calculate the difference between the ρ values of the corresponding lines whose theta values are the same, Indicating, those are parallel lines. We need to check for every possible line combination. So, we need to use nested ‘for’ loops in both the **steps 12 and 13**.

**Step 14:** Now, we have all the values we needed. We can draw the lines and mark them on the image if we can get the end points easily. We can see the binary image obtained by using imShow function and save the file by using imWrite function.