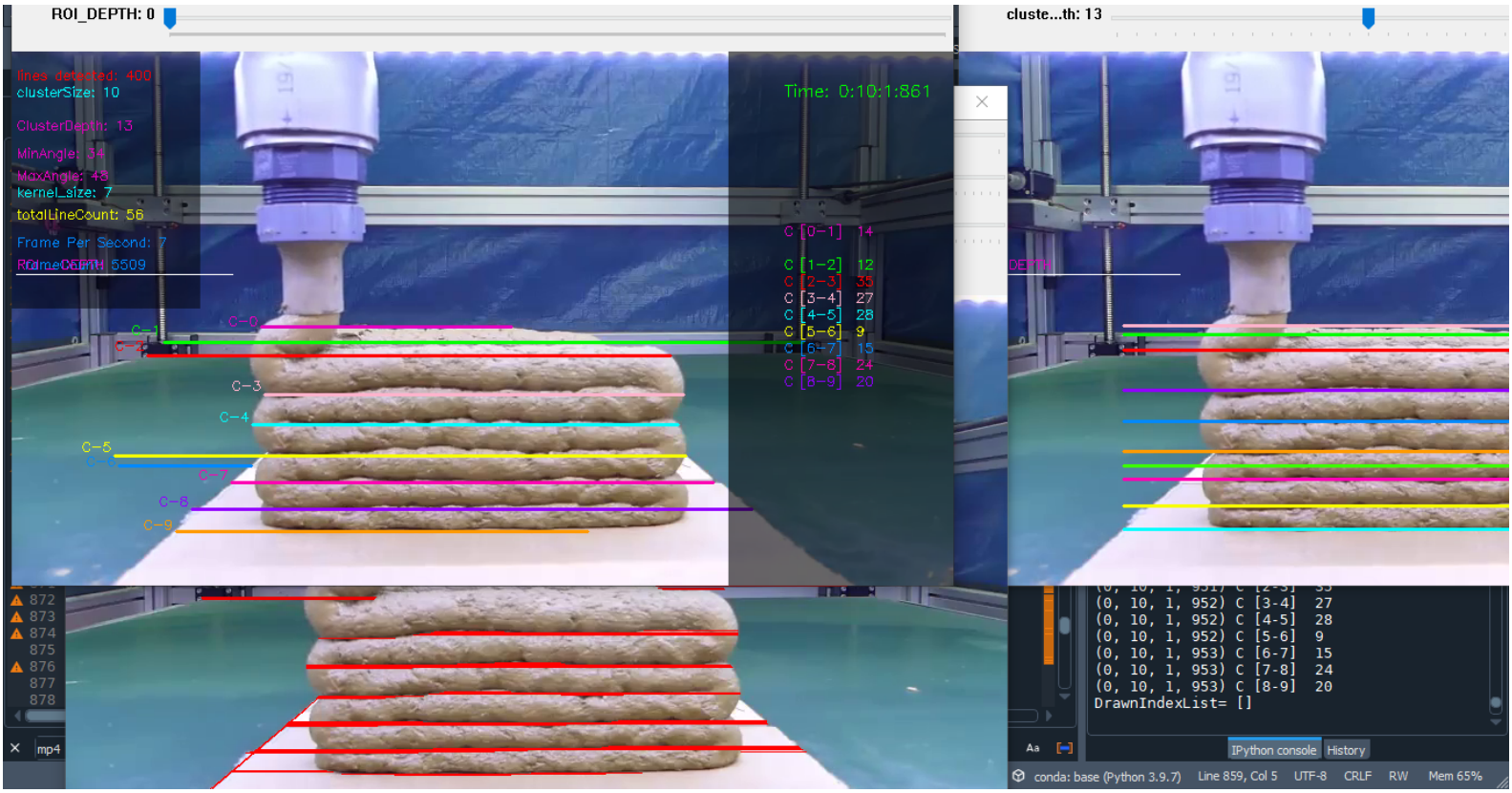
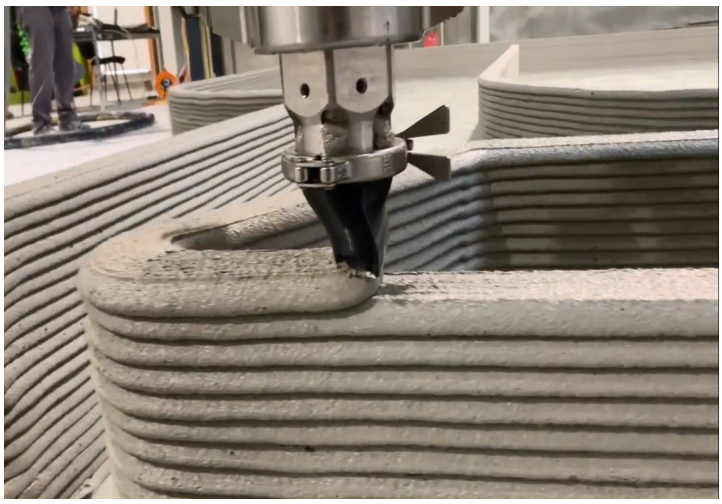
AN EXPERIMENT OF DIGITALIZATION OF THE CONSTRUCTION INDUSTRY:

CONTROLLING 3D PRINTED ELEMENTS WITH COMPUTER VISION AND AI





A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF MATHEMATICS

AND THE COMMITTEE ON GRADUATE STUDIES

OF CY TECH CERGY PARIS UNIVERSITY,
IN COLLABORATION WITH

DEPARTMENT OF CIVIL ENGINEERING

UNIVERSITY OF CERGY PONTOISE,

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER'S IN BIG DATA (ADEO2)

ACADEMIC YEAR 2021-2022

Supervisor:

Professor AbdelHak Kaci,

Department of Civil Engineering

University of Cergy Pontoise, France

By :

Md Mahbubul Islam &

Sela Zoumanigui

Students, Master's in Big Data (ADEO2)

CY Tech Cergy Paris University, France

MARCH 29, 2022

Contents

Abstract

3D printing is growing rapidly in the manufacturing industry and has gained a lot of attention from various fields owing to its ability to fabricate parts with complex features. The reliability of the 3D printed parts has been the focus of the researchers to realise additive manufacturing as an endpart production tool. Machine Learning (ML) has been applied in various aspects of 3D to improve the whole design and manufacturing workflow especially in the era of industrial revolution . In the review article, various types of ML techniques are first introduced. It is then followed by the discussion on their use in various aspects of AM such as design for 3D printing, material tuning, process optimization, in-situ monitoring. Potential applications building & constructions will be highlighted. Measurement results previously provided by a database that is primarily used to improve existing models for usual processes. Considering the complexity of the parameters and process inspection, we use machine learning to ensure quality manufacture of the product by first improving the quality of the image from the printing video focusing on tracking the area of interest and predict the target function against the input.In-situ monitoring of 3D processes will significantly benefit from the object detection ability of Machine Learning. As a large data set is crucial for ML, data sharing of Additive Manufacturing would enable faster adoption of ML in 3D. Standards for the shared data are needed to facilitate easy sharing of data. The use of ML in 3D printing will become more mature and widely adopted as better data acquisition techniques and more powerful computer chips for ML are developed.

Acknowledgment

We would like to acknowledge and thank our supervisor Abdelhak Kaci for this support , follow ups and encouragement during the research work.

Special thanks also to Professor Rachid Celouah , the director of the program ADEO2 , CY Tech University , for his continuous encouragement and followup.

Thank you.

Introduction

This research is important because it's about promoting development Innovative building systems from efficient processes while minimising ecological risks construction operations (consumption, transportation, CO2 waste reduction, rational use of resources), optimization Economic efficiency (reduced construction cost and time, quality assurance) and improved occupational safety (remote control, less pain). Reflections on training programmes adapted to new technologies and digital change in the construction industry is also closely related to the development of related professions[8]. Use multiple science skills materials (formulations), mechanics (rheology, characterization in hardened state), chemistry (through cement), Artificial Intelligence , Instrumentation (Physical Measurements). Task automation acquisition enables the creation and implementation of an industrial image processing system that will have the purpose of reviewing and controlling the process and quality of making 3D concrete.

The applications of machine learning are potentially endless. However, one noteworthy application is the automated monitoring of 3D printed parts. Supervised machine learning algorithms can seamlessly monitor the 3D printing process by integrating features such as a camera and image processing.
Machine vision should then make it possible to propose a formulation according to the parameters of the raw materials available on site (data 7 input: grain size, water content, adjuvant properties, etc.) by integrating related fixed parameters the process (pipe diameter, circuit length, etc.) and those to be regulated (pressure, flow, etc.).

Additionally, quality control of the 3D printed parts has gained wide attention from the industries to ensure parts fabricated for functional use satisfy specific requirements, particularly in quality and reliability. As more advanced 3D materials are developed and used in critical parts of structures (Wong and Hernandez 2012, Madara and Selvan 2017), high part quality must be ensured. Unwanted porosity is a known issue in 3D printed part processes (Aboulkhair, Everitt et al. 2014, Liu, Guessasma et al. 2018). These porosities significantly affect the mechanical performance of the 3D printed parts[7]. A study has shown that a highly dense 3D printed part (>99.8%) can be produced using a well-controlled system (Sing, Wiria et al. 2018, Yu, Sing et al. 2019). In-situ quality control techniques are therefore required to further improve the quality of the 3D printed parts through detection of anomaly and corrective printing using closed loop feedback.

This report presents a review of the research development concerning the use of ML in 3D printing, especially in the areas of monitoring quality control for 3D printing parts, process optimization, in-situ monitoring for quality control and setting a timer to track the change in various parameters. The organisation of the report is as follows: first section shows details of the elements that are used, the classification, etc…….. Next, show a detailed step of the evolution of the work that has been done.

Keywords

Abbreviation	Elaboration	Abbreviation	Elaboration
OpenCV	Open Computer Vision (Library)	CNN	Convolutional Neural Network
IOT	Internet Of Things	AM	Additive Manufacturing
ROI		UOM	Unit Of Measurement
ML	Machine Learning,		
AI	Artificial Intelligence,		
3D printing	3 Dimensional printing		

Importance of Digitization of Construction Industry

The trend of digitization in construction is increasing over the past years, however, now its speed is accelerating.

The Construction Industry is reshaping itself, albeit slowly but certainly faster than previously[6]. Pressure for change is coming from several complementary directions:

Evolving client expectations

Clients, influenced by other rapidly changing markets (such as B2C with platforms that have triggered new relationships, tailored products and powerful service levels), now expect also the same from their homes, offices, commercial buildings and infrastructures to make their “connected lives” even more a reality. Constructions need to be more and more individualised, modular, connected to the Internet of Things (IoT) and allow for specific performance tracking, optimization of energy and improved security and health parameters for instance. Client demands are quickly rising and become more and more complex with expectations increasingly on the “usage” more than on the product itself.

New technological capabilities

Sensors and various hardware as well as software have seen cost drop and efficiency rise over the past few years opening the path to new possibilities. Technologies available on the market are more

numerous than ever before (such as virtual and augmented reality, drones, robotics and additive printing) making it urgent to separate the more valuable ones from mere novelties.

New generation of craftsmen and professionals

Tech savviness is spreading in the construction industry, which traditionally has been resistant to change, accelerating the adoption of digital tools. Innovative university curricula are training the younger generations for emerging tech-related jobs. Many new jobs, not yet known, will be created in the years to come with the adoption of new tools and processes.

Booming start-up environment

Startups have taken advantage of the market opportunities induced by some of these trends to fill newly created added-value gaps. Oliver Wyman has identified nearly 1,200 startups worldwide since 2010 in real estate and construction. These startups have received around US$19.4 billion in funding over the period, half of it in 2017.

Supportive legal frameworks

Governments, particularly in the Nordic countries and the UK, are increasing their CO2 and energy efficiency regulations and raising their targets. Digitalization provides a great opportunity to reduce the environmental impact of construction projects. There will also be heightened requirements on data usage and cyber security in buildings and infrastructures going forward that will need to be carefully analysed (General Data Protection Regulation). Importantly, labels and groups are also more and more launched to help the market move in one common direction and support innovation.

