

A REPORT  
ON  
**SNOW LEVEL CHANGE DETECTION USING IMAGE PROCESSING AND MACHINE LEARNING**

BY

Sudheer Arja

2018B3AA0878H

Somya Sawlani

2018AAPS0252G

AT

Indira Gandhi Centre for Atomic Research (IGCAR)- Kalpakkam

A Practice School-I station of

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

**(June 2020)**

**A REPORT**  
**ON**  
**SNOW LEVEL CHANGE DETECTION USING IMAGE PROCESSING AND MACHINE LEARNING**

**BY**

Sudheer Arja	2018B3AA0878H	M.Sc. Economics B.E. Electronics and Communication
Somya Sawlani	2018AAPS0252G	B.E. Electronics and Communication

Prepared in partial fulfillment of the  
Practice School-I Course Nos.  
BITS C221/BITS C231/BITS C241

**AT**

Indira Gandhi Centre for Atomic Research (IGCAR)- Kalpakkam

A Practice School-I station of

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

**(June 2020)**

### **Acknowledgments:**

I would first like to thank the University- Birla Institute of Technology and sciences Pilani for providing me such a wonderful opportunity to work for the Practice School-I Under one of India's Premier Nuclear Research Centers Indira Gandhi Centre for Atomic Research (IGCAR). I feel it is an honor to work under the mentorship of such important research facility in India. I also like to thank my Instructors Prof. Sanket Goel and Prof. Sandip S. Deshmukh for providing support during these tough times. I like to thank my Parents for Sustaining and tolerating me. In the end, I express my sincere Gratitude to my mentor Ms. Vinita Daiya from IGCAR for providing her valuable insight and guidance which helped me very much working with the Project.

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI  
(RAJASTHAN)  
Practice School Division**

**Station:** Indira Gandhi Centre for Atomic Research (IGCAR)- Kalpakkam  
**Centre:** Kalpakkam, Tamil Nadu

**Duration:** 18 May 2020 to 27 June 2020

**Date of Start:** 18 May 2020

**Date of Submission:** 25 June 2020

**Title of the Project:** SNOW LEVEL CHANGE DETECTION USING IMAGE PROCESSING AND MACHINE LEARNING

<b>Name:</b>	<b>ID.No.:</b>	<b>Discipline:</b>
Sudheer Arja	2018B3AA0878H	M.Sc. Economics B.E. Electronics and Communication
Somya Sawlani	2018AAPS0252G	B.E. Electronics and Communication

**Expert:** Ms. Vinita Daiya- Scientific Officer (IGCAR Kalpakkam)

**PS Faculty:** Prof. Sanket Goel and Prof. Sandip S. Deshmukh

**Key Words:** Image Processing, Edge Detection, Image Segmentation, Binary Image, Canny Edge Detector, Hough Line Transform, Convolution, Cyclomatic Complexity.

**Project Areas:** Image Processing, OpenCV with Python, Level Detection.

**Abstract:** The Project is based on Algorithm development for Snow level detection using various Image Processing techniques on the Images collected using low power cameras. With the developed Algorithm an optimized code is written in Python environment for the detection of Snow level using the field Images. The Snow level calculated is a parameter used for Avalanche detection using WSN.

Signature(s) of Student(s)

*sudheer*  
*somya*

Date: 25 June

Signature of PS Faculty

Prof. Sanket Goel

Prof. Sandip S. Deshmukh

Date: 25 June

## Table of Contents

Acknowledgments .....	III
Abstract .....	4
Introduction .....	6
Literature Survey .....	8
Algorithm Development .....	9
Phase-I .....	13
Phase-II .....	15
Limitations.....	19
Conclusions .....	20
Appendix- A.....	21
Appendix- B.....	23
Appendix- C.....	24
References .....	26
Glossary .....	27

## Introduction

Our Project is Snow level Change detection using Image Processing and Machine Learning. This Project is for the WSN-Avalanche Project which aims at using the Wireless Sensor Network Technology to detect any possible avalanches at the site and send a warning about the same. Wireless sensor network (WSN) has a crucial role in computer networking for searching area and also in information assortment. Now a days WSN is being used in several areas, including environment monitoring and data storage. Because of the rapid urbanization, the range of applications of WSN has been increasing enormously. WSN has numerous sensors which are communicated via sending data from one sensor to a different sensor during a wide area via packets. WSN is ubiquitously used in various fields such as military surveillance, the Health care industry, and other industries. The rise of the 3<sup>rd</sup> millennium industries has made it fascinating to research in WSN. The actual implementation of WSN was started in 1950 and therefore the US military used this technology. The first developed WSN was named the “Sound Surveillance System” (SOSUS). It was to detect the threat of underwater submarines. Acoustic sensors (a sort of sensor won't check the amplitude of the waves) are used for (SOSUS) and this technology is additionally being practiced nowadays. Today, in every part of life, a sensor is used to detect sound, detect any kind of vibrations, check the water level, check the snow level, detect the temperature level in smart houses and smart phones but most importantly, we use some kind of sensors in human body. The WSN consists of nodes, and through these nodes, sensors communicate with one another. The hierarchy of those is sort of a star where every node connects to every other or sort of a mesh. Each node collects data from this sensor and communicates it to the other nodes in the network. In general, a sensor node consists of a microcontroller, transceiver, external memory, power source and one or more sensors.

It is documented that for meteorological centres the detection of inclemency conditions is crucial, especially with demand for air, sea and ground traffic management. The detection and therefore the characterization of weather usually involves dedicated sensors, like a visibility meter, a dendrometer or a radar, etc. However, the value of those sensors sets a limit to their extensive deployment. The massive deployment of video surveillance cameras for security and safety reasons is an opportunity for using them to monitor weather conditions.

