Gyanmanjari Innovative University

Syllabus
Gyanmanjari Institute of Technology
BTech. Semester-5

Subject: Computer Vision: Image Processing and AI-Based Recognition - BET1CE15319

Type of course: Professional Core

Prerequisite: Basic Python Syntax, Loops, Lists, and Functions Programming fundamental, logic and problem-solving skill, Mathematical logic.

Rationale: Computer Vision (CV) enables machines to extract meaningful data from digital images and live video streams. Moving to a Python ecosystem leverages industry-standard, high-efficiency tools like OpenCV (cv2), NumPy, and Pillow (PIL). This setup simplifies complex matrix calculations into short, intuitive lines of code, allowing students to focus on solving real-world tracking, filtering, and automation problems directly on their laptops.

Teaching and Examination Scheme:

Teaching Scheme			Credits	Examination Marks		Total Marks
CI	T	P	C	SEE	CCE	
4	0	2	5	100	50	150

Legends: CI-Class Room Instructions; T – Tutorial; P - Practical; C – Credit; SEE - Semester End Evaluation; CCE-Continuous and Comprehensive Evaluation.

GYANMANJARI INNOVATIVE UNIVERSITY BHAVNAGAR

Course Content:

Sr. No	Course Content	Hrs.	% Weightage
1	**Introduction to Computer Vision and Digital Image Fundamentals** ***Theory Topics:*** Introduction to Computer Vision applications; Digital image structures as NumPy 2D/3D matrix arrays; Coordinate systems and image dimensions; Pixel storage architectures (BGR vs. RGB); Color space translations (Grayscale, HSV, and Channel splitting); File read/write safety buffers and thresholding mathematics. ***Practical:*** 1. Read an image file from your laptop drive using cv2.imread() and display it on screen. 2. Write a script to print image matrix dimensions (Width, Height, Channels) using the .shape attribute. 3. Save a processed copy of an image to a specific desktop folder using cv2.imwrite(). 4. Isolate the **Blue** color matrix layer from a BGR image using standard NumPy slicing arrays (img[:, :, 0]). 5. Isolate the **Green** color matrix layer from an image array using slicing parameters (img[:, :, 1]). 6. Isolate the **Red** color matrix layer from an image array using slicing parameters (img[:, :, 2]). 7. Convert a bright, full-color graphic image file directly into an 8-bit Grayscale layout via cv2.cvtColor(). 8. Convert a standard input image matrix into an HSV (Hue, Saturation, Value) format array. 9. Code a static threshold filter to turn a grayscale image into a pure black-and-white binary image. 10. Create a script that swaps pixel color bands to convert an image from BGR to traditional RGB format. 11. Create an inverse copy (negative) of a target photograph using Python's bitwise NOT operator (cv2.bitwise_not()). 12. Build an empty, zero-filled pure black canvas array of a specific pixel size using np.zeros(). 13. Program an interactive color square canvas by assigning numeric color values directly to coordinate blocks.	18	20%

Evaluation Method:

Sr. No.	Evolution Methods	SEE	CCE
1	**Error Analysis and Correction:** Students will solve four faculty-assigned programming problems to assess their debugging, error-correction, and problem-solving skills.	20	
2	**Program Refinement:** Students will optimize two programs to improve efficiency, reduce complexity, and maintain correct functionality.		10
	Total	20	10

Examination Style:

Error Analysis and Correction (20 Marks)
In this assessment component, students will be presented with four distinct programming problems, which may contain either logical inconsistencies or syntactical inaccuracies. The nature of the questions will be unsolved, practical, or oriented toward competitive programming paradigms, and the problems will be selected and provided by the faculty member. This task is explicitly structured to evaluate and gauge the students' proficiency in debugging techniques, their capacity to identify and rectify errors within code, as well as their broader problem-solving acumen when confronted with non-trivial programming scenarios.

Program Refinement (10 Marks)
In this evaluation component, students will be given two distinct programs, which functionally produce correct outputs but are suboptimal in terms of algorithmic design, execution efficiency, or code verbosity. Students are required to refactor and enhance the underlying logic, reduce the time complexity, and minimize the number of lines of code wherever feasible, while strictly preserving the original functionality and expected behavior of the program. The questions will be of a practical, implementation-oriented nature and will be specifically curated by the faculty to assess the students' ability to write well-structured, efficient, and concise code that adheres to good programming practices and optimization principles.