Launch of large infrastructure projects

Market needs are tremendous in terms of new infrastructure networks between cities (such as the Grand Paris Express with 200km of new automated metro lines in France, the High Speed 2 in the UK or the Rastatt tunnel in Germany) as well as in terms of upgrading partly old existing structures

All players in the industry, whether they are promoters, engineering companies, builders, suppliers of equipment, materials, or distributors will be impacted by digital pressures. Each one will experience this differently, of course, but there is no doubt that significant change is coming. The outlined trends put pressure on incumbents (both equipment and traditional construction and service players) by producing a more complex and dynamic competitive landscape and the progressive disruption of traditional channels in the battle for customer access and control

Role of 3D Printers in Constructions

[4] 3D printing in construction is an emerging technology that is still under development, this additive manufacturing certainly holds promise for the future of construction, with its many benefits- additional speed, reduction of human error, and decreased waste from production processes,

Faster, affordable construction - Where a build may sometimes take weeks or months to complete-with 3D printers, the work can often be accomplished in mere days or even in 24 hours you have a home. Now additional build time is available for more projects, which lowers overall construction costs.

New markets - 3D printers are opening up new avenues to be relevant within the construction market. For example, in climates where seasonal weather limits the building schedule, pre-printed components can be assembled quickly, providing a competitive edge for the builder.

Decreased waste and brand improvement - The construction industry is often labelled as wasteful and not environmentally friendly but with 3D printing, Contractors work with components that can be printed to order, so no waste. Any unused materials can generally be recycled (which will contribute to the reduction of environmental pollution). Additionally, some 3D printers can use locally sourced materials which improve the sustainability of the build.

Reduced health and safety risks - Construction, especially concrete work can be hazardous but with 3D printing, the risks are minimal -so a whole lot less worker's compensation paperwork to file.

Design freedom - 3D concrete printing enables you to make any shape. You can bend it, you can make angles, you can make virtually any organic shape you want.

Reduce human error- Statistics show that more than 5,000 workers are killed on the job each day. Because construction would be more programmable and automated, worker injuries and fatalities would likely decrease if 3D printing was incorporated onto the jobsite.
But despite its potential and some impressive early use cases, 3D printing has yet to become a widely used tool in most construction projects.

Figure1: 3D printed house designs

Statement of the problem

To control 3D printed elements with computer vision and artificial intelligence and perform overall quality assurance of the3D printing in construction.

Description of the 3D Printer System

The 3D printer has an X,Y and Z axis system where the printer head stays on the Z axis. The movement of the head is controlled by an IOT system like Raspberry Pi. The mould or casting material works as an “INK” for the printer.

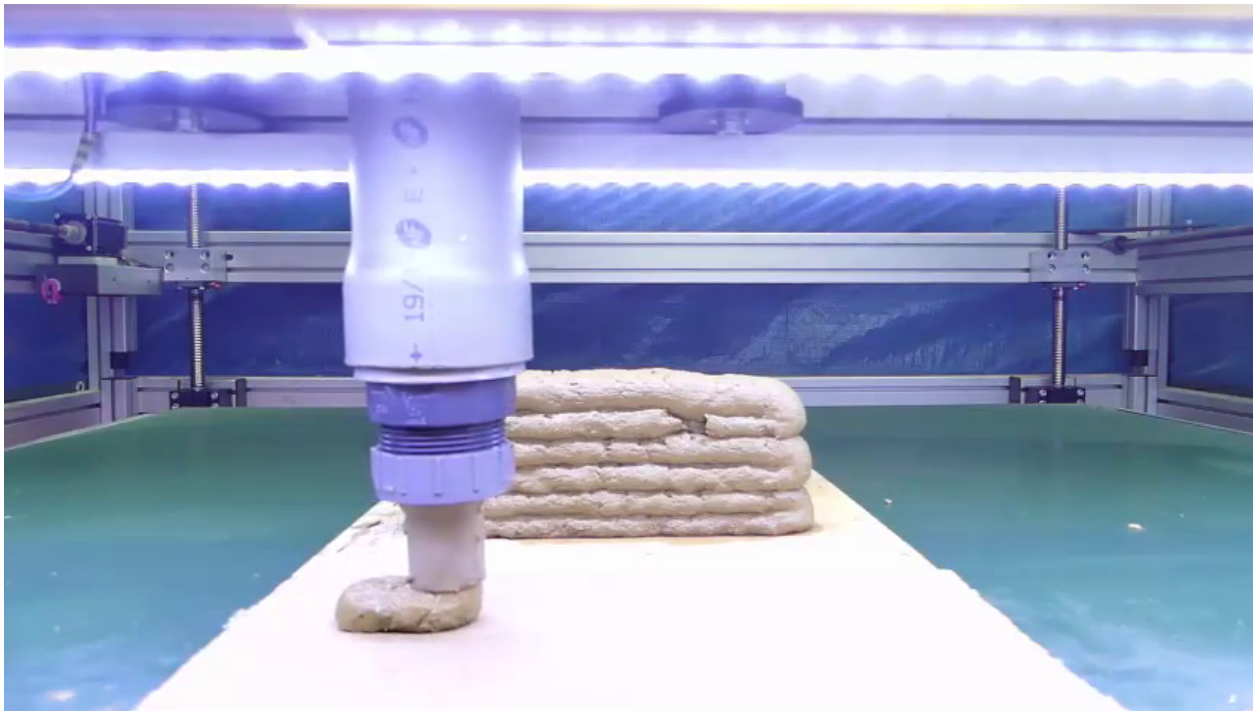


Figure2: Definition of an experimental 3D printer

Objective

In order to solve the problem we decided to use the OpenCV library which is a popular computer vision library. And Python as the programming language. First step is to extract each image frame from the video stream and perform the following : operations:

Transform the BGR image to GREY . Apply Gaussian blur . Apply morphological transformation on the image by adjusting kernel size as needed uniti we get the desired result. Apply Canny and HoughLine algorithms to finally get the lines detected on the image.
The detected lines are arbitrary and we do not have much information on them . In order to identify the number of lines that represent the separating gaps or distortions between two layers of the printed materials we need to cluster the lines into different groups. A group of lines represent a certain gap or a distortion. Once the clustering is done we now know which lines are representing which gaps and which distortions. To view the different clusters we colour them with distinctive colours than its neighbours. . Now We need to sort the groups . And then measure distance between the adjacent clusters. We need to keep track of the time as well as the reading of distance . This data is then passed to a MQ (Message Que) server to asynchronously spit off the data without blocking the main processing thread. The MQServer (for example RabbitMQ or Kafka) will then save this data using REST API to the database for further processing. This time series data will be used in the regression analysis to predict various properties of the mould and the printed elements. This process is shown in the logical diagram below.

Logical diagram

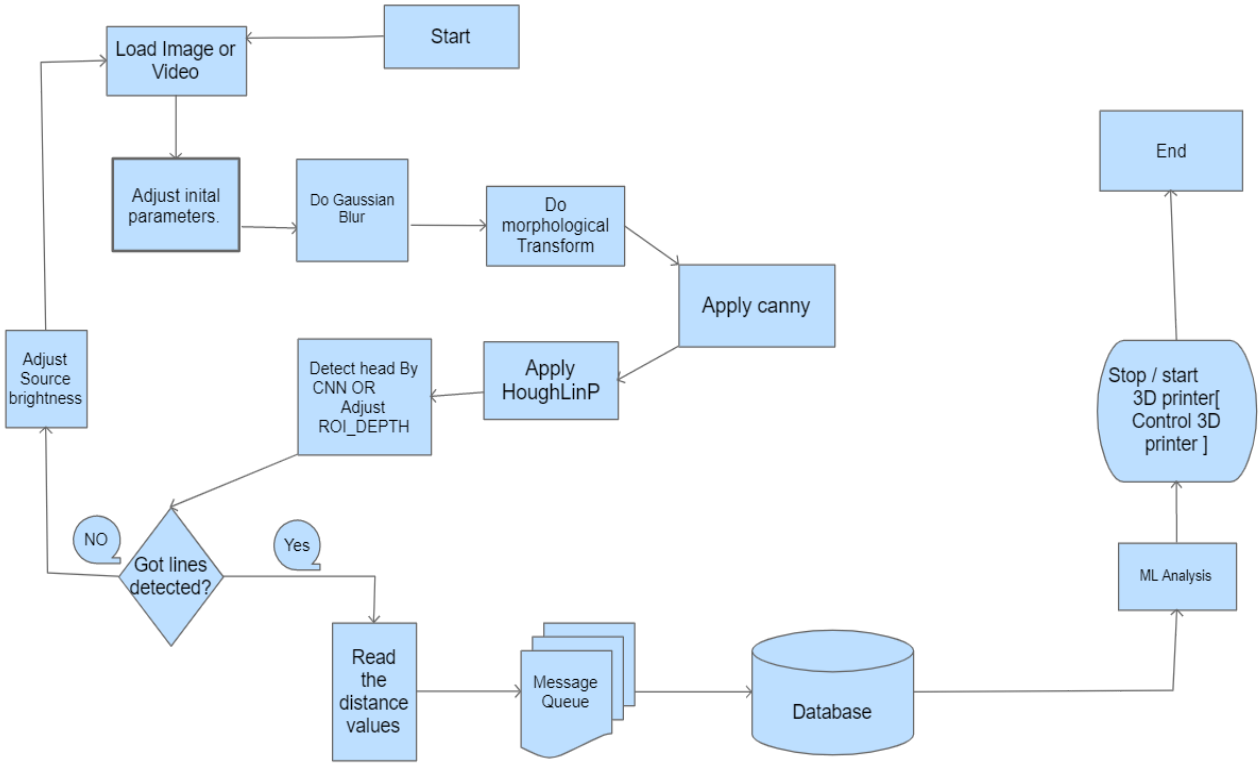


Figure 3: Logical diagram of the whole process.

Methodologies

OpenCV

[6] (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source Apache 2 Licence. Starting in 2011, OpenCV features GPU acceleration for real-time operations.

Python language

[5] Python is a high-level, general-purpose programming language. Its design philosophy emphasises code readability with the use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small- and large-scale projects.
Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library

Gaussian Blur

[1] In this method, instead of a box filter, a Gaussian kernel is used. It is done with the function, cv.GaussianBlur(). We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image.
It is also possible to create a Gaussian kernel with the function, cv.getGaussianKernel().
The above code can be modified for Gaussian blurring:
We can set the value of the kernel_size variable to 1, 3, 5, 7 etc until we get a satisfactory result.
blur_gray = cv2.GaussianBlur(grey, (kernel_size, kernel_size), 0)

Morphological Transformation

The light source on the image is very crucial in order to get the desired result.
[2] Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, the second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation.

Erosion

Figure 4: Gaussian blurred image

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of the foreground object (Always try to keep the foreground in white). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel are 1, otherwise it is eroded (made to zero).
So what happens is that all the pixels near the boundary will be discarded depending upon the size of the kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image.

Dilation

It is just the opposite of erosion. Here, a pixel element is '1' if atleast one pixel under the kernel is '1'. So it increases the white region in the image or size of the foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because erosion removes white noises, it also shrinks our object. So we dilated it. Since noise is gone, they won't come back, thus our object area increases. It is also useful in joining broken parts of an object.

Morphological Gradient

It is the difference between dilation and erosion of an image.

```
detected_lines = cv2.morphologyEx(blur_grey, cv2.MORPH_OPEN, vertical_kernel, iterations=2)
```

Colour Function (Grey) and Canny function

We use the function grey to eliminate the colours and have the image in shades of grey The class named Imgproc of the package org.opencv. imgproc provides methods to convert an image from one colour to another. Converting Colored Images to Grayscale A method called cvtColor() is used to convert colored images to grayscale. It is possible to convert colored images to grayscale by passing the code Imgproc. COLOR_RGB2GRAY with source and destination matrices as parameter to cvtColor() method.

To be able to be more precise for the function which will follow. Also,we will pass the grey image with a blur. During the Gaussian blur operation, the image is convolved with a Gaussian filter instead of the box filter.
The Gaussian filter is a low pass filter that removes high frequency components.
To perform this operation on an image using the Gaussianblur() method of the imgproc class.

