**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **30-5-2020** | **Name:** | **Gaganashree P** |
| **Course:** | **LOGIC DESIGN** | **USN:** | **4AL15EC024** |
| **Topic:** | **1)Application of Programmable logic controller** | **Semester & Section:** | **8th sem ‘A’ sec** |
| **Github Repository:** | **Gaganashree-P** |  |  |

|  |  |
| --- | --- |
| **FORENOON SESSION DETAILS** | |
| **Image of session**          **Report:**     * 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](https://www.iaasiaonline.com/what-a-plc-upgrade-can-do-for-you-2/) 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](https://www.iaasiaonline.com/what-a-plc-upgrade-can-do-for-you-2/) 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.  **Programming A PLC**  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.   **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. 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. | |
| |  |  |  |  | | --- | --- | --- | --- | | **Date:** | **30-5-2020** | **Name:** | **Gaganashree P** | | **Course:** | **Udemy** | **USN:** | **4AL15EC024** | | **Topic:** | **Python for image and video processing using open CV** | **Semester & Section:** | **8th A** | | **Github Repository:** | **Gaganashree-P** |  |  | |
|  |
|  |
| **AFTERNOON SESSION DETAILS** | |
| **Image of session:** | |
| **Report –** 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.  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()  **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  0, fy 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,                           interpolation = cv2.INTER\_CUBIC)        # Display the resulting frame      cv2.imshow('Frame', frame)        # using cv2.Gaussianblur() 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:**  https://media.geeksforgeeks.org/wp-content/uploads/20200126121606/gblur.png  **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:**  https://media.geeksforgeeks.org/wp-content/uploads/20200126122343/edge3.png  **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:**  https://media.geeksforgeeks.org/wp-content/uploads/20200126123904/BIT1.png  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) | |