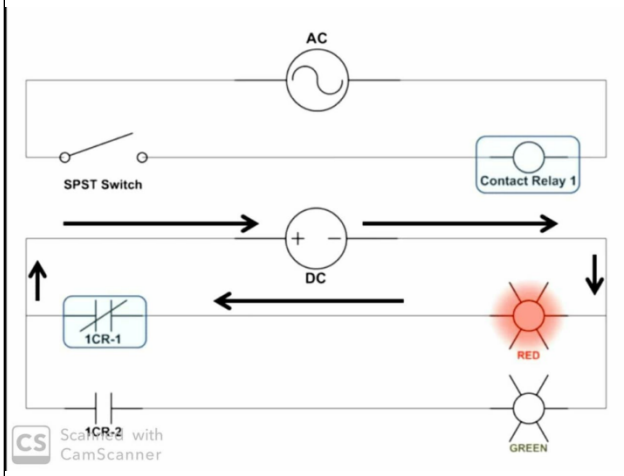
DAY 11 REPORT

Date:	30-May-2020	Name:	Ankitha c c
Course:	Logic Design	USN:	4AL16 EC004
Topic:	1)Application of Programmable logic controller	Semester & Section:	8th sem & 'A' section
Github Repository:	ankitha-course		

MORNING SESSION
DETAILS

Image of session



A programmable logic controller, or PLC, is a computer with a microprocessor used for industrial automation that can automate a specific process, machine function, or an entire production line.

A PLC is an electronic device used in many industries to monitor and control building systems and production processes. It is designed to perform a single set of tasks, except under real-time constraints and with superior reliability and performance.
To meet the demands of harsh industrial environments, PLCs are designed to be robust, often capable of withstanding extreme temperatures, humidity, vibration, and electrical noise. Logic controllers are commonly tasked with monitoring and controlling a very large number of sensors and actuators, and are therefore distinct from other computer systems in their extensive input/output (I/O) arrangements.
The PLC receives information from connected sensors or input devices, processes the data, and triggers outputs based on pre-programmed parameters.
Depending on the inputs and outputs, a PLC can monitor and record run-time data such as machine productivity or operating temperature, automatically start and stop processes, generate alarms if a machine malfunctions, and more. PLCs are a flexible and robust control solution, adaptable to almost any application.
There are several key features that set PLCs apart from industrial PCs, microcontrollers, and other industrial control solutions:

I/O – The PLC's CPU stores and processes program data, but input and output modules connect the PLC to the rest of the machine; these I/O modules are what provide information to the CPU and trigger specific results. I/O can be either analogue or digital; input devices might include sensors, switches, and meters, while outputs might include relays, lights, valves, and drives. Users can mix and match a PLC's I/O in order to get the right configuration for their application.

Communications – In addition to input and output devices, a PLC might also need to connect with other kinds of systems; for example, users might want to export application data recorded by the PLC to a supervisory control and data acquisition (SCADA) system, which monitors multiple connected devices. PLCs offer a range of ports and communication protocols to ensure that the PLC can communicate with these other systems.

Human Machine Interface (HMI) – In order to interact with the PLC in real time, users need an HMI. These operator interfaces can be simple displays, with a text-readout and keypad, or large touchscreen panels more similar to consumer electronics, but either way, they enable users to review and input information to the PLC in real time.
PLCs are used for continuously monitoring the input values from sensors and produces the outputs for the operation of actuators based on the program. Every PLC system comprises these three modules:

CPU Module

A CPU module consists of central processor and its memory. The processor is responsible for performing all the necessary computations and processing of data by accepting the inputs and producing the appropriate outputs.

Power Supply Module

This module supplies the required power to the whole system by converting the available AC power to DC power required for the CPU and I/O modules. The 5V DC output drives the computer circuitry.

I/O Modules

The input and out modules of the programmable logic controller are used to connect the sensors and actuators to the system to sense the various parameters such as temperature, pressure and flow, etc. These I/O modules are of two types: digital or analogue.

Communication Interface Modules

These are intelligent I/O modules which transfers the information between a CPU and communication network. These communication modules are used for communicating with other PLC's and computers, which are placed at remote place or far-off

locate.

The program in the CPU of programmable logic controller consists of operating system and user programs. The purpose of the operating system with CPU is to deal with the tasks and operations of the PLC such as starting and stopping operations, storage area and communication management, etc. A user program is used by the user for finishing and controlling the tasks in automation.

Programming A PLC

In these modern times, a PC with specially dedicated software from the PLC manufacturer is used to program a PLC. The most widely used form of programming is called ladder logic. Ladder logic uses symbols, instead of words, to emulate the real world relay logic control. These symbols are interconnected by lines to indicate the flow of current through relay like contacts and coils. Over the years the number of symbols has increased to provide a high level of functionality.