Canny Image and it's difficulties

[8] The *Canny Edge detector* was developed by John F. Canny in 1986. Also known to many as the *optimal detector*, the Canny algorithm aims to satisfy three main criteria:

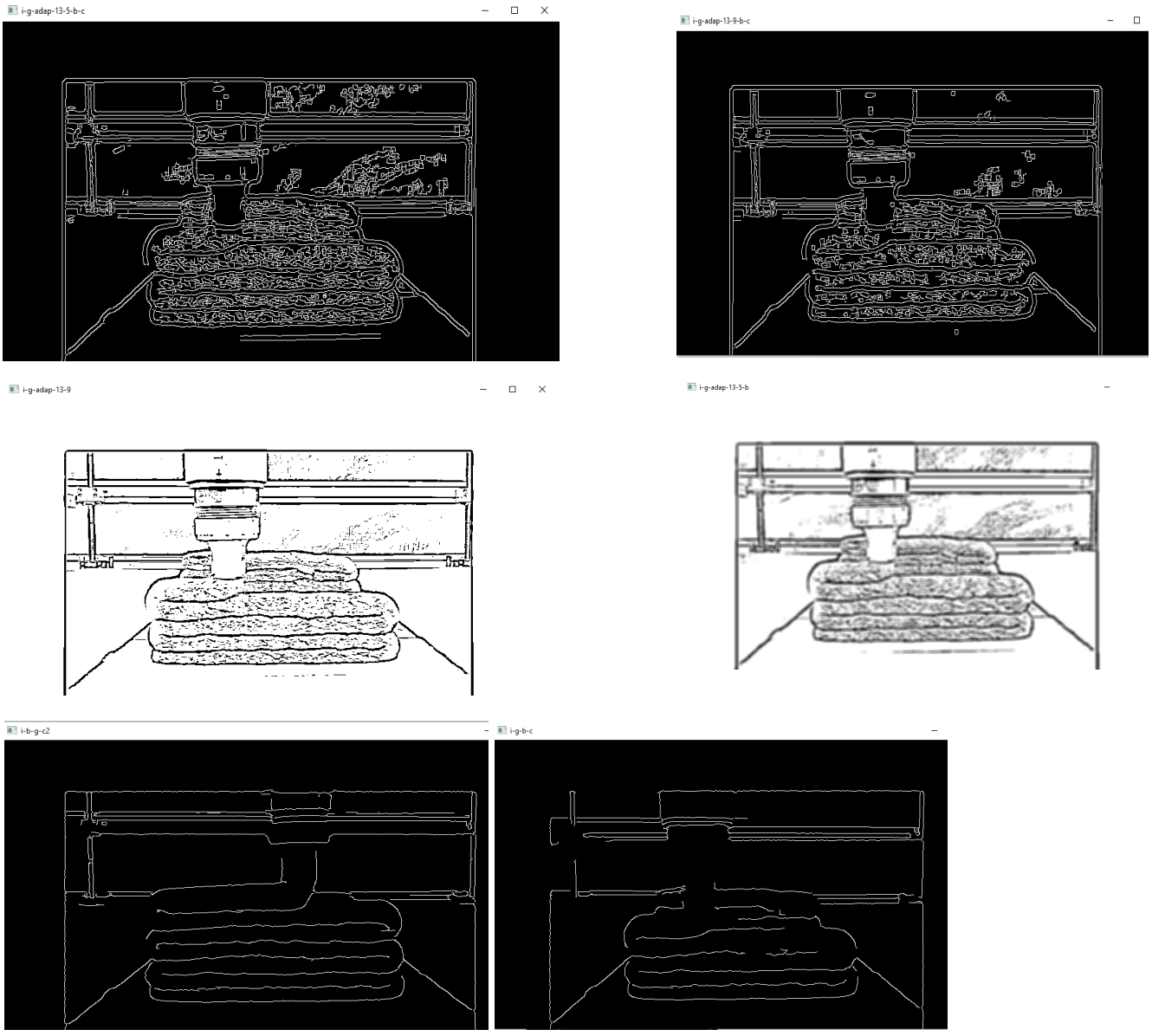
- **Low error rate:** Meaning a good detection of only existent edges.
- **Good localization:** The distance between edge pixels detected and real edge pixels have to be minimised.
- **Minimal response:** Only one detector response per edge.

As we started playing with canny images , different values of kernel size in gaussian blur and other parameters were producing various lines and in some cases contours. And these output of canny image and houghline transformation are not stable. Even for the same video where different images have almost the same environments. This was very frustrating . Then we decided to adjust all the key parameters on the fly ad and decide the best fit for the image and our purpose.

Standard and Probabilistic Hough Line Transform

[3] The Hough Line Transform is a transform used to detect straight lines.
To apply the Transform, first an edge detection preprocessing is desirable.
OpenCV implements two kind of Hough Line Transforms:

a. The Standard Hough Transform
It consists in pretty much what we just explained in the previous section. It gives you as result a vector of couples (θ,rθ)
In OpenCV it is implemented with the function HoughLines()
b. The Probabilistic Hough Line Transform
A more efficient implementation of the Hough Line Transform. It gives as output the extremes of the detected lines (x0,y0,x1,y1)
In OpenCV it is implemented with the function HoughLinesP()



Figures 5: Applying morphological transformation and canny to the images

Filtering out the vertical lines

At this point we have detected lines that we need. But there are horizontal and vertical lines .
We need only the horizontal lines , so we need to get rid of all the vertical lines in our calculations. For this we calculate the angle of each line . And we take only those lines that fall between 34 to 48.

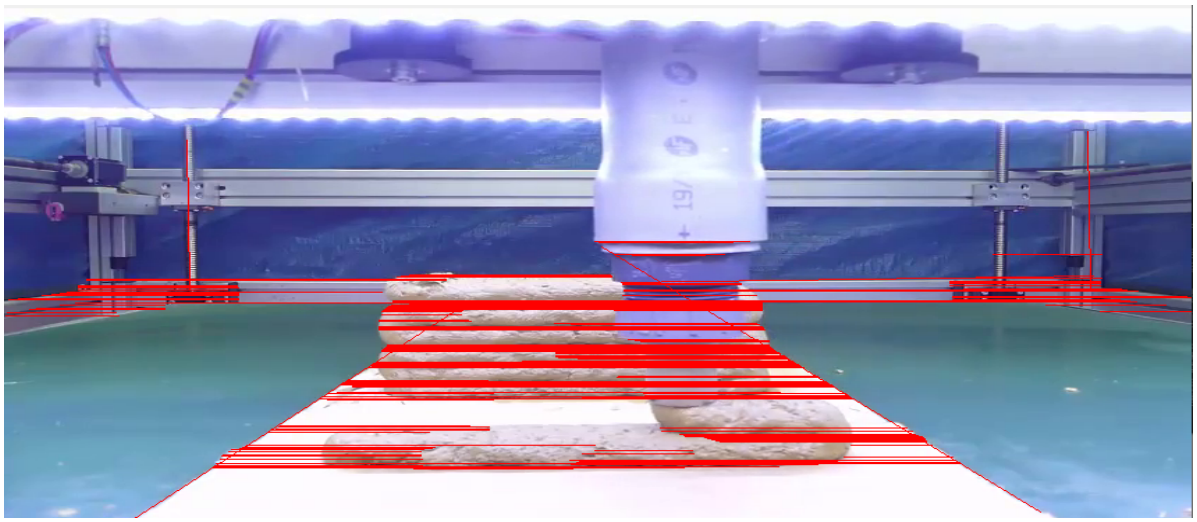


Figure 6: Lines directed after HaughLineP applied.

Taming the detected lines

Once the lines are detected ,we have the coordinates of the lines . Theres lines are scattered all over the image and have no orderting. We need to identify the locations of the lines and find only those lines that belong to the gaps of the printed mould layers and also the distortions of the gaps in between the layers of printed moulds.

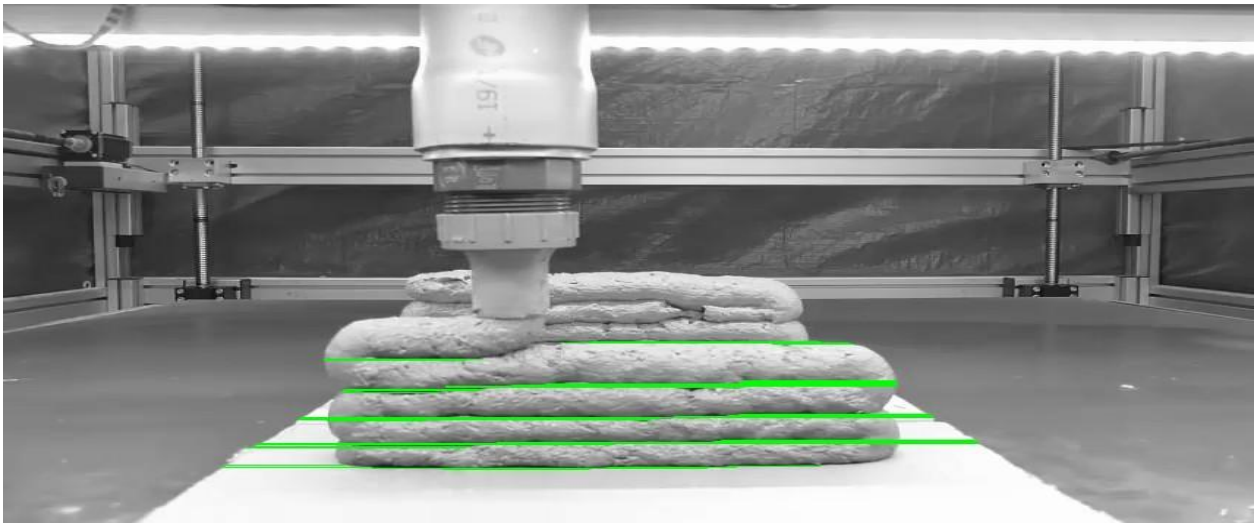


Figure 7: Taming the detected lines and keeping only those that we need.

Assign colours to identify each cluster

It's interesting to set different colours to different clusters to visually identify the clusters .We use a sequence of colours with their index values and map them by the cluster number .

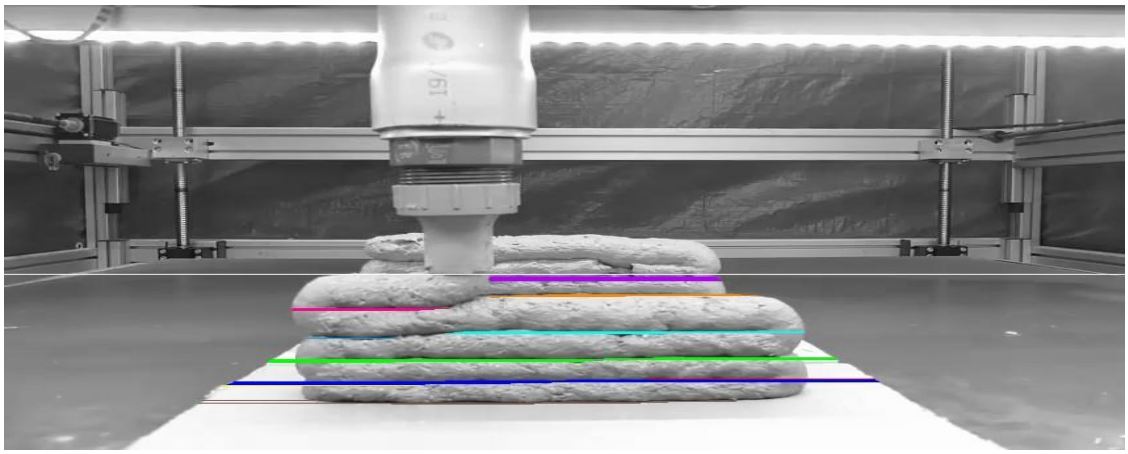


Figure 8: Assigning different colours to different clusters.

Clustering lines into groups

In order to group the lines that fall between the gaps of two adjacent layers of printed moulds , we group them in different clusters. So a cluster will consist of a bunch of lines that collectively represent a single line which falls in between the gaps of the layers.
The appearance of more lines or varied in numbers represent the expansion of the gaps or the distortions. In order to clusterise the line , we first set a variable called clusterDepth. This is the maximum height of a cluster. All the lines that fall within this height of the Y axis will be clustered together. So each cluster has a lower band and an upper band. Normally each cluster has a centroid . This is the average height of the cluster. All the clusters that fall within a range of clusterDept/2 from the centroid on both sides centroid +clusterDept/2 and centroid -clusterDept/2 are considered to belong to the same cluster. Other lines lie outside of the boundary of the cluster.