Snow avalanche occurs whenever newly generated snow accumulated on an important snow layer and doesn't properly adhere to the prevailing layer thereby leading to avalanche breakdown. It then results in various kind of vibration caused due to skating, firing etc. Warning systems can detect avalanches which develop slowly, like ice avalanches caused by icefalls from glaciers. High-resolution Cameras, or motion sensors can monitor instable areas over an extended term, lasting from days to years. Experts interpret the recorded data and are ready to recognize upcoming ruptures so as to initiate appropriate measures. Such systems can recognize events several days in advance. In the past few years, to detect change in snow/water level image processing technology has been applied. Years Essentially, image processing can replace

the human eye with implementation of the visualisation decision and avoid expensive human resources. Compared to other non-contact methods, such as the microwave radar, image-based methods may cost fewer human resources and money, but still have advantages of automation, convenience, and efficiency. Image Processing may be a method to perform some operations on a picture, so as to urge an enhanced image or to extract some useful information from it. It is a kind of signal processing during which input is a picture and output could also be image or characteristics/features related to that image. Image processing is among the rapidly growing technologies in recent times.

Many Parameters are being observed for the Avalanche detection such as Temperatures, Pressure and Snow Level etc. at the nodes. For monitoring the snow build, Low Power cameras have been installed at the site. So, our Project aims at extracting the snow level from the reference pole mounted in the snow-covered areas. For calculating the snow level, we need to perform various Image Processing techniques on the Images that are provided to us by the low power cameras. As the embedded systems are not connected with any AC supply so we need to develop the Algorithm which uses minimal processing power. The key is to recognise the trade-off between complexity and accuracy. So, we need to perform complexity analysis on the code for finalization as well. For the Algorithm developed we need to implement the same in OpenCV Python environment to extract the snow level cover from multiple images. We have roughly divided the project into three phases. Those are:

- Developing Algorithm for edge Detection and edge length calculation.
- Developing the code for calculating the edge lengths using simpler images with plain background.
- Developing the code for calculating the Snow Level Cover in field images.

So, we have developed an Algorithm for edge detection and calculating the edge lengths. We have developed the code for detecting edges and edge lengths for simpler images and then we have modified the developed code accordingly to acquire the snow level cover at the region and also implemented it on field images for accuracy testing. In the end we have performed complexity analysis and optimization tests on the developed code.

## Literature Survey

To get some Knowledge regarding the Project we have done a literature survey initially. The Literature Survey was based on change or level detection using Image Processing. As a part of the Literature Survey we have gone through many research papers in the web. We managed to Understand many technical terms and came across many popular techniques that are involved in Image Processing. Change detection techniques and Algorithms are not new to us, many people developed fine techniques which helped us in preparing a tentative work plan for the Project. Image Processing is being used from observing environmental characteristics to Laboratory purposes. Measuring water level under check dams generally uses image processing [1]. Based on the resources we divided the process into three steps:

- Image Filtering
- Image Segmentation
- Image Analysis and Calibration.

For any Image Processing technique for Level or change detection these three steps are covered in one or other way. In the Literature Survey we came across the first two steps mainly. In every paper we came across, the first step primarily is to soften the image by applying some filter like **Gaussian filter** and then converting the Image into a monochrome Image i.e. **Grey scale image** for easier calculations. For Better accuracy **HSV** colour filter is used.

To Elaborate, Images are two types: a bitmap or a Vector graphics (SVG). The most common image format is a bitmap which is nothing but a 2D Array with each block (called a pixel) having a colour. To be precise the pixel value is actually an intensity value. For a grey scale image, the pixel contains the intensity of the colour. Where in an 8-bit image 0 indicates complete black and 255 indicates complete white, in between values gives the shades of grey colour. There by total number of possible values for a pixel are 256 which is  $2^8$ . In an **RGB Image** (colour Image) a pixel contains the three intensity values corresponding to red, green and blue. There by total possible values are  $2^{24}$  [2].

To work with RGB Image directly we must consider all the three intensities which can become complex during calculations. So, the colour to grey scale filter is generally applied before starting to work on the image. Then the next part is Image segmentation. This was the important part of the survey. Most of the information we gathered talked about this step. In general, this step is considered the crucial for Image Processing. This constitutes various methods. But, the most popular Image segmentation techniques includes **Region of Interest (ROI)** [3] and **Thresholding**. Region of interest is simply selecting a part of the image which is in our interest and studying it instead of the whole image. This is done generally to make the analysis simpler. Thresholding is nothing but selecting an intensity level which acts as a barrier. Any intensity value (of pixel) which is less than the barrier is changed to 0 and higher values to 255. As a result, what we get is a **binary Image** from Thresholding.



Instead of directly applying thresholding many consider using **Edge Detection Algorithms** as an effective tool for Image Segmentation. Edge Detection is also considered very much helpful and widely used for Image Processing. There are many Edge Detection Algorithms. **Sobel Edge Detection, Canny Edge Detection, Prewitt Edge Detection** and **Laplacian Edge Detection** are the most popular [4]. All these Edge Detection Algorithms include Thresholding and Filtering as well. But the important part is that using specific operators they calculate the **Gradient values** [5] and directions for the image. They apply Thresholding on the Gradient values. Some Algorithms consider directions, and some do not. So, from these Algorithms we can get the edges more accurately than just normal thresholding by **Otsu's method** (see appendix A) [6].

## Algorithm Development

Based on the Literature survey, we have started to develop the Algorithm for the phase-I. To get a clear understanding of the process, we have studied various Edge Detection Algorithms and Line Detection Algorithms in detail. Initially, we studied the Sobel Algorithm and Canny Algorithm to understand the back end of Edge Detection. Among the Edge Detection Algorithms Sobel is the basic Algorithm.

- **Sobel Edge Detection.**

Sobel first filters the Image using Gaussian Filter. Then, Sobel uses **Kernel Convolution** (see appendix A) by Sobel operator (kernel) to calculate the gradient approximation. Many filtering techniques also uses Kernel Convolution across the image. Sobel uses two convolutions horizontal and vertical. These two gradient values are then used to get the gradient magnitude either by addition or root square. Then based on the approximate gradient magnitudes and directions spatial gradient is obtained. Then thresholding is applied based on the magnitudes of gradient values. Gradient angle is used for the orientation of the edges. In this way Sobel edge detection detects the edges [7].



Figure 1: Image after Sobel edge detection (binary image).