2	**Foundations of Image Processing and Camera Modeling** ***Theory Topics:*** Matrix slicing and Region of Interest (ROI) selection; Camera projection geometry and calibration principles; Intrinsic and extrinsic parameters; Radial lens distortion corrections; Motion representation math for rigid objects; Image brightness constancy equations and motion parallax cues; Histogram calculation and contrast adjustments. ***Practical:***	18	20%

14. Crop a specific sub-region (Region of Interest) out of an image matrix using simple NumPy index slicing tricks.
15. Brighten an image safely using NumPy addition alongside np.clip() to keep pixel values between 0 and 255.
16. Darken a photo matrix using scalar subtraction while protecting against negative value wrapping errors.
17. Create a 256-bin pixel distribution chart of an image using cv2.calcHist().
18. Boost the visibility of low-light photos using global Histogram Equalization routines (cv2.equalizeHist()).
19. Generate a simulated set of 2D/3D coordinate arrays to define a calibration pattern space.
20. Detect checkerboard grid corners inside validation images using cv2.findChessboardCorners().
21. Compute intrinsic camera matrices and lens distortion vectors using cv2.calibrateCamera().
22. Fix radial lens curvature artifacts from a raw laptop camera capture by running cv2.undistort().
23. Compute the transformation matrices required to capture external rigid rotations.
24. Flip an image array horizontally using the cv2.flip() command with a positive direction flag.
25. Flip an image vertically by modifying row positioning arrays inside the runtime pipeline.
26. Build a program that calculates differences between consecutive frame matrices to estimate motion parallax cues.

Evaluation Method:

Sr. No.	Evolution Methods	SEE	CCE
1	**Proficiency-building Task:** The practical examination will consist of four programming questions. Students must write, execute, and submit the program with the corresponding output.	20	
2	**Syntax and Logic Error Detection Challenge:** Students will be given two faulty programs to identify, analyse, and correct errors. Evaluation will be based on debugging skills, problem-solving ability, and successful program execution.		10
	Total	20	10

	Examination Style: **Proficiency-building Task (20 Marks)**: The practical examination will consist of four questions based on the practical implementation of concepts covered in the respective unit. Students must write the complete program, execute it successfully, and submit the code along with the appropriate output. The programs should demonstrate correct logic, proper syntax, and accurate execution results. All questions will be prepared and assigned by the faculty members to evaluate the students' practical knowledge, problem-solving ability, programming accuracy, and understanding of coding concepts. **Syntax and Logic Error Detection Challenge (10 Marks)**: In this assessment component, students will be given two distinct programs, which contain deliberately introduced logical inconsistencies, semantic ambiguities, or syntactical defects. The problems will be framed in an unsolved, application-oriented, or competitive-style format and will be prepared and provided by the faculty to emulate realistic programming scenarios encountered in practice. Students are required to perform a structured examination of the code, identify the sources of error, analyze their impact on program behavior, and apply appropriate corrections to restore the intended functionality. This task is intended to systematically evaluate the students' debugging expertise, their analytical and problem-solving skills, and their ability to modify erroneous code in such a way that it consistently generates the correct and expected output across relevant test cases.		
3	**Image Analysis: Edges, Morphology, and Contours** ***Theory Topics:*** Mathematical gradients for edge boundary analysis (Sobel and Laplacian math); Canny edge detection pipeline rules; Morphological transformations (Erosion, Dilation, Opening, Closing); Contour extraction mechanics, structural hierarchies, and geometric calculations (Area, Perimeter, Aspect Ratio). ***Practical:*** 27. Track horizontal changes in image structures using a directional Sobel edge filter (cv2.Sobel()). 28. Map vertical edge layouts across an image using vertical Sobel gradient filters. 29. Isolate clean structural outlines from a photo file using the automated Canny Edge Detection pipeline. 30. Expand structural highlights in binary image maps using morphological Dilation loops. 31. Shrink away noise dots in background maps using a morphological Erosion filter array. 32. Use a morphological Opening sequence (cv2.MORPH_OPEN) to clean up white noise speckles.	18	20%

33. Use a morphological Closing sequence (cv2.MORPH_CLOSE) to plug black structural holes in broken shapes.
34. Extract coordinate sets for all outer boundaries in a binary image map using cv2.findContours().
35. Overlay extracted contour lines on a canvas using the cv2.drawContours() visualization function.
36. Calculate the exact pixel area of a target shape contour using the cv2.contourArea() metric.
37. Extract bounding boxes (cv2.boundingRect()) for all isolated items found inside an image matrix.
38. Draw visible boundary boxes around target items using the cv2.rectangle() canvas tool.
39. Highlight the path around a complex shape by tracking its convex hull vector endpoints.