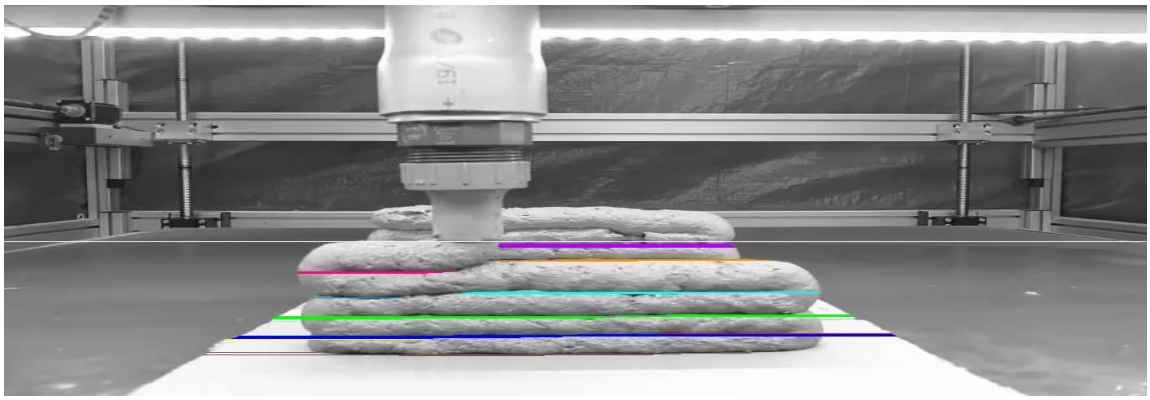


Figure 9: Clustering lines into different groups

Merging all the lines in a group

Once the clustering of the lines are done, we need to merge the lines to one single line so that it represents the gap. We tried various methods. First we printed a single thick line on the centroid. Later we found that it's more accurate to take the lowest boundary as the standards for representing the cluster line.

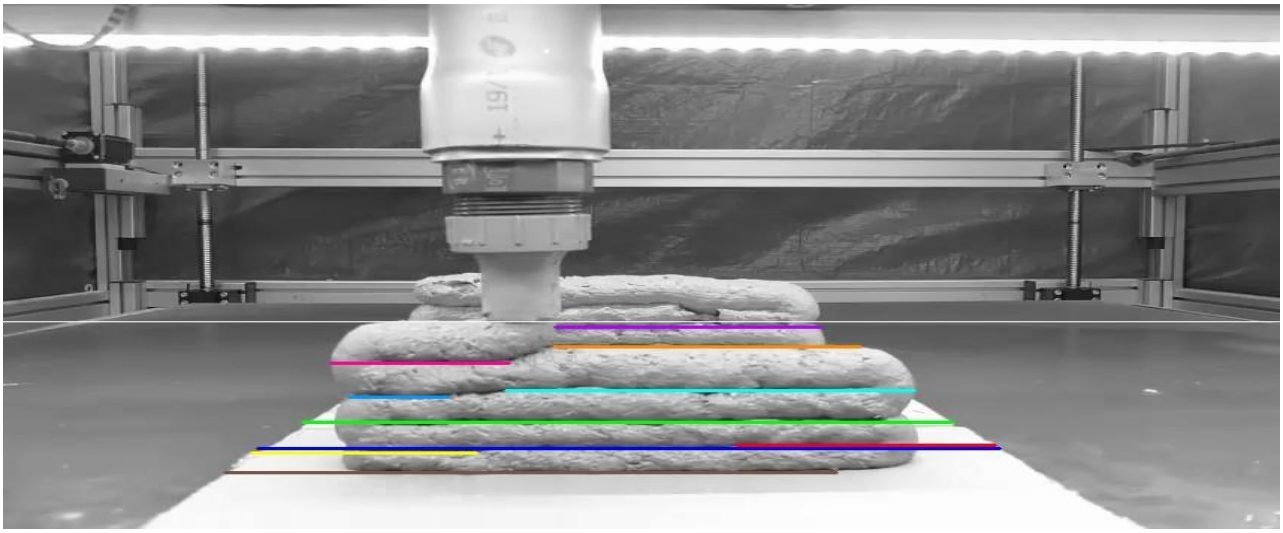


Figure10: merging all the lines in a cluster to one single line

Noise Clusters

As we can see in the above image, there are some noise clusters. We need to remove these noise clusters. For example, the second cluster from the bottom has 2 additional noise clusters. They represent distortion as well. There are some noise clusters on the right side of the head too. So, we need to get rid of all the clusters starting from the ROI_DEPTH (the first white line) till the first level of the cluster which is below the printer head. Then we will display the width or distance between adjacent clusters and print them on image with respective legends.

Sorting the clusters

Now that clustering is done, we need to sort the clusters. The clusters are stored in a dictionary of cluster objects. The keys are the cluster numbers whereas the clusters are represented by an object consisting of various information that are necessary for processing further procedures.
We define a comparator that compares the cluster numbers and sorts them in ascending order for example.

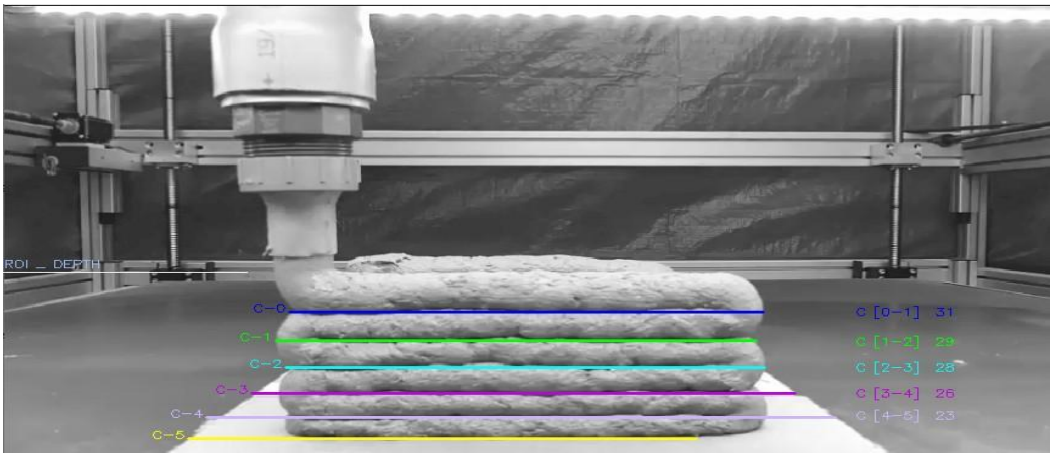


Figure 11: Measuring the distances between two adjacent clusters having a common intercept

In this image we can see the distance reading for cluster C-1 to C-2 is missing. This is because the distance between the adjacent clusters are calculated only if there is an intersection or overlapping between them. This can be improved to display the distance for the first two adjacent clusters that have an intersection between them.

Unified cluster length

We made each line to have an equal length (we unified the length of each cluster). But this assumption hides some important information like distortions. So, we will consider each cluster to have their original length. Thus, if we need to address the distortions in more detail, we can analyse these lines in the same cluster and compare with the cluster density and decide whether to ignore some of them or to report the distortion levels

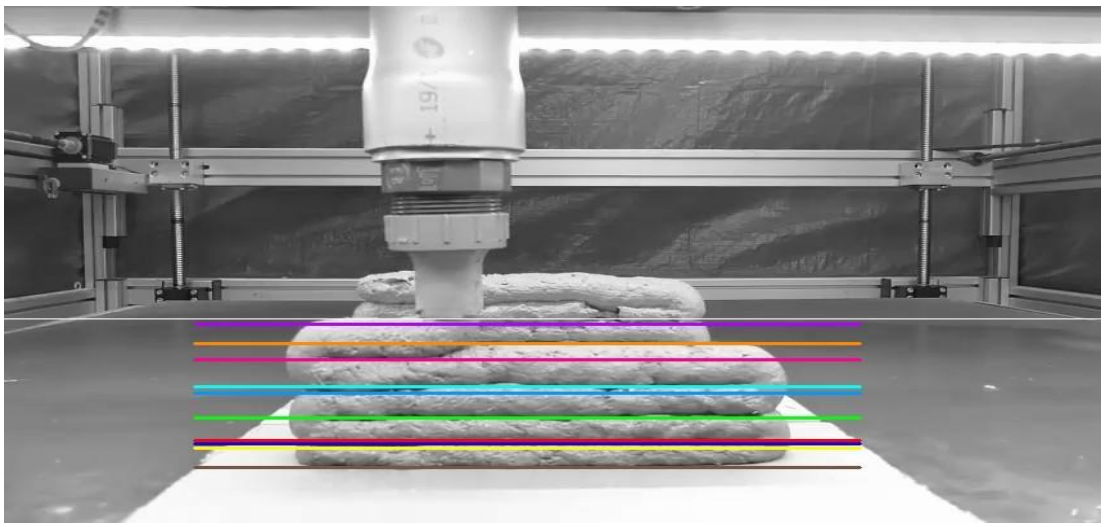


Figure 12: Unified cluster length

Region Of Interest ROI

Our experiment should concentrate our focus only to those areas where the printed moulds lie. This will eliminate unnecessary longer overhead and also avoid unwanted detections and processing of lines. Thus fixing a region of interest around the printed moulds only will boost up the frame wise performance and hence the collective performance as a whole. All the lines that fall beyond the ROI will be discarded . One way to achieve this is to define a Region Of Interest (ROI)

Defining ROI initially

Defining the ROI (Region of Interest) to increase the accuracy of the line detection. that means isolate the part of the image that only represents only the layers the molds and distortion if any, ignore the rest. We can start doing a selection using the mouse initially and see the result. There are thousands of lines detected.

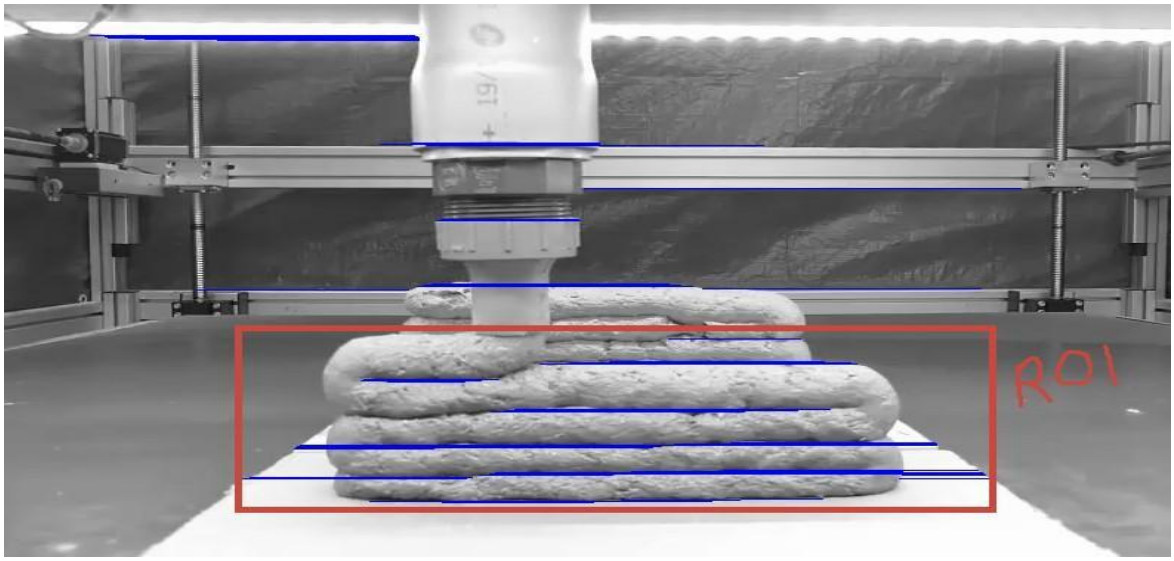


Figure13: Defining ROI

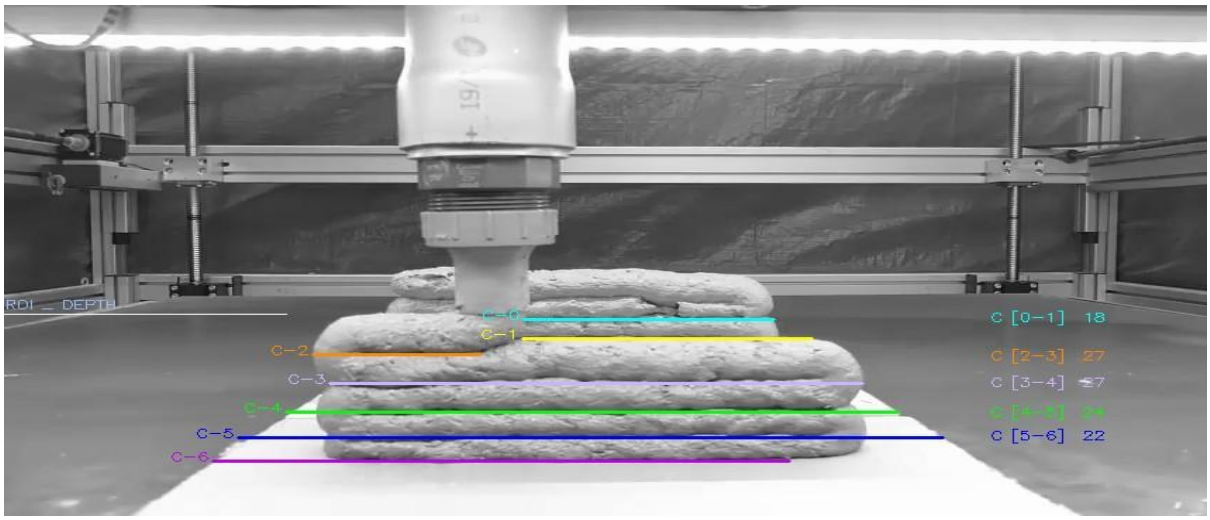
Currently we're decreasing the height of y axis and take only one layer or two in order to continue the research.
Starting by application of various morphological transformations on the image and observe it, to find a better set of parameters so that we can get the desired lines

We set ROI_DEPTH = 290 for the test image.