The completed program looks like a ladder but in actuality it represents an electrical circuit. The left and right rails indicate the positive and ground of a power supply. The rungs represent the wiring between the different components which in the case of a PLC are all in the virtual world of the CPU. So if you can understand how basic electrical circuits work then you can understand ladder logic.

Today, a number of different programming languages are used, but each PLC supplier has their own programming specifications based on the IEC 61131-3 standard. Although they have roughly the same sort of components found in many other computer systems, PLCs operate quite differently. A PLC operating cycle, or scan, consists of:

- Reading and storing the current value of each input,
- Changing all physical outputs to match the output table values stored in data memory,
- Sequentially executing the instructions in program memory, while storing any updated variables or outputs to data memory.

The order in which these tasks are performed can vary from product to product and software to software, but each task is performed in sequence. This means that the inputs, outputs, and program instructions are, in a sense, isolated from each other. If one or more physical inputs change during the logic scan, the input value table will not be changed until the next scan cycle, and so the program logic will not be invalidated. Similarly, the output channels will not be updated to match the output table until the program cycle is complete. These scans are made quite rapidly, often taking just microseconds to complete.

While most commonly used for industrial manufacturing processes, the unique strengths of PLCs can make them viable for many different applications. A building automation system (BAS) may use a PLC (instead of some other controller type) for a substantial boost in processing power and system response. A PLC used in building automation may have a variety of sensors as inputs, ranging from a simple limit switch to a mesh network of airflow sensors. Various types of I/O ports let building operations and maintenance personnel interface with the PLCs to monitor and adjust their behaviours from a central location, typically referred to as a SCADA system.

PLC In Automation

PLCs have helped in improving the productivity of automation, such as by lowering the amount of power consumed by working machines, controlling systems via proper keeping of records and reducing required manpower via the supply of manpower. The PLCs have also helped lower automation maintenance. If a business uses trailing cables for operating the automated storage and recovery systems, this will cause time wastage and higher costs. The higher cost is due to the fact that these cables will need frequent maintenance and replacement. Applying PLCs in automating these systems lowers maintenance costs and reduces needless downtime. PLCs have been effective in reducing automation downtime. Due to its capabilities, PLCs are becoming more exclusive than conventional personal computers and workstation arenas. The PLCs are now able to share data rapidly within and between companies. This rapid data sharing has helped reduce downtime as PLCs are capable of automating the FTP and web servers, international databases and even email sending.

How programmable logic controllers work

Each PLC system has three modules namely: CPU module, power supply module and one or more input/output (I/O) module.

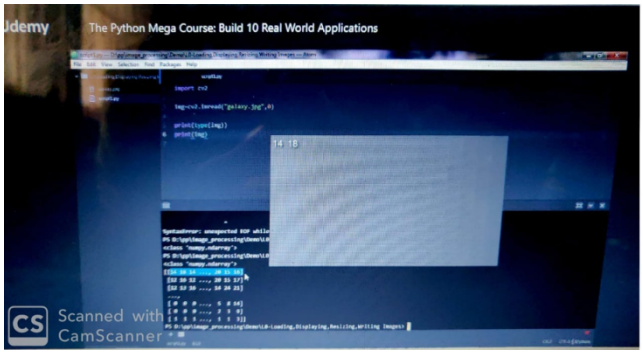
- **CPU Module**
  This module is comprised of a central processor and its memory component. This processor performs all the needed data computations and processing by receiving inputs and producing corresponding outputs.
- **Power supply module**
  PLC's computer circuitry runs on a 5V DC output and this is supplied by the power supply module. This is essentially the module responsible for powering up the system.
  It receives AC power and converts it to DC power that the two other modules (CPU and input/output modules) use.
- **I/O Modules**
  The input/output modules are responsible for connecting the sensors and actuators to the PLC system to sense the different parameters such as pressure, temperature, and flow.
  The I/O modules can be digital or analogue.

DAY 11 report

Date:	30-5-2020	Name:	Ankitha c c
Course:	Python programming	USN:	4AL16EC004
Topic:	1.Python for image and video processing using openCV	Semester & Section:	8th A
Github Repository:	Ankitha-course		

AFTERNOON SESSION DETAILS

Image of session



Processing of images & video using OpenCV

Processing a video means, performing operations on the video frame by frame. Frames are nothing but just the particular instance of the video in a single point of time. We may have multiple frames even in a single second. Frames can be treated as similar to an image.

So, whatever operations we can perform on images can be performed on frames as well. Let us see some of the operations with examples.