Figure 2: actual colour Image

- **Prewitt Edge Detection.**

Prewitt edge detection is similar to Sobel edge Detection. Horizontal and vertical gradients are also calculated in Prewitt edge detection using Prewitt operator. However, unlike the Sobel operator it does not place any emphasis on the pixels that are closer to the centre of the mask. It is a fast method for edge Detection. It is a good method to estimate the magnitude and direction of an edge.



Figure 3: using Prewitt operator.

- **Canny Edge Detection.**

However, we have selected Canny Edge Detection method for the project. This is because of the higher efficiency in terms of accuracy, processing time than the other edge detection algorithms. Canny uses two additional steps than the Sobel. The first two steps are similar to Sobel. Canny additionally performs **Non-maximum suppression**. That means the edges detected are checked for local maxima along the direction of the gradient. If it doesn't form maxima, then the edge (edge point) is put to zero. In this way noise is cancelled. Then it performs **Hysteresis Thresholding** rather than normal Thresholding. This takes in two Threshold values minimum and a maximum. The points which are greater than maximum are directly considered. Points less than minimum are neglected directly and in between points are considered if they are directly attached to clear edges. [8] [9]

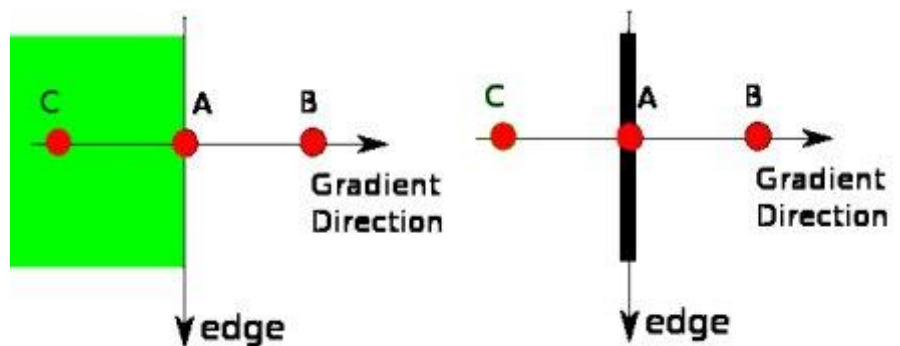


Figure 4: non-maximum suppression in Canny



Figure 5: Using Canny operator.

- **Hough Line Transformation.**

This part can be considered under **Image Analysis**. For Line detection we have used the **Hough line Transformation method**. Hough Transformation for lines basically considers the line equation  $\rho = x \cos \theta + y \sin \theta$ . For Every point it calculates all the possible parameters  $\rho$  and  $\theta$ . It maintains the count of points that satisfy the parameters. It stores the count in a 2D array with rows as the perpendicular distance and columns as the angle. So, the high no of count reflects that the line with those parameters can be actually present. This means that Hough line transform can detect the line even if it is broken or discontinued a bit. By using all the mentioned techniques, we have developed the algorithm which seemed best in the harsh climate of snow-covered regions. We can also see that only noise problems can be encountered using these techniques, pixel resolution is not likely to affect Hough line transform or Canny. This is the reason we selected them for our Algorithm [10].

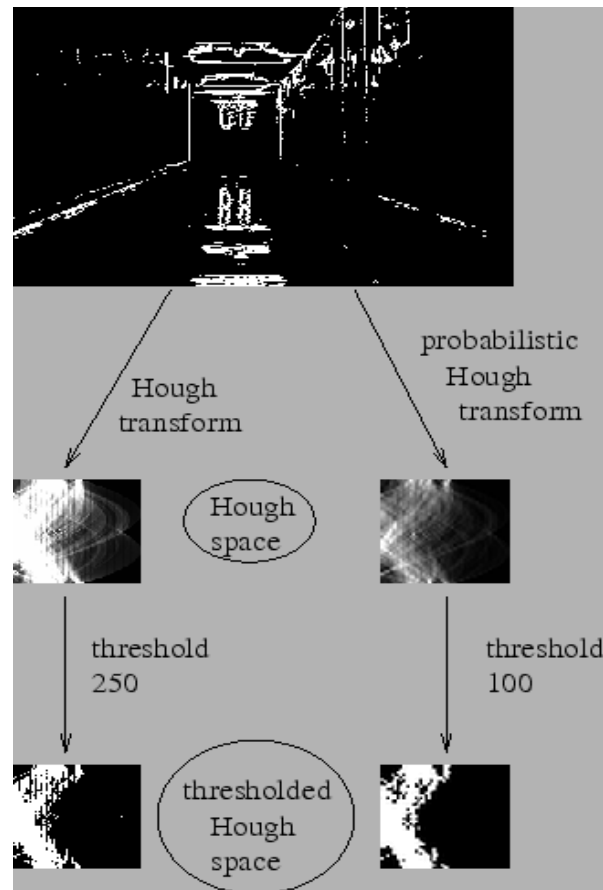
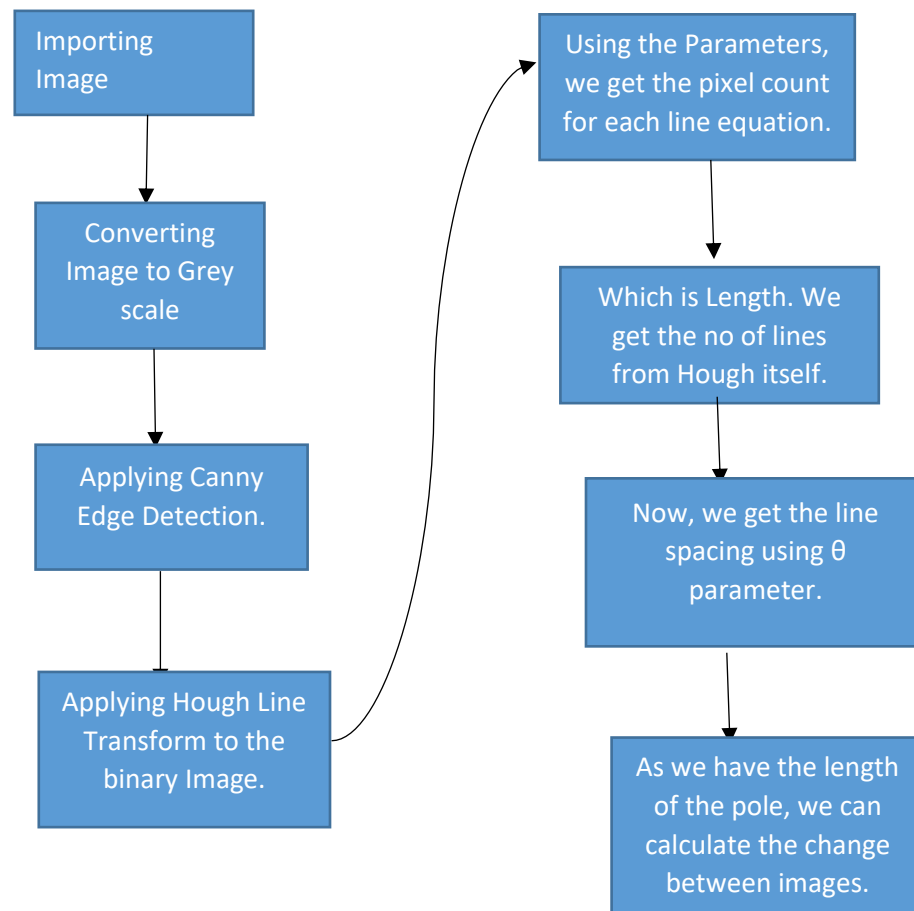