One way to achieve the selection of the ROI is to detect the head of the image and increase or decrease the region of interest as the head prints and increases the height of the region of interest.

Measuring distance between the adjacent layers.

The distance between two adjacent clusters is the distance between the lower bounds of each cluster. Calculate distance only if the adjacent layers have an intercept
In some cases the two adjacent layers are not intercepting . In that case we need to iterate through the rest of the layers until we find an intersecting layer . We then calculate the distance between these two layers.



Finger14: Measuring distance between the clusters

Generate time series data along with cluster distances

At last we print the time stamps in the format of HH:MM:SS:MSS where MSS is millisecond along with the cluster's information and the relevant distances to the log file for the time being . But in future we will spit this date asynchronously to a Message Queue and save it in the database automatically which intern will be used by machine learning algorithms to find out further insights like elasticity, viscosity, tolerance , effect of gravitational forces over increasing height and weights etc to determine and control the efficient output of the final printed construction units..

Results

Detection of the image and finding lines

Initially, the lines were detected on the image which contained more than 2000 lines. Then we decreased them to 1800 lines. we have calculated the shortest distance from each line to the there.

Testing with video input

Putting everything together and testing on a video

So far we test our experiments on the image only. Now we do our experiments with a video input instead of an image. The results are not satisfactory unless we adjust a few values.
It is obvious that the parameters need to tweek for different cases. Because some moulds are thicker and some are thinner . For example, the lower kernel size is used for thinner layers. And thus the

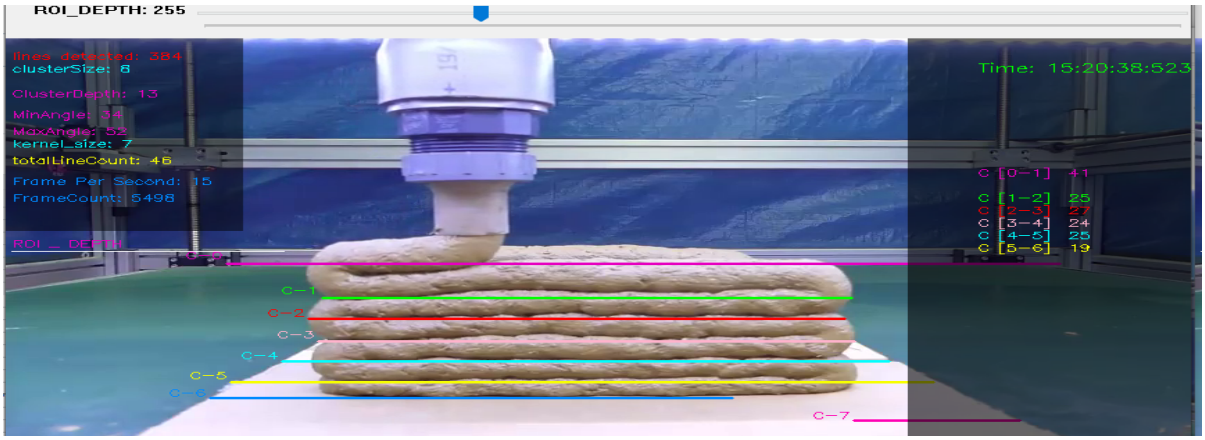


Figure15: Running the solution with a video

Lower value of the clusterDepth is needed. On the other hand the ROI_DEPTH is needed to adjust in order to fit the target moulds within scope.
The list of values and variables are listed below for the input video used in the experiment:

Variable	Lower limit	Upper limit	Optimum value (input video)	Remarks
KernelSize	1	12	7	Odd values only
ROI_DEPTH	0	Frame_Height	270	Adjust just below the head
minAngle	0	90	34	Lower bound of the angle
maxAngle	35	90	48	Setting less than 30 will read no lines

Table 1: Different variables for adjusting the result of the detection and processing

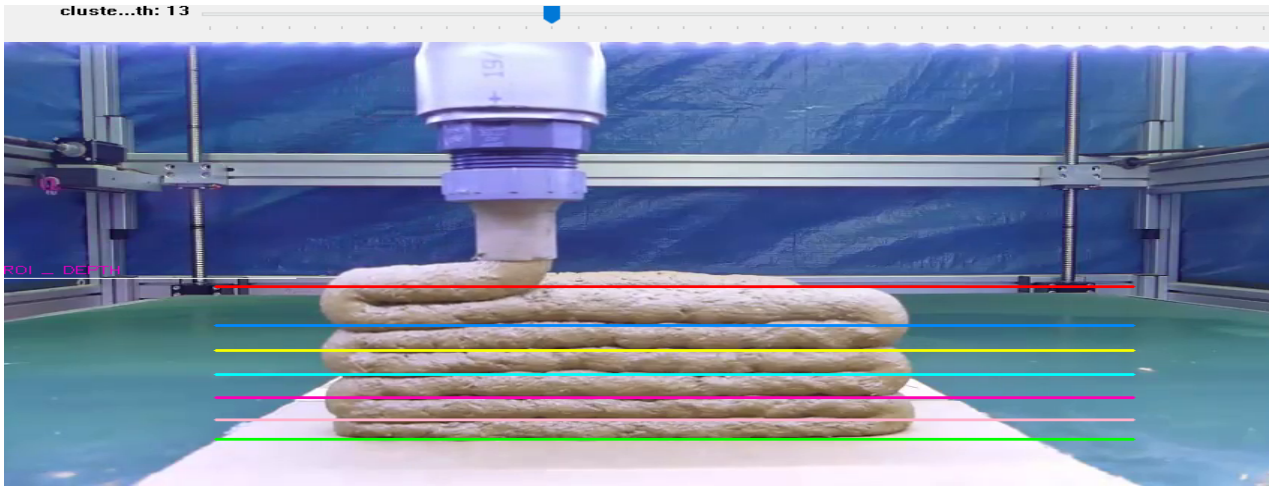


Figure 16: Adjusting the clusterDepth

Challenge to stabilise the distance values in a video

An overwhelming number of clusters and distance values are generated.in a too flickering manner to read the values..This is the case when we started testing the work on a video. There were a lot of clusters and the distance belonging to the clusters are also very transient and fluctuating. In these circumstances , it's not possible to read data and utilise it for further processing. To address the too many clusters outside the ROI_DEPTH , we adjusted the ROI_DEPTH so that the target 3D printed moulds fit within our scope. In this condition we have much comfortable results consisting of fewer clusters . But the fluctuations did not decrease. In order to tame the fluctuation, we tweak various variables . Firstly finding the optimum kernelSize by sliding the trackbar . Secondly fixing the proper clusterDepth value that suits the height of the mould. And finally adjusting the ROI_DEPTH once again to point below the printer head nozzle . This eliminates the possibilities of any false clusters , and also stabilises the distance values without intolerable flickering.

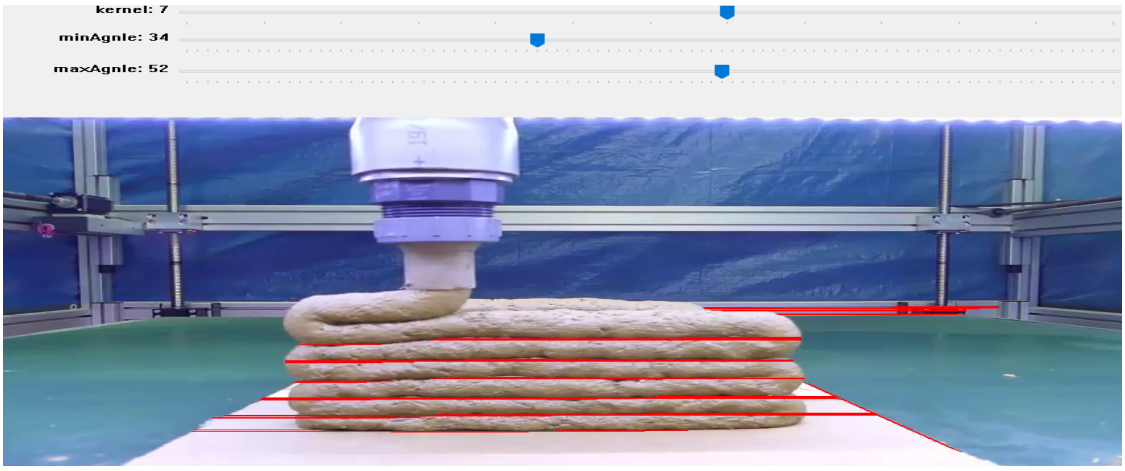


Figure17: Adjusting Kerne, minAngle and maxAngle values dynamically in video test.

Setting a timer

At this stage we set a timer as suggested by our supervisor, to track the change in various parameters including the height of the 3D printed moulds over time.

Discussion

Importance of light

For successful detection of the lines in the input image or in put video , it is very important to have sufficient light on the region of interest. Otherwise the morphological transformation and canny and HoughLineP algorithms may fail to generate enough lines for the experiment.
The source of light can be provided in two ways:

Physical light source

Providing a physical light source is possible and is easily achievable. A light source with adjustable brightness is also a good option to dynamically control the brightness along with the 3D printer.

Dynamically adjusting the light source

It's possible to control / adjust the brightness of the target image / video programmatically if the detection quality decreases. But this adds a significant overhead to the processing of each frame .

For example the following image was not not getting detected . So we changed its brightness :
And loaded the image again . We adjusted the ROI_DEPTH value to get the desired result.

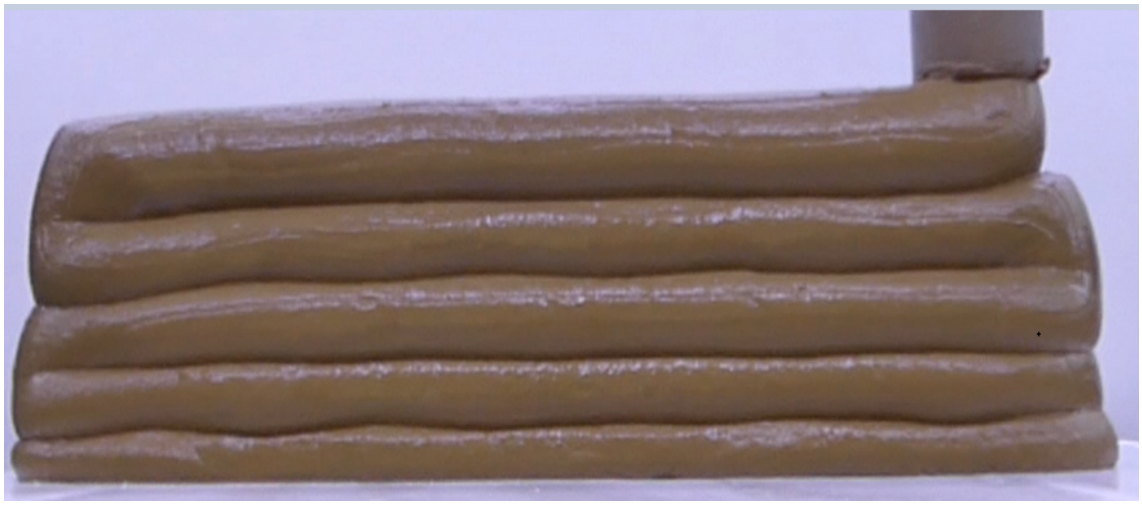


Figure18: Image with insufficient brightness or light source.