Adaptive Threshold –

By using this technique we can apply thresholding on small regions of the frame. So the collective value will be different for the whole frame.

```
# importing the necessary libraries
import cv2
```

```
import numpy as np

# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')


# Loop untill the end of the video
while (cap.isOpened()):

        # Capture frame-by-frame
        ret, frame = cap.read()
        frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
        interpolation = cv2.INTER_CUBIC)
        # Display the resulting frame
        cv2.imshow('Frame', frame)
        # conversion of BGR to grayscale is necessary to apply this operation
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # adaptive thresholding to use different threshold
        # values on different regions of the frame.
        Thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
        cv2.THRESH_BINARY_INV, 11, 2)
        cv2.imshow('Thresh', Thresh)
        # define q as the exit button
        if cv2.waitKey(25) & 0xFF == ord('q'):
                break
# release the video capture object
cap.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

Output:

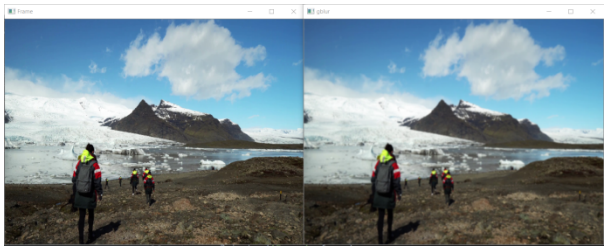
Smoothing –

Smoothing a video means removing the sharpness of the video and providing a blurriness to the video. There are various methods for smoothing such as cv2.Gaussianblur(), cv2.medianBlur(), cv2.bilateralFilter(). For our purpose, we are going to use cv2.Gaussianblur().

```
filter_none
brightness_4
# importing the necessary libraries
import cv2
import numpy as np

# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')


# Loop untill the end of the video
while (cap.isOpened()):
```

```
    # Capture frame-by-frame
    ret, frame = cap.read()
    frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                       interpolation = cv2.INTER_CUBIC)

    # Display the resulting frame
    cv2.imshow('Frame', frame)

    # using cv2.Gausssianblur() method to blur the video

    # (5, 5) is the kernel size for blurring.
    gaussianblur = cv2.GaussianBlur(frame, (5, 5), 0)
    cv2.imshow('gblur', gaussianblur)

    # define q as the exit button
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

# release the video capture object
cap.release()

# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

Output:



Edge Detection –

Edge detection is a useful technique to detect he edges of surfaces and objects in the video. Edge detection involves the following steps:

- Noise reduction
- Gradient calculation
- Non-maximum suppression
- Double threshold
- Edge tracking by hysteresis

```
filter_none
brightness_4
# importing the necessary libraries
import cv2
import numpy as np

# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')
```

```
# Loop untill the end of the video
while (cap.isOpened()):
    # Capture frame-by-frame
    ret, frame = cap.read()

    frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                       interpolation = cv2.INTER_CUBIC)

    # Display the resulting frame
    cv2.imshow('Frame', frame)

    # using cv2.Canny() for edge detection.
    edge_detect = cv2.Canny(frame, 100, 200)
    cv2.imshow('Edge detect', edge_detect)

    # define q as the exit button
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

# release the video capture object
cap.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

Output:

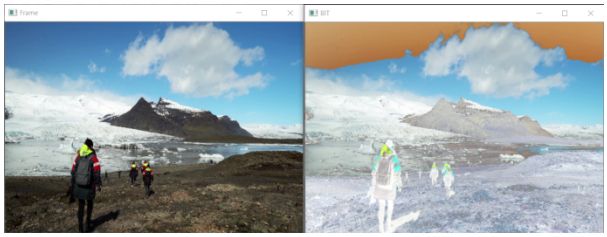
Bitwise Operations –

Bitwise operations are useful to mask different frames of a video together. Bitwise operations are just like we have studied in the classroom such as AND, OR, NOT, XOR.

```
filter_none
brightness_4
# importing the necessary libraries
import cv2
import numpy as np

# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')


# Loop untill the end of the video
while (cap.isOpened()):
    # Capture frame-by-frame
    ret, frame = cap.read()
    frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
```

```
                    interpolation = cv2.INTER_CUBIC)

    # Display the resulting frame
    cv2.imshow('Frame', frame)

    # conversion of BGR to grayscale is necessary to apply this operation
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    _, mask = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # apply NOT operation on image and mask generated by thresholding
    BIT = cv2.bitwise_not(frame, frame, mask = mask)
    cv2.imshow('BIT', BIT)

    # define q as the exit button
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

# release the video capture object
cap.release()

# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

Output:



Batch Image Resizing (Practice)

Write a script that resizes all images in a directory to 100x100. You can find an attached ZIP file with some image files in the *Resources*.

Solution

```
import cv2
import glob

images=glob.glob("*.jpg")

for image in images:
    img=cv2.imread(image,0)
    re=cv2.resize(img,(100,100))
    cv2.imshow("Hey",re)
    cv2.waitKey(500)
    cv2.destroyAllWindows()
    cv2.imwrite("resized_"+image,re)
```

I first created a list containing the image file paths and then iterated through the aformentoned list.

The loop: reads each image, resizes, and displays the image; waits for the user input key, closes the window once the key is pressed, and writes the resized image. The name of the resized image will be "resized" plus the existing file name of the original image.