Figure 6: Hough transform

Take a look at the flow chart of the proposed Algorithm for Phase-I. Which we have implemented using OpenCV in python.



We have implemented this algorithm for Phase-I and a similar algorithm is being followed for the Phase-II as well. We modified the algorithm slightly in order to increase accuracy and reduce the processing time for Phase-II.

## Phase-I

We have developed a code to detect the edges and lines in a simple image. The code also calculates the length of lines detected, number of lines in the image and also the spacing between them. We have developed the code in Python 3 environment using OpenCV library. We have developed the Python code using the well-known Jupyter notebook or formerly known as the IPython notebooks computational environment [11] with the Anaconda console [12]. After setting up the required environment, we have started implementing the algorithm for simple images.

We have searched the internet for various inbuilt operators and functions which are useful in the vast OpenCV library. We came across many useful inbuilt functions like **Canny**, **Sobel**, **HoughlinesP** etc. In the Phase-I we have used various edge detection operators like Sobel, Canny and Prewitt and analysed their accuracy and processing times respectively. Based on this we have chosen the Canny inbuilt operator. It shown very high accuracy and similar processing time compared to the other inbuilt operators. For line detection, we have used **Houghlines** and HoughlinesP. These inbuilt functions use the traditional Hough line Transformation. HoughlinesP uses Probabilistic Hough line Transformation, which is similar to the Hough transformation but uses random sets of points instead of the complete image like in case of Houghlines. So, as expected we have observed less processing time in case of HoughlinesP. The accuracy in line detection is clearly more for Houghlines. But still the results from HoughlinesP are considerably accurate. So, we have used HoughlinesP for line detection.

A brief explanation of the code is: first using the imread function we input the required image (as Grayscale). Now we apply the Canny function on the input image from which we get a binary image as output. On the Binary image we apply the HoughlinesP function which gives the detected lines that crossed the Threshold as an array of end points. Using the array, we calculate the no of lines detected and their lengths. Based on the angles made by the lines with the Horizontal, their line Spacing is calculated using nested for loop and conditional statements. Finally, using the imwrite function the edited image is saved. The developed codes are being attached as objects in the appendix.

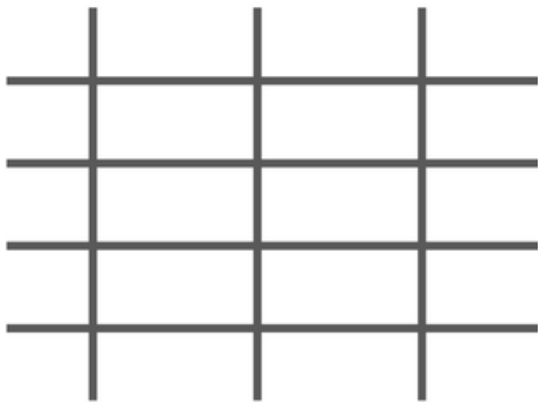


Figure 7: Sample image 2

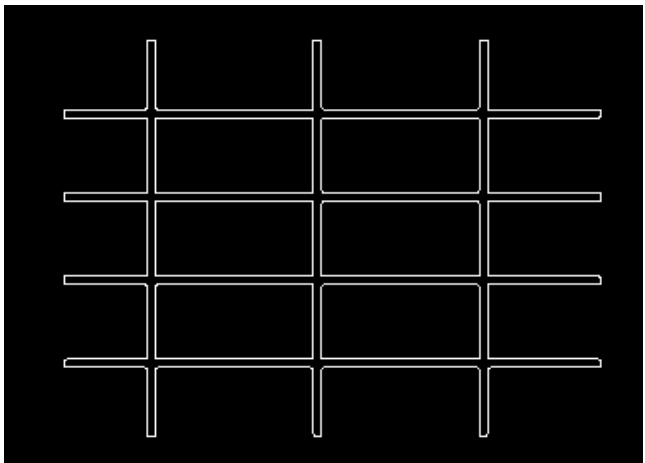


Figure 8: after applying canny

The number of lines, their lengths and Line Spacing are being calculated accurately using the developed code. These are the wall times with using Houghlines and HoughlinesP in the below images.

```
cv2.imwrite('canny image3
```

2.3 ms ± 306 µs per loop

Figure 9: using HoughlinesP runtime

```
#cv2.destroyAllWindows()
```

```
cv2.imwrite('canny image3.
```

91.8 ms ± 4.24 ms per loop

Figure 10: using Houghlines runtime

## Phase-II

After we have completed the Phase-I of the Project, we started testing the developed code on field images provided by our mentor. We received satisfactory results regarding the accuracy by using the code of phase-I on field images. So, we proceeded with the same algorithm and started developing the code for Snow level change using the field images. This process can be briefly divided into three steps.