Figure 19: Corrected brightness manually using image editor.

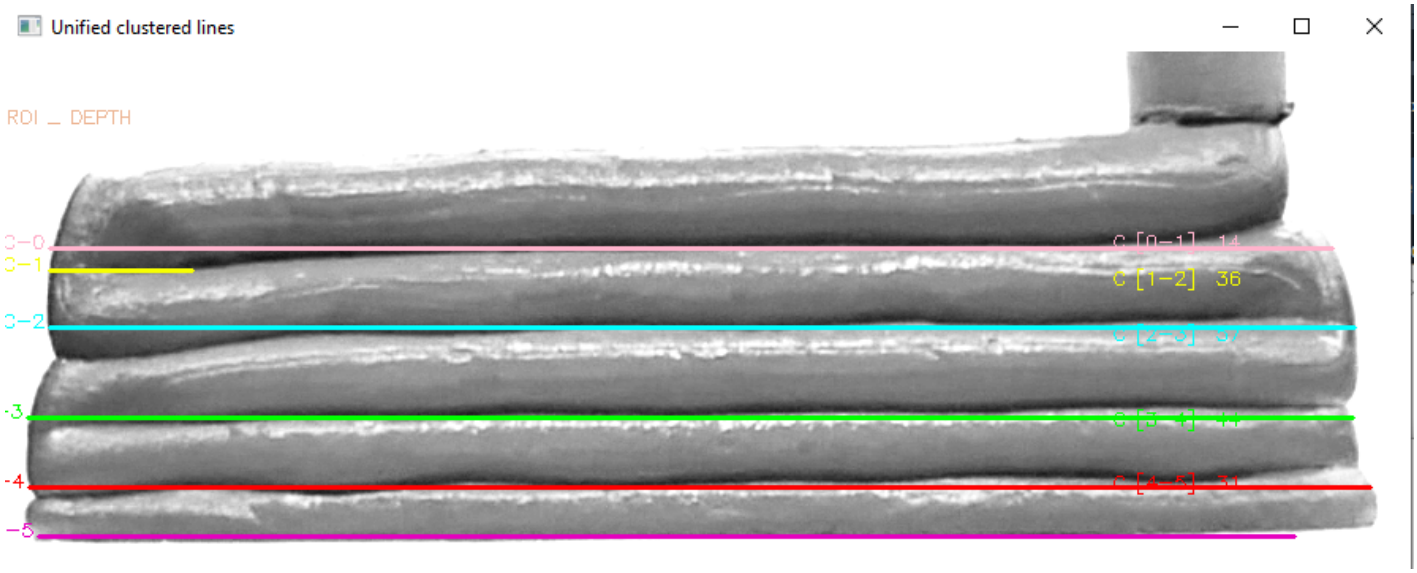


Figure20: desired result with distances.

Too many frames per second: As suggested by the supervisor , It's a good idea to apply the processing to a set of selective image frames in a second . This will decrease the overload on the CPU and increase overall throughput of the system.

Taming the flickering output of he clusters and their distance over time

It's a challenge to track the distance of a particular layer of printed mould . Because the detection of lines varies from frame to frame and thus the clusters keep changing .

So we need to try to decrease the fluctuation of reading. One way is to find parameters that make the result more stable. We need to modify the ROI value to point to a level just below the head so that it only covers the focus on the stable layers of printed moulds . Thus the stability of the measures are achieved although the image frames keep changing.

Conclusion

Although we faced difficulties getting desired results initially, we have achieved our desired results. As we implemented our own logic for clustering the lines we have now far more granular control on the detected lines . The fact of detecting lines depends on the image quality and appearance of sufficient light . It's sometimes possible to get better results with increasing the alpha , beta and gamma correction on the image programmatically but it adds extra overhead to the main process and thus it shows down the overall throughput. That is why it is highly recommended to keep sufficient light in the region of interest . Modifying the region of interest ROI position to a marginal point having a height just enough to allow passing the newly generating cluster will eliminate unwanted noise and also stabilise the display of the cluster to cluster distance measurement values which is very crucial for reaping benefits from the solution. We can calculate the unit of measurement for the comparison of the measurement of the head in image to the actual size of the head. Thus we can apply thie ratio to the obtained distance between two adjacent clusters and convert it to cm unit of measurement . After achieving the distance we can transmit the time series data to a Message Queue so that we can decouple the long running database taks from the image processing unit. The setup of a message queue can be done in an intranet network and thus no need for cloud servers. But in production we recommend having a replication of the server systems in the intranet if cloud servers are not available label yet. .

Perspective

How can we automate the of detection of the ROI

We can automate setting the ROI by detecting the printer head. Advanced version of CNN can be applied to detect the printer head in real time. Initially we need to collect various input images from different angles and lighting, distortions, different colours etc . We then need to create a bounding box and define the x,y coordinates of these bounding boxes containing heads in each image. This information is stored in a separate metadata file. We load images and metadata files to Google colab for free of cost training of our model with preferred version of CNN . Then download the trained model and use it to test out the model. Once we are satisfied with the result, we can use this model in our existing project and thus we can get the head detected in every image frame. This location of the head will be used to determine the ROI_DEPTH , for example the bottom line of the head is the ROI_DPTH which will change as the head moves.

Determining UOM from ROI

Once we find the distance we need a standard or reference in the image to convert the pixel distance to the actual unit of measurement like centimetre. This can be easily done in various ways. Once such

a way could be dependent on recognition of the head. Once we know the bounding box dimensions of the head, and we know the actual dimension of the head, we can establish a relationship of the ratio of the change and apply this formula to calculate the distance in centimetres.

Decoupled data transfer using Message Queue

We generate time series data along with cluster distances.
At last we print the time stamps in the format of HH:MM:SS:MSS where MSS is millisecond along with the cluster's information and the relevant distances to the log file for the time being . But in future we will spit this date asynchronously to a Message Queue and save it in the database automatically which intern will be used by machine learning algorithms to find out further insights like elasticity, viscosity, tolerance , effect of gravitational forces over increasing height and weights etc to determine and control the efficient output of the final printed construction units..

Detecting faults

Further analysis of the variation of the thickness of the detected clusters before unifying them could reveal the extent of distortions.

Handshaking with printer instructions on propagation

To make the whole system more intelligent a handshaking between the printer control unit Raspberry pi to this image recognition will increase the decision making and adjusting of ROI positions intelligently. And at the time of catastrophic failure the IOT device can be notified of the incidence so that it can be stopped and take necessary actions to save the waste and loss.

Appendix

List of figures

Bibliography

[1] https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

[2] https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

[3] https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

[4] https://constructionblog.autodesk.com/3d-printing-construction/

[5] https://en.wikipedia.org/wiki/Python_(programming_language)

[6] Digitalization in the construction industry (oliverwyman.com)

[7] ai for 3d printing_revised_accepted version.pdf (ntu.edu.sg)

[8] OpenCV: Canny Edge Detector

Code:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb  9 17:41:06 2022
Last edited : Feb 19 , 5:28 AM
@author: Mahbub islam



"""

import cv2
import numpy as np
import math
import operator
from time import time

global ROI_DEPTH,kernel_size,clusterDepth,minAngle,maxAngle , clusters
global  changedV, initV,minAngle,maxAngle

kernel_size =7#7#9#5
ROI_DEPTH = 200 #400 #410
clusterDepth=13 # 10,15,20,30,50 <-- check output
minAngle=38.0
maxAngle=48.0

clusters = {}
counter = 0
img = cv2.imread('lab3d2.png', cv2.IMREAD_GRAYSCALE)
uniqueClusterKey = - 1
drawnList = []

# Initialize output
out1 = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
out2 = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
out3 = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
gray = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)



def callMain():
    # main1(gray,out1)
    h=0


def findOverlap(minX1, maxX1, minX2, maxX2):
    overLappingX = - 1
```

```
    if minX1 <= minX2 <= maxX1:
        overLappingX=minX2
    elif minX2<=minX1<=maxX2:
        overLappingX=minX1
    elif minX1<=maxX2<=maxX1:
        overLappingX=maxX2
    elif minX2<=maxX1<=maxX2:
        overLappingX=maxX1


    return overLappingX

def printText(img,x,y,text,clusterId,fontWeight):
    position = (int(x),int(y))
    color=changeColor(clusterId)
    rgb=getRGBColor(color)
    cv2.putText( img,str(text), position,
                cv2.FONT_HERSHEY_SIMPLEX,  0.4, rgb, fontWeight)


def printInitParams():
    params='@@@@ clusterDepth= '+str(clusterDepth)+'  ROI_DEPTH= '+str(ROI_DEPTH)+'  '
    clusters={}

def getDistance(lines):
 lineArray = []
 for i in range(len(lines[0])):
     if i < (len(lines)-1):
         line1 = lines[i]
         line2 = lines[i+1]
         distance = getDistanceBetween2Lines(line1, line2)
         print(str(distance)+"= distance between "+line1+" and "+line2)


def getDistanceBetween2Lines(line1, line2):
    xDist = line1[0]-line2[0]
    yDist = line1[1]-line2[1]
    distance = math.sqrt(abs(xDist)*abs(xDist)+abs(yDist)*abs(yDist))
    return distance


def showLines(lines):
    counter = 0
    for a in range(0, (len(lines)), 2):

        line1 = lines[a]
        line2 = lines[a+1]
        x1 = line1[0][0]
        y1 = line1[0][1]
```

```
        x2 = line1[0][2]
        y2 = line1[0][3]
        counter += 1
        angle = np.rad2deg(np.arctan(y2/y1))
        if angle == 45:  # and angle <= 50:
            di = getDistanceBetween2Lines(line1, line2)
            cv2.line(out1, (x1, y1), (x2, y2), (0, 0, 255), 1)
            print(str(di)+"["+str(a)+","+str(a+1)+"]")


def findSlope(x1, y1, x2, y2):
    x = (x2 - x1)
    m = float("nan")
    if x != 0:
        m = (y2 - y1) / x
    return m


def findYIntercept(x1, y1, m):
    c = float("nan")
    if math.isnan(m) == False:
        c = y1-m*x1
    return c


def isAngleInRange(line1, low, high):

    y1 = line1[0][1]
    y2 = line1[0][3]
    if y1<ROI_DEPTH :
        return False
    angle = np.rad2deg(np.arctan(y2/y1))

    if angle >= low and angle <= high:
        return True
    return False


def detectLinesFromImage(gray):

    # Read input
    # img = cv2.imread('lab3d.png')

    # gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # out = gray
    blur_gray = cv2.GaussianBlur(gray, (kernel_size, kernel_size), 0)

    vertical_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1, 5))
```

```
    detected_lines = cv2.morphologyEx(
        blur_gray, cv2.MORPH_OPEN, vertical_kernel, iterations=2)

    edges = cv2.Canny(detected_lines, 0, 255)
    lines = cv2.HoughLinesP(edges, rho=0.25, theta=np.pi / 180, threshold=20,
                            lines=np.array([]), minLineLength=50,
                            maxLineGap=100)
    return lines

global linesWritten

def drawLinesOnImage(gray,lines,minAngle,maxAngle):
    global linesWritten
    linesWritten     = 0
    for line in lines:
        for x1, y1, x2, y2 in line:
            #print(line)
            angle = np.rad2deg(np.arctan(y2/y1))
            if angle >= minAngle and angle <= maxAngle and y2 > ROI_DEPTH:
             #   print(angle)
                cv2.line(gray, (x1, y1), (x2, y2), (0, 0, 255), 1)
                linesWritten += 1


    return linesWritten

def printLine(out,x1,y1,x2,y2,currentColor , lineThickness):

            # DRAW ROI_DEPTH
            printText(out, 4,ROI_DEPTH-5, 'ROI _ DEPTH', 100, 1)
            cv2.line(out, (4, ROI_DEPTH), (200, ROI_DEPTH), (255, 255, 255), 1)

            rgb=getRGBColor(currentColor)
            cv2.line(out, (x1, y1), (x2, y2), rgb, lineThickness)


def getRGBColor(currentColor):
    rgb=(0,0,0)
    if currentColor=="GREEN":
        rgb= (0, 255, 0)
    elif currentColor=="RED":
        rgb=  (0, 0,255)
    elif currentColor=="BLUE":
        rgb=(204,179,255)
    elif currentColor=="PERSIAN_GREEN":
         rgb= (191,0,230)#(0,179,149)#(121, 85, 72)
    elif currentColor=="YELLOW":
         rgb= (255,255,0)
    elif currentColor=="SKY":
```