Evaluation Method:

Sr. No.	Evolution Methods	SEE	CCE
1	**Program Optimization:** This section consists of three questions of 5 marks each. Students are required to optimize the given code for better efficiency, readability, and performance while maintaining correct functionality.	15	
2	**Difference and Comparison Questions:** Students must compare or differentiate two concepts from the syllabus.	05	
3	**Active Learning Activity** **Smart Object Boundary Detection:** Develop a program for edge detection and contour extraction. Submit the implementation, output, and results individually through the GMIU Web Portal.		10
	Total	20	10

Examination Style:

Program Optimization (15 Marks):
This section comprises three questions of 05 marks each, designed to evaluate students' understanding of code optimization and efficient programming practices. Students are required to analyze the given code, enhance the program logic, and optimize the implementation by reducing time and/or space complexity while maintaining the original functionality and accuracy of the program. They are also expected to minimize redundant statements and improve code structure for better readability and execution efficiency. The questions will be designed and assigned by the faculty members to assess the students' capability in developing clean, structured, efficient, and optimized programming solutions.

	Difference and Comparison Questions (05 Marks): The difference and comparison question carries 05 marks and will be asked from the units covered in the syllabus. Students will be required to clearly differentiate or compare two related concepts, terms, or techniques. This task is designed to test conceptual clarity and the ability to highlight precise distinctions. **Smart Object Boundary Detection (10 Marks):** This activity focuses on the development of an image processing system designed to detect, extract, and analyze object boundaries from input images using advanced edge detection and contour extraction methodologies. Students are expected to design and implement a program capable of identifying prominent intensity transitions, detecting meaningful edge structures, and extracting contour information corresponding to objects present within the image. The complete model along with the corresponding implementation Image and results must be individually submitted by each student through the GMIU Web Portal.		
4	**Advanced Feature Detection and Object Analysis** ***Theory Topics:*** Localized key point tracking; Corner detection math (Harris Corner and FAST algorithms); Descriptor arrays (SIFT and HOG features); Color-based mask segmentation using HSV thresholds; Connecting components and managing real-time coordinates inside Python data maps (Dictionaries, Lists). ***Practical:*** 40. Identify crisp structural corners inside an image using Harris Corner Detection rules (cv2.cornerHarris()). 41. Track fast target corner locations on a laptop screen using a GFTT algorithm (cv2.goodFeaturesToTrack()). 42. Isolate clear target features using SIFT (Scale-Invariant Feature Transform) descriptor runs. 43. Generate feature vectors for image analysis using HOG (Histogram of Oriented Gradients) frameworks. 44. Isolate an orange or blue object from an image by running an HSV threshold pass (cv2.inRange()). 45. Combine multiple binary image layers using standard bitwise AND masking commands. 46. Group connected pixel zones into indexed maps using cv2.connectedComponents(). 47. Filter out small noise artifacts from contour arrays by checking bounding dimensions. 48. Map tracking parameters to detected target profiles using standard Python dictionaries ({}). 49. Save a structured list of key coordinates directly into a standard text CSV file on disk.	18	20%

50. Read saved coordinate entries from a CSV data file to reconstruct tracking markers on an image layout.
51. Group color regions automatically using a fast K-Means pixel clustering process (cv2.kmeans()).
52. Build a simple program that flags shifts in lighting by monitoring average frame brightness values.

Evaluation Method:

Sr. No.	Evolution Methods	SEE	CCE
1	**Logic Enhancement:** Students will optimize two programs to improve efficiency, reduce execution time and memory usage, and maintain correct functionality. Evaluation will be based on code optimization and programming skills.	10	
2	**Viva/Oral Examination:** Evaluates students' understanding of practical concepts, program logic, and applications through oral questioning.	10	
3	**Active Learning Activity Dynamic Contrast Optimizer using Live Histograms:** Develop a real-time video processing system to enhance image brightness and contrast using histogram-based techniques. Submit the source code, implementation, and output individually through the GMIU Web Portal.		10
	Total	20	10

Examination Style:

Logic Enhancement (10 Marks):
In this task, students will be given two programs, each carrying 05 marks. Students need to improve the program logic, reduce execution time and memory usage wherever possible, and shorten the code without changing its original output or functionality. The questions will be practical-based and prepared by the faculty to evaluate the students' understanding of efficient programming, logical thinking, and clean coding practices.

Viva/Oral Examination Style (10 Marks):
The viva/oral test carries 05 marks and will be conducted individually to evaluate the student's understanding of practical concepts. Questions will be asked from the units covered in the syllabus, including logic explanation, code reasoning, and real-time application relevance. This test helps assess clarity of thought, communication skills, and conceptual depth.