- Reference Pole Calculations
- Snow level Detection
- Optimization and Complexity analysis

### **Reference Pole Calculations:**

To calculate the Snow level cover at a region using images, we have considered a simple but an efficient way of detecting pole length changes in multiple images. So, first we need to calculate the length of the reference pole before any snow fall. We call this image as the reference image. As a first step we calculated the reference pole length in the reference image. As the reference pole mounted at the site was white. We could not process the field images for edge detection because of the white back-ground due to the snow fall. We have thus used the support pole as the reference for snow level detection.

By using NumPy library in python, we have selected the Region of interest manually for a better analysis. Then by using the line function and imshow in OpenCV we have manually calculated the length of the reference pole (support pole) in the reference image. For Calibration we will require the pixel to metric ratio to convert the calculations in pixel length into meters. We calculated the pixel to metric ratio using the reference pole markings in the image [13].

We have calculated the reference pole calculations manually for better accuracy and these are required to be calculated only once.





*Figure 11: Reference Image*

### **Snow Level Detection:**

We imported the reference calculations for snow level detection using field images. This is the important step and code development part in Phase-II. We have used a similar Algorithm like in Phase-I but we modified the analysis for better accuracy in Snow level detection. First, we have input the image as a colour image and then stored Gray scale image in another variable. The Region of interest is selected from the Gray scale image manually. Like in Phase-I we perform edge detection and line detection on the Gray scale ROI. But additionally, we performed Gaussian blur by using GaussianBlur function in OpenCV for smoothening. Canny operator also consists a smoothening step using Gaussian operator. We have set the Threshold for HoughlinesP at 135. Which means only the lines with at least 135-pixel length will be considered. From the lines detected, we separate the lines which are at least 85 degrees with the horizontal into a variable. Now, we calculate the lengths and angles for the separated lines and store them in lists. From these separated lines we select two lines which are most close to an angle of 90 degrees with horizontal.

So, from these two selected lines we acquire the start point and end point of the reference pole in the field image. We have considered two cases for this:

- (i) If the Selected lines have a difference in lengths less than or equal to 20 pixels.
- (ii) If the selected lines have a difference in lengths greater than 20 pixels.

For Case (i), we consider the mid points of the end points of selected lines as the start point and end point. For Case (ii), we consider the end points of larger line as the start point and the end point. So, now using the start point and end point we calculate the pole length in the field image.



The difference between the initial pole length and pole length in the field image gives us the Snow Level. But in Pixels. So, now we divide the value with the pixel to metric ratio to get the snow level in meters. This step is known as Calibration. We have tested this code on 3 different field images of a node.

In the above Images. The red line joins the start point and end point considered. The green lines indicate the separated lines at least 85 degrees with the horizontal.



*Figure 12: Field image 1 analysis*



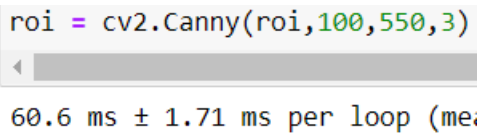
*Figure 13: field image 2 analysis*

We have done this process for avoiding any distortions in line detection by the Probabilistic Hough Transformation. Such distortions like in the field image 2 analysis where the right edge is not detected as a single line are caused due to the pixelation of edges and the intensity fluctuations caused due to the sunlight. Pixelation is tried to be avoided by increasing the resolution for the region of interest as much as possible.

### **Optimization and Complexity Analysis:**

After the Code development, we have tried to optimize the code. We tried using different approaches for Image Segmentation and Image analysis parts. For Image segmentation, as the Canny operator is the more efficient compared to other inbuilt edge detection operators. We have verified the same by testing the accuracy and processing times for Sobel and Prewitt operators. So, we have modified the Canny operator and explicitly written the function by skipping some

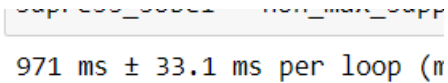
steps which are seemed not necessary. Then we have tested the same for accuracy and processing time for 7 runs (for 10 loops each), we have got the accuracy as considerable but the processing time for the same has exceeded compared to that of the inbuilt function [14] [15].



```
roi = cv2.Canny(roi, 100, 550, 3)
```

60.6 ms ± 1.71 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

Figure 14: inbuilt canny operator runtime



```
canny = canny_manual(img, 100, 550, 3)
```

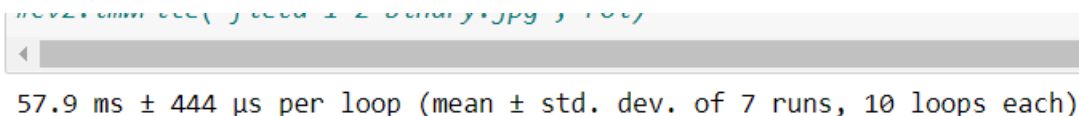
971 ms ± 33.1 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

Figure 15: manual canny operator runtime

The reason for such huge fluctuation is that the OpenCV library is written in C++ but the manual written operator is in Python. Modifying the operator in python is not sufficient. The processing time difference can be achieved using C++ instead of Python for code development. But due to the problem of constrained time limit we have used Python with OpenCV instead of C++ with OpenCV. As the implementation of Image processing in C++ is much complex than in Python.

We have tried other approaches for reaching optimization in Image analysis. We have used the Structural analysis and shape descriptors method to do the image analysis part. For this, we have used findcontours and arcLength functions in OpenCV. By using this approach, we can get the perimeter of the object. Which can then be used for calculating the pole length differences. On performing tests for the processing time, we received the processing time very much similar to that of probabilistic Hough transformation approach. The feasibility for performing the analysis is better in case of probabilistic Hough transformation [16].

Hence, we concluded that the further optimization of the code cannot be possible in this environment with the developed Algorithm.



```
result = find_contours(img, 100, 550, 3)
```

57.9 ms ± 444 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

Figure 16: The wall time for developed code Phase-II using a field image.

We have also performed a complexity analysis on the developed code to get an overview of the complexity of the code. We have used cyclomatic complexity as suggested by our mentor. The cyclomatic complexity gives us the number of linearly independent paths used in the code. For this we have used the Lizard code analyser which is a simple code complexity analyser. The CCN (cyclomatic complexity number) for the developed code has not exceeded the threshold of 15 and neither the threshold of NLOC (number of lines of code) of 1000000. So, the developed code is less complex and feasible to be tested [17] [18].