```
        rgb=  (0 , 255,247)
    elif currentColor=="GOLDEN":
        rgb=  (255,137,0)
    elif currentColor=="MAGENTA":
        rgb=  ( 179,0,255)
    elif currentColor=="PINK":
        rgb=  (255 , 0, 145)
    elif currentColor=="TARQIZ":
        rgb= (0,154,255)
    elif currentColor=="LIGHT_GREEN":
        rgb= (0 , 255, 60)
    elif currentColor=="BROWN":
        rgb= ( 84, 3, 30)
    elif currentColor=="GREEN_2":
       rgb=  (66, 84, 3)
    elif currentColor=="WHITE":
        rgb= (5, 255, 237)
    return rgb

def drawEqualizedLineLength(lines,out,clusterList):
    cMinx =9999
    cMaxX=-1
    counter =0
    for v in clusterList:
        if counter==0:
            cMinX=v.minX
        if v.minX<cMinx:
            cMinX=v.minX
        if v.maxX>cMaxX :
            cMaxX=v.maxX
        counter+=1
    for k in clusters.keys():
        v=clusters[k]
        minX,maxX=getMinXMaxX(lines,      v.clusterIndexList)
        indexList=v.clusterIndexList
        currentColor=changeColor(v.clusterId)
        minY=min(v.yList)
        maxY=max(v.yList)
        meanY=math.floor((minY+maxY)/2)
        printLine(out,cMinX,meanY,cMaxX,meanY ,currentColor,2)



def drawSummaryLine(lines, out,clusterList):
    cMinx =9999
    cMaxX=-1
    counter =0

    for v in clusterList:
```

```
        if counter==0:
            cMinX=v.minX
        if v.minX<cMinx:
            cMinX=v.minX
        if v.maxX>cMaxX :
            cMaxX=v.maxX
        counter+=1
    # for k in clusters.keys():
        # v=clusters[k]
    cl=0
    for v in clusterList:
        minX,maxX=getMinXMaxX(lines,v.clusterIndexList)
        currentColor=changeColor(cl)
        minY=min(v.yList)
        maxY=max(v.yList)
        meanY=math.floor((minY+maxY)/2)
        # printLine(v.minX,meanY,v.maxX,meanY ,currentColor,3)
        # printLine(cMinX,meanY,cMaxX,meanY ,currentColor,1)
        # printLine(out,minX,meanY,maxX,meanY ,currentColor,2)
        printLine(out,minX,maxY,maxX,maxY ,currentColor,2)
        printText(out, minX-30, meanY, 'C-'+str(cl), cl,1)
        cl+=1



def printClusterDistance(lines, out, clusterList,tString):
    clusterCounter =0;
    distance=-1


    overlay = out.copy()
    cv2.rectangle(overlay, (650,0), (900,930), (0,0,0),-1);
    alpha = 0.4  # Transparency factor.
    cv2.addWeighted(out, alpha, overlay, 1 - alpha,0.0, out)

    cv2.putText( out,"Time: "+str(tString), (700,40),cv2.FONT_HERSHEY_SIMPLEX,  .5, (0, 255,0), 1)

    for v1 in clusterList:

        cLen=len(clusterList)
        if clusterCounter+1<cLen:
            minX1, maxX1 = getMinXMaxX(lines, v1.clusterIndexList)
            currentColor1 = changeColor(v1.clusterId)
            minY1 = min(v1.yList)
            maxY1 = max(v1.yList)
            meanY1 = math.floor((minY1 + maxY1) / 2)
            v2=clusterList[clusterCounter+1]
            minX2, maxX2 = getMinXMaxX(lines, v2.clusterIndexList)
            currentColor2 = changeColor(v2.clusterId)
```

```
            minY2 = min(v2.yList)
            maxY2 = max(v2.yList)
            meanY2 = math.floor((minY2 + maxY2) / 2)
            overLappingX=findOverlap(minX1, maxX1, minX2, maxX2)
            #DOC3: Draw distance only if the two closest adjacent
            #clusters have an overlapping point
            if overLappingX!=-1:
                # distance=abs(meanY1-meanY2)
                distance=abs(maxY1-maxY2)
                if clusterCounter==0:
                    y=1
                else:
                    y=clusterCounter+2
                printText(out, 700,
                          150+y*15,
                          # 30*y+1,
                          'C ['+str(clusterCounter)+'-'
                          +str(clusterCounter+1)+']  '+str(distance),
                          clusterCounter,1)
                print(str(getTimeHMS())+' '+'C ['+str(clusterCounter)+'-'
                +str(clusterCounter+1)+']  '+str(distance))
            # else:
            #DOC3 todo :
                # for each next adjacnet cluster,
                #find an overlap, if overlap found ,
                #calculate & draw distance


        clusterCounter+=1


def drawEachLine(lines,out,clusterList):
    global drawnList
    # for k in clusters.keys():
      # v=clusters[k]
    for v in clusterList:

      print('--- '+' '+str(v))
      indexList=v.clusterIndexList
      currentColor=changeColor(v.clusterId)

      minY=min(v.yList)
      maxY=max(v.yList)
      meanY=math.floor((minY+maxY)/2)

      for cli in indexList:
          if cli not in drawnList:
              line = lines[cli]
              drawnList.append(cli)
```

```
            for x1, y1, x2, y2 in line:
                drawnYList.append(y1)
                printLine(out,x1,y1,x2,y2,currentColor,1)

def getMinXMaxX(lines,indexList):
    xMin=0
    xMax=0
    for k in indexList:
        if k <len(lines):
            line =lines[k]
            for x1,y1,x2,y2 in line:
                if xMin==0:
                    xMin=x1
                elif x1<xMin:
                    xMin=x1
                if xMax==0:
                    xMax=x2
                elif x2>xMax:
                    xMax=x2

    return (xMin,xMax)



def changeColor(k):
    global currentColor, prevK, firstCall

    colorDict={
            0:"PERSIAN_GREEN",
            1:"GREEN",
            2:"RED",
            3:"BLUE",
            4:"YELLOW", # 255,255,0
            5:"SKY", # 0 , 255,247
            6:"GOLDEN", # 255,137,0
            7:"MAGENTA", # 179,0,255
            8:"PINK" , # 255 , 0, 145
            9:"TARQIZ",  # 0,154,255
            10:"LIGHT_GREEN", # 0 , 255, 60
            11:"BROWN" ,# 84, 3, 30
            12:"GREEN_2", # 66, 84, 3
            13:"PERSIAN_GREEN",
            14:"GREEN",
            15:"RED",
            16:"BLUE",
            17:"YELLOW", # 255,255,0
            18:"SKY", # 0 , 255,247
            19:"GOLDEN", # 255,137,0
```

```
20:"MAGENTA", # 179,0,255
21:"PINK" , # 255 , 0, 145
22:"TARQIZ",  # 0,154,255
23:"LIGHT_GREEN", # 0 , 255, 60
24:"BROWN" ,# 84, 3, 30
25:"GREEN_2", # 66, 84, 3
26:"GREEN_2", # 66, 84, 3
27:"PERSIAN_GREEN",
28:"GREEN",
29:"RED",
30:"BLUE",
31:"YELLOW", # 255,255,0
32:"SKY", # 0 , 255,247
33:"GOLDEN", # 255,137,0
34:"MAGENTA", # 179,0,255
35:"PINK" , # 255 , 0, 145
36:"TARQIZ",  # 0,154,255
37:"LIGHT_GREEN", # 0 , 255, 60
38:"BROWN" ,# 84, 3, 30
39:"GREEN_2", # 66, 84, 3
40:"PERSIAN_GREEN",
41:"GREEN",
42:"RED",
43:"BLUE",
44:"YELLOW", # 255,255,0
45:"SKY", # 0 , 255,247
46:"GOLDEN", # 255,137,0
47:"MAGENTA", # 179,0,255
48:"PINK" , # 255 , 0, 145
49:"TARQIZ",  # 0,154,255
50:"LIGHT_GREEN", # 0 , 255, 60
51:"BROWN" ,# 84, 3, 30
52:"GREEN_2", # 66, 84, 3
53:"BLUE",
54:"YELLOW", # 255,255,0
55:"SKY", # 0 , 255,247
56:"GOLDEN", # 255,137,0
57:"MAGENTA", # 179,0,255
58:"PINK" , # 255 , 0, 145
59:"TARQIZ",  # 0,154,255
60:"LIGHT_GREEN", # 0 , 255, 60
61:"GREEN",
62:"RED",
63:"BLUE",
64:"YELLOW", # 255,255,0
65:"SKY", # 0 , 255,247
66:"GOLDEN", # 255,137,0
67:"MAGENTA", # 179,0,255
68:"PINK" , # 255 , 0, 145
```

```
            69:"TARQIZ",  # 0,154,255
            70:"LIGHT_GREEN", # 0 , 255, 60
            71:"BROWN" ,# 84, 3, 30
            72:"GREEN_2", # 66, 84, 3
            73:"PERSIAN_GREEN",
            74:"GREEN",
            75:"RED",
            76:"BLUE",
            77:"YELLOW", # 255,255,0
            78:"SKY", # 0 , 255,247
            79:"GOLDEN", # 255,137,0
            80:"MAGENTA", # 179,0,255
            81:"PINK" , # 255 , 0, 145
            82:"TARQIZ",  # 0,154,255
            83:"LIGHT_GREEN", # 0 , 255, 60
            84:"BROWN" ,# 84, 3, 30
            85:"GREEN_2", # 66, 84, 3
            86:"GREEN_2", # 66, 84, 3
            87:"PERSIAN_GREEN",
            88:"GREEN",
            89:"RED",
            90:"BLUE",
            91:"YELLOW", # 255,255,0
            92:"SKY", # 0 , 255,247
            93:"GOLDEN", # 255,137,0
            94:"MAGENTA", # 179,0,255
            95:"PINK" , # 255 , 0, 145
            96:"TARQIZ",  # 0,154,255
            97:"LIGHT_GREEN", # 0 , 255, 60
            98:"BROWN" ,# 84, 3, 30
            99:"GREEN_2", # 66, 84, 3
            100:"PERSIAN_GREEN",

            }
    if k>=0 and k<=100:
        return colorDict[k]
    else :
        return "WHITE"

drawnYList=[]



def getMean(yList):
    yList=getDistictList(yList)
    tot = 0
    meanY=0
    for y in yList:
        tot+=y
```

```
    if len(yList)>0:
        meanY=tot/len(yList)
    return meanY


def copyDistictList(srcList, destList):
    for src in srcList:
        if src not in destList:
            destList.append(src)
    return destList


def addToDistictList(indexList, value):
    value=getSingleValue(value)
    indexList = getDistictList(indexList)
    if value not in indexList:
        indexList.append(value)
    return indexList

def getDistictList(yList):
    dest=[]
    for l in yList:
        if isinstance(l, list):
            for x in l:
                dest.append(x)
        else:
            dest.append(l)
    return dest

class DistInfo:

    clusterId=-1
    minX = 0
    maxX = 0
    yList = []
    meanY=0
    maxY=0
    clusterIndexList=[]

    def getClusterIndexList(self):
        return self.clusterIndexList

    def addToClusterIndexList(self, v):
        self.clusterIndexList = addToList(v, self.clusterIndexList)

    def __str__(self):

        strA = 'ClusterId= '+str(self.clusterId) +" minX="+str(self.minX)+", maxX= "+str(self.maxX)+\
            " Y-Mean= "+str(self.meanY)+"  maxY= "+str(self.maxY)+\
```

```
            " minY= "+str(np.min(self.yList))+" yList= "+str(self.yList)+\
            " , clusterIndexList= "+str(self.clusterIndexList)
        return strA


def decomposeList(lst):
    yList = []
    for li in lst:
        if isinstance(li, list):
            for it in li:
                if not it in yList:
                    yList.append(it)
        else:
            if li not in yList:
                yList.append(li)
    return yList



def findMean(list1):
    tot=0
    cnt=0
    for l in list1:

        if isinstance(l, list):
                for li in l:
                    tot+=li
                    cnt+=1
        else:
            tot+=l
            cnt+=1
    mean =np.nan
    if cnt!=0:
        mean=math.floor(tot/cnt)
    return mean

globalMinX=0
globalMaxX=0
def addSingleIndexToCluster(line1Y, index,line1):
    global uniqueClusterKey , clusters, globalMinX, globalMaxX
    indexAdded=False
    for key in clusters.keys():

        v = clusters[key]

        yList = decomposeList(v.yList)
        maxY=max(yList)
        minY=min(yList)
        clusterIndexList=v.clusterIndexList
```