	Dynamic Contrast Optimizer using Live Histograms (10 Marks): This project is based on processing live video frames to identify and correct extreme variations in illumination conditions, such as sudden brightness reduction, low-light environments, or overexposed scenes. The system is expected to analyze frame intensity distributions through histogram techniques and automatically adjust contrast and brightness levels to improve visual clarity and image quality in real time. The implementation should demonstrate effective use of image processing concepts, histogram evaluation, and adaptive enhancement techniques for maintaining balanced scene visibility during continuous video streaming. The final source code, along with the implementation output, must be submitted by each student through the GMIU Web Portal.		
5	**Video Processing and Motion Analysis** ***Theory Topics:*** Video capture stream handling from cameras or files; Processing frame loops smoothly; Inter-frame processing loops; Static frame background subtraction math; Motion tracking mechanics (Dense vs. Sparse Optical Flow); Handling multi-threaded video processing, frame queues, and camera teardown safeguards. ***Practical:*** 53. Open a real-time frame stream from an integrated laptop webcam via cv2.VideoCapture(). 54. Run a simple loop to continuously pull, decode, and show active webcam frames without lag. 55. Add a keyboard check to close active video display windows when the user hits 'q'. 56. Extract live frame rates (FPS) by measuring clock differences during stream processing loops. 57. Draw active text readouts on top of moving frames using the cv2.putText() visualization function. 58. Isolate moving targets from static office background environments using an automated MOG2 subtractor. 59. Trigger a frame-saving action when the difference between consecutive frame matrices jumps sharply. 60. Compute motion vectors across sequential frames using Lucas-Kanade Sparse Optical Flow checks. 61. Track continuous real-time movement paths using a dense Farneback calculation pass. 62. Build a real-time face-tracking layer by loading pre-trained Haar Cascade XML files (cv2.CascadeClassifier()). 63. Smooth out video feeds by dropping frame resolutions inside a downscaling matrix step. 64. Run a fast processing script that applies an edge filter overlay to a live camera stream in real time.	18	20%

65. Safely release camera hardware connections and close all open display windows when exiting (cap.release()).

Evaluation Method:

Sr. No.	Evolution Methods	SEE	CCE
1	**Program Correction Task:** Students will be given two faulty programs to identify and correct errors. Evaluation will be based on debugging, coding accuracy, problem-solving, and successful program execution.	10	
2	**Advanced Coding Challenge:** Students will solve one coding problem based on a real-world or competitive programming scenario. Evaluation will focus on logical reasoning, problem-solving, algorithm design, and correct implementation.	10	
3	**Active Learning Activity Real-Time Face Tracker using Pre-trained Haar Cascades:** Develop a real-time face detection and tracking system using a webcam and Haar Cascade. Submit the source code, video, and output individually through the GMIU Web Portal.		10
	Total	20	10

Examination Style:

Program Correction Task (10 Marks):
In this Program Correction Task, students will be provided with two programs containing logical and/or syntactical errors, with each program carrying 05 marks. The questions may be based on unsolved practical problems or competitive coding scenarios prepared by the faculty members. Students are required to identify the errors, correct the program logic, and ensure successful execution with accurate output. This task is intended to evaluate the students' debugging skills, analytical thinking, coding accuracy, and problem-solving capabilities in practical programming situations.

Advanced Coding Challenge (10 Marks):
In the Advanced Coding Challenge, students will be required to solve one question carrying 10 marks based on a real-time problem-solving scenario inspired by competitive programming concepts and practical coding situations. The problem may not be directly related to the prescribed syllabus and is intended to evaluate students' logical reasoning, analytical thinking, algorithmic approach, and programming

	efficiency in unfamiliar problem environments. Students are expected to design an appropriate solution, implement accurate logic, and produce the desired output within the given constraints. The question will be prepared and assigned by the faculty members to assess the students' overall coding proficiency and problem-solving capabilities. **Real-Time Face Tracker using Pre-trained Haar Cascades (10 Marks):** In this project, students are required to develop a live computer vision application that integrates with a webcam video stream to perform real-time face detection and tracking. The system should continuously capture video frames, identify human faces using pre-trained Haar Cascade models, and accurately track and label detected faces on the display screen. The implementation must ensure smooth video rendering with minimal delay or frame lag during continuous processing. The project is intended to evaluate students' understanding of image processing, object detection techniques, real-time video analysis, and efficient implementation of computer vision algorithms using live camera input. The final source code along with the video and output must be submitted by each student through Google drive in the GMIU Web Portal.		

Suggested Specification table with Marks (Theory):100

Distribution of Theory Marks (Revised Bloom's Taxonomy)						
Level	Remembrance (R)	Understanding (U)	Application (A)	Analyze (N)	Evaluate (E)	Create (C)
Weightage %	10%	15%	30%	15%	10%	20%

Course Outcome:

After learning the course, the students should be able to:	
CO1	Process color models, load media, and manipulate pixel matrices using NumPy and OpenCV.
CO2	Run calibration configurations, handle lens distortions, and map vector motion fields accurately on live input arrays.
CO3	Implement edge detection pipelines and morphological operations to extract structural target paths.
CO4	Extract local feature profiles, isolate regions using color channels, and manage spatial parameters inside collection maps.
CO5	Build high-speed processing pipelines to analyze movement, filter live video feeds, and manage camera streams efficiently.