## Limitations

There are a few limitations that are observed while testing the accuracy of the developed code.

- We need to adjust the threshold value for the line detection by the probabilistic Hough transformation. This helps us in avoiding unwanted lines detected and the noise. But the trade-off for this threshold is that it **limits the maximum snow level** that can be detected.
- We have to provide the **ROI and the threshold values manually**. These can vary with the node (positioning of the camera and the reference pole). ROI and Thresholding selection can affect the accuracy of detection.
- **External factors like Shadows** etc. which are very close to the pole can affect the edge detection process which can deviations in line detection. This results in an underestimation of the Snow level.
- The line detection by probabilistic Hough line transformation can be affected by the resolution of the image which causes **pixelation of edges**. The sunlight can also cause unexpected fluctuations in the intensity of edge points.

## Conclusions

- We have developed an algorithm and code for the detection of snow level using image Processing.
- Various Image processing techniques and algorithms are studied for developing the required Algorithm.
- Various edge detection Algorithms are studied in detail for the project.
- We have decided to use Canny Edge Detection because of its accuracy and precision.
- Canny is used in the Project due to the low processing power usage despite of its computational expense.

Image type/method	Canny CPU usage%	Sobel CPU usage%	Prewitt CPU usage%
JPEG	20	21	22
PNG	19	31	29

- Various Image Analysis techniques are being studied for the project.
- Probabilistic Hough Transformation is used because of its wide range of application and less complexity.
- The developed code in Phase-I has provided high accurate results.
- The developed code in Phase-II has provided considerably accurate results.
- The developed code in Phase-II is tested using 3 field images provided by the mentor. The Snow levels detected are:

Image (JPG format)	Snow Level detected (in cms)	Wall time (in ms)
Field Image 1	63.38	57.9 ± 444μs
Field Image 2	49.22	59.2 ± 284μs
Field image 3	111.17	57.8 ± 572μs

The wall time is generated in a computer with Processor- Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1992 MHz, 4 Core(s), 8 Logical Processor(s)

- We have studied about various functions in OpenCV and used them for the project.
- There is a further scope of automating the code using ML algorithms like KNN for object detection (for identifying reference pole and setting ROI and threshold values automatically).

## Appendix- A

- **Kernel Convolution**

In Image Processing a Kernel, or a mask is nothing but a small matrix (generally 3x3 or 5x5). Which is used for Filtering the images, Sharpening the images, edge detection and more. These processes are done by performing a convolution between a Kernel and an Image which is also a matrix but larger in size. Convolution is the process of adding each respective element of the Image to its local neighbours, weighted by the kernel. This is nothing but a form of mathematical convolution. The Matrix operation- Convolution is not a traditional matrix multiplication. This is denoted by '\*'.

For Example, if we have a three by three matrix as a Kernel and we have an image piece. To perform convolution, we take the kernel and do the operation on all the sets of matrices possible in the Image piece.

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} * \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} [2,2] = (a.1) + (b.2) + (c.3) + (d.4) + (e.5) + (f.6) + (g.7) + (h.8) + (i.9)$$

. The first matrix here is the Kernel and the second matrix is a part of the image piece with [2,2] as the central element of the subset.

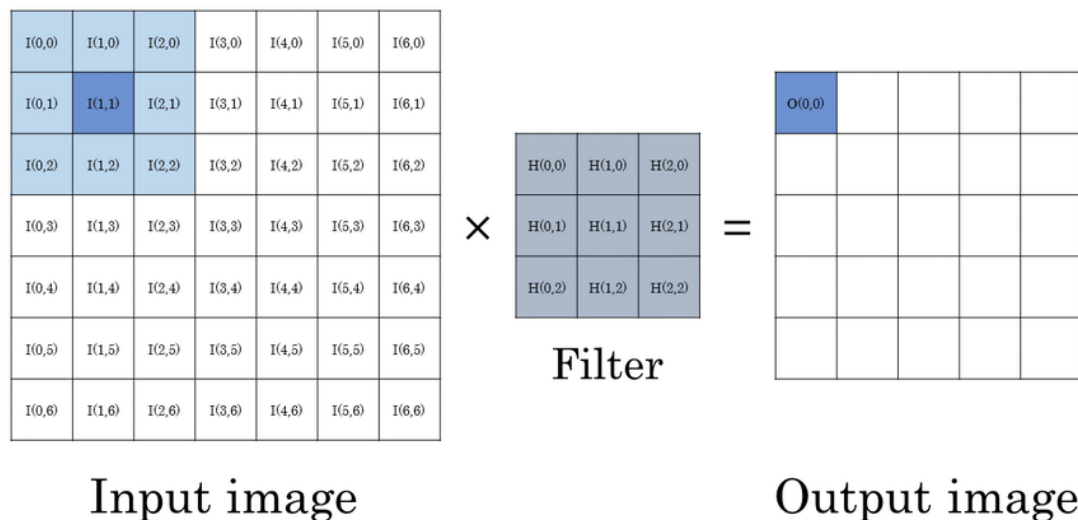


Figure 17: Kernel Convolution

- **Otsu's Algorithm**

In Computer Vision and Image Processing, Otsu's method is widely used for performing automatic image thresholding. In the simple terms, the algorithm returns a single intensity which is used as a Threshold value that separates pixels into two classes, foreground and background. The Algorithm comprehensively searches for the threshold which gives a minimum intra-class variance, which is defined as the weighted sum of variances of the two classes

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Here weights  $\omega_0$  and  $\omega_1$  are the probabilities of the two classes separated by a threshold  $t$ , and  $\sigma_0^2$  and  $\sigma_1^2$  are variances of the two classes.

Where  $\omega_0 + \omega_1 = 1$ . The class probabilities and class mean can be calculated iteratively.