```
        if index in clusterIndexList:
            return
                        if  (line1Y>=maxY  and     line1Y-minY<=clusterDepth  )or  (line1Y<maxY  and
maxY-line1Y<=clusterDepth):
            if index not in clusterIndexList:
                clusterIndexList=addToDistictList(clusterIndexList, index)
                v.clusterIndexList=clusterIndexList
                # ADD Y
                v.yList=addToDistictList(v.yList, line1Y)
                # CALC GLOBAL MinX
                x1=getSingleValue(line1[0][0])
                x2=getSingleValue(line1[0][2])
                if globalMinX==0:
                    globalMinX=x1
                elif x1 <globalMinX:
                    globalMinX=x1
                #ADD MinX
                if x1<v.minX:
                    v.minX=x1
                    # CALC GLOBAL MaxX
                if globalMaxX==0:
                    globalMaxX=x2
                elif x2 >globalMaxX:
                    globalMaxX=x2
                #ADD MaxX
                if x2>v.maxX:
                    v.maxX=x2
                #ADD meanY
                meanY=findMean(v.yList)
                v.meanY=meanY
                #ADD maxY
                v.maxY=max(yList)
                clusters[key]=v
                indexAdded=True
                addedIndex=index
                addedY=line1Y
                addedCluster=key
                break

    if indexAdded==False:
        # create a new Cluster and add the index to it
        minX=getSingleValue(line1[0][0])
        maxX=getSingleValue(line1[0][2])
        yList=[line1Y]
        newDistInfo = DistInfo()
        newDistInfo.minX=minX
        newDistInfo.maxX=maxX
        newDistInfo.clusterIndexList=[index]
        newDistInfo.yList=[getSingleValue(line1Y)]
```

```
        #ADD maxY
        newDistInfo.maxY=max(yList)
        uniqueClusterKey+=1
        newDistInfo.clusterId=uniqueClusterKey
        clusters[uniqueClusterKey]=newDistInfo

        print("\n\n==== added new Cluster = "+str(uniqueClusterKey)\
            +" index = "+str(index)\
                +" lin1Y= "+str(line1Y) \
                    + " new yList= "+str(yList)\
                        +" new indexList= "+str(newDistInfo.clusterIndexList))




def addToExistigCluster(minX, maxX, k, h, line1Y, line2Y, line1,line2):

    linY1=getSingleValue(line1Y)
    linY2=getSingleValue(line2Y)
    addSingleIndexToCluster(linY1, k,line1)
    addSingleIndexToCluster(linY2, h,line2)

def addToList(v, lst):
    l = len(lst)
    if isinstance(v, list):
        lst.extend(decomposeList(v))
    else:
        lst.append(v)

    return lst

def processClustering(lines,minAngle, maxAngle, clusterDepth):
    global clusters
    clusters={}
    for k in range(len(lines)-1):
        line1 = lines[k]
        if isAngleInRange(line1, minAngle, maxAngle) and line1[0][1] > ROI_DEPTH:
            c1 = np.nan # type(np.nan)=> float
            m1 = findSlope(line1[0][0], line1[0][1], line1[0][2], line1[0][3])

            if math.isnan(m1) == False:
                c1 = findYIntercept(line1[0][0], line1[0][1], m1)

            if not math.isnan(c1):
                for h in range(k+1, len(lines)-1):

                    line2 = lines[h]
                    if line2[0][3] > ROI_DEPTH:
                        m2 = findSlope(line2[0][0], line2[0][1],
```

```
                        line2[0][2], line2[0][3])
                if not math.isnan(m1) and not math.isnan(m2):
                    floorM1 = math.floor(m1)
                    floorM2 = math.floor(m2)
                    c2 = findYIntercept(line2[0][0], line2[0][1], m2)
                    minDist = abs(c2-c1)/math.sqrt(1+m1*m1)
                    if minDist<0:
                        minDist=0
                    if minDist <= clusterDepth:
                        minX = line1[0][0] if line1[0][0] < line2[0][0] else line2[0][0]
                        maxX = line1[0][2] if line1[0][2] > line2[0][2] else line2[0][2]
                        addToExistigCluster(minX, maxX, k, h,
                                getSingleValue(line1[0][3]), getSingleValue(line2[0][3]), line1,line2)


def getSingleValue(valOrList):
    retVal=-1
    if isinstance(valOrList,list):
        retVal=valOrList.pop(0)
    else:
        retVal=valOrList
    return retVal


def drawCluster(out1,lines,tString):
    global  drawnList , clusters

    clusterList=sorted(clusters.values(),key=operator.attrgetter('maxY'))

    drawSummaryLine(lines,out1,clusterList)
    # drawEachLine(lines,out2,clusterList)
    drawEqualizedLineLength(lines,out3,clusterList)
    printClusterDistance(lines, out1, clusterList,tString)


def main1(gray, out1,tString):
    global  drawnList , clusters

    printInitParams()
    lines = detectLinesFromImage(gray)

    processClustering(lines,minAngle, maxAngle, clusterDepth)
    totalLineCount=drawLinesOnImage(gray,lines,minAngle, maxAngle)
    drawCluster(out1,lines,tString)
    clusterSize=len(clusters.keys())


    overlay = out1.copy()
```

```
    cv2.rectangle(overlay, (0,0), (170,230), (0,0,0),-1);

    alpha = 0.5  # Transparency factor.

    # Following line overlays transparent rectangle over the image
    cv2.addWeighted(out1, alpha, overlay, 1 - alpha,0.0, out1)

    printText(out1, 4,25, "lines detected: "+str(lines.size), 2, 1)
    printText(out1, 4,40, "clusterSize: "+str(clusterSize), 4, 1)
    #printText(out1, 4,100, "Time: "+str(tString), 100, 1)
    printText(out1, 4, 70, "ClusterDepth: " + str(clusterDepth), 100, 1)
    printText(out1, 4,95, "MinAngle: "+str(minAngle), 100, 1)
    printText(out1, 4,115, "MaxAngle: "+str(maxAngle), 100, 1)

    printText(out1, 4,130, "kernel_size: "+str(kernel_size), 4, 1)
    printText(out1, 4,150, "totalLineCount: "+str(totalLineCount), 5, 1)
    printText(out1, 4,175, "Frame Per Second: "+str(framePerSecond), 6, 1)
    printText(out1, 4,195, "FrameCount: "+str(frameCount), 6, 1)

        # print('  EndSecond=  '+str(endSecond)+'  startSecond=  '+str(startSecond)+'  framePerSecon=
'+str(frameCount))




    printText(out1, 4,ROI_DEPTH-5, 'ROI _ DEPTH', 100, 1)
    cv2.line(out1, (4, ROI_DEPTH), (200, ROI_DEPTH), (255, 255, 255), 1)



    # print("====== num of lines detected  ="+str(lines.size))
    # print(" num of lines written="+str(len(drawnList)))
    # print(lines[0])
    # print(lines[len(lines)-1])
    # for kc in clusters.keys():
    #     distInfo = clusters[kc]
    #     print(distInfo)
   # cv2.imshow('Unified clustered lines', out1)
    # cv2.imshow('Clustered Lines', out2)
    # cv2.imshow('Lines read ', gray)
    # cv2.imshow('Equalized line length+ Unified lines ', out3)
    print("DrawnIndexList= "+str(sorted(drawnYList)))

    #cv2.waitKey(0)
    #cv2.destroyAllWindows()


callMain()
```

```
global mainImage

def setInitV(img):
    global initV
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)
    initV=v

def increase_brightness(img, value=30):
    global initV,changedV
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)
    if initV==-999999:
        initV=v

    lim = 255 - value
    v[v > lim] = 255
    v[v <= lim] += value

    final_hsv = cv2.merge((h, s, v))
    changedV=final_hsv
    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    return img
changedV=0

def onChangeV(v):
    global changedV
    changedV=cv2.getTrackbarPos('V', 'wqualized')
    increase_brightness(mainImage,changedV)

    # todo:  Fix HSV to adjust brightness


def onChangeROI(v):
    global ROI_DEPTH
    ROI_DEPTH=cv2.getTrackbarPos('ROI_DEPTH', 'Frame')

def onChangeMinAngle(v):
    global minAngle
    minAngle=cv2.getTrackbarPos('minAgnle', 'gray')

def onChangeMaxAngle(v):
    global maxAngle
    maxAngle=cv2.getTrackbarPos('maxAgnle', 'gray')

def onChangeClusterDepth(v):
    global clusterDepth
    clusterDepth=cv2.getTrackbarPos('clusterDepth', 'wqualized')
```

```
def onChangeKernel(v):
    global kernel_size


    tempKernel=cv2.getTrackbarPos('kernel', 'gray')

    if  tempKernel%2==0:
        tempKernel+=1
        cv2.setTrackbarPos('kernel', 'gray', tempKernel)
        kernel_size=tempKernel
    else:
        kernel_size=tempKernel

def getTimeHMS():
    ms=int((time())*divisor)-t
    mss=int(ms%divisor/1000)
    s = (int) (ms // divisor) % 60 ;
    m = (int) ((ms // (divisor*60)) % 60);
    h   = (int) ((ms // (divisor*60*60)) % 24);
    return (h,m,s,mss)


cap = cv2.VideoCapture(
    #'lab3d.mp4'
    'lab3d_converted.mp4'
    # 'coverModified.mp4'
    ,cv2.IMREAD_GRAYSCALE)

divisor=1000000
t=(int)(time()*divisor)


cv2.imshow('gray', out1)
cv2.imshow('wqualized', out1)
cv2.imshow('Frame', out1)

cv2.createTrackbar('kernel', 'gray', 1,12,onChangeKernel)
cv2.createTrackbar('minAgnle', 'gray', 1,90,onChangeMinAngle)
cv2.createTrackbar('maxAgnle', 'gray', 35,90,onChangeMaxAngle)
cv2.createTrackbar('clusterDepth', 'wqualized', 3,40,onChangeClusterDepth)
# cv2.createTrackbar('V', 'wqualized', 0,900,onChangeV)
cv2.createTrackbar('ROI_DEPTH', 'Frame', 0,900,onChangeROI)

# cv2.createTrackbar('kernel', 'gray', 1,12,onChangeKernel)
global framePerSecond, startSecond , endSecond, frameCount
framePerSecond=0

startSecond=0
```

```
endSecond=0
frameCount=0
processFPS=0
while (True):
    if cap.isOpened():


        ret, frame =cap.read( )

        frameCount=frameCount+1;
        if ret==True:
            # frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0, interpolation = cv2.INTER_CUBIC)
            #gray1 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            processFPS+=1
            # out1 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            uniqueClusterKey=0
            clusters=0
            globalMinX=0
            globalMaxX=0

            h,m,s,mss=getTimeHMS()
            if startSecond==0:
                startSecond=s
            endSecond=s
            if endSecond-startSecond==1:
                startSecond=s
                framePerSecond=frameCount
                frameCount=1

            # if frameCount<5:

            tString=str(h)+":"+str(m)+":"+str(s)+":"+str(mss)
            mainImage=frame.copy()
            out3=frame.copy()
            # setInitV(mainImage)

            main1(mainImage,frame,tString)
            # printText(frame, 4,95, "initV: "+str(initV), 100, 1)
            # printText(out3, 4,105, "changedV: "+str(changedV), 100, 1)



            cv2.imshow('Frame', frame)
            cv2.imshow('gray', mainImage)
            cv2.imshow('wqualized', out3)

            # out2=cv2.cvtColor(frame,cv2.COLOR_GRAY2BGR)
            # out3=cv2.cvtColor(frame,cv2.COLOR_GRAY2BGR)
            # gray=cv2.cvtColor(frame,cv2.COLOR_GRAY2BGR)
```

```
        # if cv2.waitKey(35) & 0xFF == ord('q'):
        #         break
        key = cv2.waitKey(22)

        if key == 32:
            cv2.waitKey()
        elif key == ord('q') or not cap.isOpened():
            break
# release the video capture object
cap.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```