## Appendix- B

### The developed code for Phase-I

```
import numpy as np
import cv2

# we will read the image directly as a grayscale image
img = cv2.imread('sample 4.png', cv2.IMREAD_GRAYSCALE) # gaussian filter is not used explicitly as it is already included in
img = cv2.Canny(img,50,200)
h,w = img.shape #image dimensions in (no. of rows, no. of columns)
lines = cv2.HoughLinesP(img,1,np.pi/180,150,maxLineGap = 40) # HoughLines function gives an output of an array with end points
no_of_lines = len(lines) # this gives no. of edges lines identified and crossed vote limit by hough transformation.
line_lengths = np.zeros(no_of_lines,dtype = int) # to record the line lengths.
line_spacing = []
line_lengths

import math
for i in range(0,no_of_lines):
    y = lines[i][0][3] - lines[i][0][1]
    x = lines[i][0][2] - lines[i][0][0]
    line_lengths[i] = math.sqrt(y**2 + x**2) #for Line lengths
for i in range(0,no_of_lines-1):
    for j in range(i+1,no_of_lines):
        y = lines[i][0][3] - lines[i][0][1]
        x = lines[i][0][2] - lines[i][0][0]
        if (x != 0):
            a = round(np.degrees(np.arctan(y/x)))
        else:
            a = 90
        y1 = lines[j][0][3] - lines[j][0][1]
        x1 = lines[j][0][2] - lines[j][0][0]
        if (x != 0):
            b = round(np.degrees(np.arctan(y/x)))
        else:
            b = 90
        b = round(np.degrees(np.arctan(y1/x1)))
        if(a == b):
            midy = (lines[i][0][3] + lines[i][0][1])/2
            midx = (lines[i][0][2] + lines[i][0][0])/2
            midy1 = (lines[j][0][3] + lines[j][0][1])/2
            midx1 = (lines[j][0][2] + lines[j][0][0])/2
            string = "the lines "+str(i)+" and "+str(j)+" are parallel"
            dist = round(math.sqrt((midy1-midy)**2 + (midx1-midx)**2))
            line_spacing.append([string,dist])
        else:
            string = "the lines "+str(i)+" and "+str(j)+" are not parallel"
            dist = 0
            line_spacing.append([string,dist]) # for line spacing
```

## Appendix- C

### The Developed code for Phase-II

```
import numpy as np
import math
import cv2

# we will read the image unchanged
img = cv2.imread('field 1-2.jpg', cv2.IMREAD_UNCHANGED)
roi = img[600:1800, 50:440]
roi = cv2.GaussianBlur(roi,(3,3),0)
roi2 = roi.copy()
roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
roi = cv2.Canny(roi,100,550,3)

pixel_to_metric = 134.0596881
initial_pole = 464.0387914819
intial_angle = 89.26
lines = cv2.HoughLinesP(roi,1,np.pi/180,100,maxLineGap = 100) #Hough transformation.
no_of_lines = len(lines)

edges = [] #LINE SEPERATION
angles= []
lengths = []
for i in range(no_of_lines):
    if ((lines[i][0][3]-lines[i][0][1]) != 0 and (lines[i][0][2]-lines[i][0][0]) != 0):
        ang = abs(math.degrees(math.atan((lines[i][0][3]-lines[i][0][1])/(lines[i][0][2]-lines[i][0][0]))))
        if (abs(90 - ang) <= 5):
            edges.append(lines[i][0])
            angles.append(ang)
            lengths.append(math.sqrt((lines[i][0][2]-lines[i][0][0])**2 + (lines[i][0][3]-lines[i][0][1])**2))
        elif ((lines[i][0][2]-lines[i][0][0]) == 0):
            edges.append(lines[i][0])
            angles.append(90)
            lengths.append(math.sqrt((lines[i][0][2]-lines[i][0][0])**2 + (lines[i][0][3]-lines[i][0][1])**2))
temp=angles.copy()

for i in range(len(edges)): #LINES DRAWN
    cv2.line(roi2, (edges[i][0],edges[i][1]), (edges[i][2],edges[i][3]), (0,255,0), 1)

index = [angles.index(max(angles))] # 2 LINES CLOSER TWO 90 DEGREES
temp[index[0]] = 0
index.append(temp.index(max(temp)))
```



```

for i in range(len(edges)):          #LINES DRAWN
    cv2.line(roi2, (edges[i][0],edges[i][1]), (edges[i][2],edges[i][3]), (0,255,0), 1)

index = [angles.index(max(angles))]  # 2 LINES CLOSER TWO 90 DEGREES
temp[index[0]] = 0
index.append(temp.index(max(temp)))

                                # START AND END POINTS
if (abs(lengths[0]-lengths[1]) <= 20):
    start_point = (int((edges[index[0]][0]+edges[index[1]][0])/2), int((edges[index[0]][1]+edges[index[1]][1])/2))
    end_point = (int((edges[index[0]][2]+edges[index[1]][2])/2), int((edges[index[0]][3]+edges[index[1]][3])/2))
else:
    start_point = (edges[lengths.index(max(lengths))][0],edges[lengths.index(max(lengths))][1])
    end_point = (edges[lengths.index(max(lengths))][2], edges[lengths.index(max(lengths))][3])
cv2.line(roi2, start_point, end_point, (0,0,255), 1)

                                # SNOW LEVEL CALCULATIONS.
pole_length = ((start_point[0]-end_point[0])**2 + (start_point[1]-end_point[1])**2)**0.5
snow_level = initial_pole - pole_length
snow_level_metric = snow_level/pixel_to_metric #angle of the pole is neglected because only edges with 90 +- 5 is taken and
snow_level_metric = round(snow_level_metric,5)
print("snow_level_metric:",snow_level_metric)

cv2.putText(roi2, str(snow_level_metric), (30,434), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,255,0), 2)
cv2.putText(roi2, "mts", (30,454), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,255,0), 2)

#cv2.namedWindow('roi',cv2.WINDOW_NORMAL)
#cv2.imshow('roi',roi2)

#cv2.waitKey(0)

#cv2.imwrite("field 1-2 edited.jpg", roi2)
#cv2.imwrite("field 1-2 binary.jpg", roi)

```

## References

- [1] F. Lin, W. Chang, "Applications of Image Recognition for Real-Time Water Level and Surface Velocity, 2013 IEEE International Symposium on Multimedia, Anaheim," 24 Feb 2014. [Online]. Available: <https://ieeexplore.ieee.org/document/6746801>. [Accessed 26 may 2020].
- [2] Space telescope, "space telescope.org," 05 Jan 2011. [Online]. Available: [https://www.spacetelescope.org/static/projects/fits\\_liberator/image\\_processing.pdf](https://www.spacetelescope.org/static/projects/fits_liberator/image_processing.pdf). [Accessed 24 may 2020].
- [3] Meng, Thomas Hies and Tan kok, "Research gate," Jan 2012. [Online]. Available: [https://www.researchgate.net/publication/262337135\\_Enhanced\\_water-level\\_detection\\_by\\_image\\_processing](https://www.researchgate.net/publication/262337135_Enhanced_water-level_detection_by_image_processing). [Accessed 23 may 2020].
- [4] Math Works, "Math works.com," [Online]. Available: <https://in.mathworks.com/discovery/edge-detection.html>. [Accessed may 2020].
- [5] Anonymous, "Google Patents," 04 Nov 2005. [Online]. Available: <https://patents.google.com/patent/JP4001162B2/en>. [Accessed 25 may 2020].
- [6] F. E. Samann, "Real-time Liquid Level and color Detection system using Image Processing," Apr 2018. [Online]. Available: [https://www.researchgate.net/publication/326345616\\_Real-time\\_Liquid\\_Level\\_and\\_color\\_Detection\\_system\\_using\\_Image\\_Processing](https://www.researchgate.net/publication/326345616_Real-time_Liquid_Level_and_color_Detection_system_using_Image_Processing). [Accessed 26 may 2020].
- [7] OpenCV, "OpenCV docs," [Online]. Available: [https://docs.opencv.org/3.4/d2/d2c/tutorial\\_sobel\\_derivatives.html](https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html). [Accessed may,June 2020].
- [8] OpenCV, "OpenCV docs," [Online]. Available: [https://docs.opencv.org/trunk/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html). [Accessed June 2020].
- [9] A. V. Mamta joshi, "Comparison of Canny edge detector with Sobel and prewitt edge detectors using different image formats," 2014. [Online]. Available: <https://www.ijert.org/research/comparison-of-canny-edge-detector-with-sobel-and-prewitt-edge-detector-using-different-image-formats-IJERTCONV2IS03009.pdf>. [Accessed 29 may 2020].
- [10] OpenCV, "OpenCV docs," [Online]. Available: [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_houghlines/py\\_houghlines.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html). [Accessed june 2020].
- [11] Jupyter, "Jupyter.org," [Online]. Available: <https://jupyter.org/>. [Accessed 11 June 2020].
- [12] Anaconda, "Anaconda.com," [Online]. Available: <https://www.anaconda.com/>. [Accessed 20 may 2020].
- [13] OpenCV, "OpenCV docs," [Online]. Available: [https://docs.opencv.org/2.4/modules/core/doc/drawing\\_functions.html](https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html). [Accessed 22 june 2020].

- [14] Sofiane Sahir, "Canny Edge Detection step by step in Python- Computer Vision," 25 January 2019. [Online]. Available: <https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123>. [Accessed 21 June 2020].
- [15] Abhisek Jana, "Implementation Canny edge Detector using Python from scratch," 20 May 2019. [Online]. Available: <http://www.adeveloperdiary.com/data-science/computer-vision/implement-canny-edge-detector-using-python-from-scratch/>. [Accessed 21 June 2020].
- [16] OpenCV, "OpenCV docs," [Online]. Available: [https://docs.opencv.org/2.4/modules/imgproc/doc/structural\\_analysis\\_and\\_shape\\_descriptors.html](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html). [Accessed 19 June 2020].
- [17] N. Cooling, "Code quality- Cyclomatic Complexity," 27 July 2018. [Online]. Available: <https://blog.feabhas.com/2018/07/code-quality-cyclomatic-complexity/>. [Accessed 20 June 2020].
- [18] Pypi, "pypi.org," 5 january 2020. [Online]. Available: <https://pypi.org/project/lizard/>. [Accessed 20 June 2020].

## Glossary

**Avalanche:** A heavy mass of snow and rocks coming down across a Mountain side which can be dangerous.

**Binary Image:** Images with only two possible intensity values for pixels. Normally displayed as black and white.

**Bitmap:** A bitmap is nothing but a type of image in matrix form. Where each cell (pixel) represents an intensity or colour.

**Canny Edge detector:** Canny Edge detector is a multi-step Algorithm used to detect a wide range of edges in images.

**Convolution:** In Image Processing Convolution is nothing but a matrix operation generally performed between kernel and an image iteratively.

**Cyclomatic Complexity:** A type of Complexity analysis, which gives us a quantitative measure of the number of linearly independent paths in the source code.

**Edge Detection:** An Image Processing technique used for finding the boundaries of objects within images.

**Grey Scale Image:** or Gray scale image is simply the image in which the only colours are shades of Gray.

**Hough Line Transform:** Hough Transform is a popular technique used to detect any shape, if you can represent that shape in mathematical form.

**Hysteresis Thresholding:** This is a form of thresholding involves using a maximum and a minimum after then perform Hysteresis tracking.

**HSV image:** short for Hue, saturation and value is a colour model which is used sometimes in place of an RGB colour model.

**Image Calibration:** A ratio which provides a pixel to real distance conversion factor.

**Image Filter:** A software process which changes the appearance of an image or a part of an image by changing the shades and colours of the pixels in some manner.

**Image Processing:** A method to perform some operations on an image, in order to get an improved image or to extract some useful information from it.

**Image Segmentation:** It is a process of Partitioning digital Image into multiple segments.

**Kernel:** A small matrix or a 2D Array with weights.

**Pixel:** or a Picture element is a Physical point on a bitmap Image, this is the smallest element in an image or in a display device like Television.

**Prewitt Edge Detection:** This is an Algorithm which is used for the detection of Horizontal and vertical edges in an image

**Reference Image:** The image based on which the initial calculations are done, which are later used for the analysis of field images.

**Sobel Edge Detection:** An Edge detection Algorithm widely used for its emphasis on the pixels which are closer to the centre of the kernel.

**Thresholding:** An Image Processing method that produces binary images based on setting a Threshold value on pixel intensity of the original image.