

National University of Computer and Emerging Sciences



Fundamentals Of Computer Vision

Project Report

Qasim Naveed 21L-5231

Rameez 21L-1775

Introduction

This project implements an MCQ checker algorithm using concepts of Computer Vision. The project implements every functionality by itself and does not use built-in function. It mainly uses hough transform and canny edge detection to detect circle and form circular boundary.

This report will explain the project in detail, covering the code and its results.

1. Explanation of Code

For the code the path of the image must be provided.

```
image_path = 'filled10mcq.png'  
input_image = cv2.imread(image_path, cv2.IMREAD_COLOR)
```

After that a white mask of image is created by defining a region of lower white color and high white color range. The `inRange` function uses this range and image and creates a white mask of the image where the area consider white is 255 and the rest of non-white area as 0.

```
lower_white = np.array([200, 200, 200])  
upper_white = np.array([255, 255, 255])  
  
white_mask = inRange(input_image, lower_white, upper_white)
```

After that a non white image or mask is create by just taking bitwise not.

```
non_white_image = bitwise_not(white_mask)
```

After that the input image is converted into grayscaled image as we only need to detect edges and circles so gray scale is enough and rest of the information is necessary. However, this gray image is further edited so that the non white region is converted into black. Then the image is blurred and smoothed out using gaussian blur as it helps in edge detection. Then a threshold is applied to cater for different lightning condition which helps to have a robust mechanism of detecting edges.

```
gray_image = convert_to_grayscale(input_image)  
  
for i in range(gray_image.shape[0]):  
    for j in range(gray_image.shape[1]):  
        if non_white_image[i, j] == 255:  
            gray_image[i, j] = 0  
  
blurred_image = apply_gaussian_blur(gray_image, 5, 1)  
  
thresh_image = adaptive_threshold(blurred_image, max_value=255, adaptive_method='mean', threshold_type='binary_inv', block_size=3, C=5)
```

After that this thresholded image is passed to canny edge detector which detects all the edges in the image. And after that it is passed to hough circular transform to detect circles and draw boundaries. All of the code upto now is implemented by us and the code for canny edge detection and hough circular transform is also built by us however, the code becomes very slow if we still use the self-implemented function so for testing we used cv2 houghcircle.

```
circles = cv2.HoughCircles(thresh_image, cv2.HOUGH_GRADIENT, dp=1.2, minDist=40,  
                           param1=50, param2=30, minRadius=15, maxRadius=30)
```

This gives all the circle detected by the algorithm.

After that we implement an algorithm to detect filled and empty circle. It creates a mask of a bubble and determines a fill ratio of each filled bubble by counting non zero pixels in the image.

```
circle_indices = {'empty': [], 'filled': []}  
output_image = input_image.copy()  
  
if circles is not None:  
    circles = np.round(circles[0, :]).astype('int')  
  
    circles = sorted(circles, key=lambda c: (c[1] // 40, c[0]))  
  
    for i, (x, y, r) in enumerate(circles):  
        mask = np.zeros_like(gray_image)  
        cv2.circle(mask, (x, y), r, 255, -1)  
  
        masked_bubble = cv2.bitwise_and(gray_image, gray_image, mask=mask)  
        fill_ratio = cv2.countNonZero(masked_bubble) / (np.pi * r**2)  
  
        if fill_ratio > 0.2:  
            circle_indices['empty'].append(i+1)  
            cv2.circle(output_image, (x, y), r, (0, 255, 0), 3)  
        else:  
            circle_indices['filled'].append(i+1)  
            cv2.circle(output_image, (x, y), r, (0, 0, 255), 3)
```

After that its just calculation of results and determining whether two filled bubbles appear in same row or not and printing results.

2. Output

For testing we used the following image.

Name :

Roll No. :

- | | | | | |
|-----|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 1. | <input checked="" type="radio"/> | <input type="radio"/> (B) | <input type="radio"/> (C) | <input type="radio"/> (D) |
| 2. | <input type="radio"/> (A) | <input checked="" type="radio"/> | <input type="radio"/> (C) | <input type="radio"/> (D) |
| 3. | <input type="radio"/> (A) | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> (D) |
| 4. | <input checked="" type="radio"/> | <input type="radio"/> (B) | <input type="radio"/> (C) | <input type="radio"/> (D) |
| 5. | <input type="radio"/> (A) | <input checked="" type="radio"/> | <input type="radio"/> (C) | <input type="radio"/> (D) |
| 6. | <input type="radio"/> (A) | <input type="radio"/> (B) | <input type="radio"/> (C) | <input type="radio"/> (D) |
| 7. | <input type="radio"/> (A) | <input checked="" type="radio"/> | <input type="radio"/> (C) | <input type="radio"/> (D) |
| 8. | <input type="radio"/> (A) | <input type="radio"/> (B) | <input checked="" type="radio"/> | <input type="radio"/> (D) |
| 9. | <input type="radio"/> (A) | <input type="radio"/> (B) | <input checked="" type="radio"/> | <input checked="" type="radio"/> |
| 10. | <input checked="" type="radio"/> | <input type="radio"/> (B) | <input type="radio"/> (C) | <input type="radio"/> (D) |









































This is the answer key for this bubble sheet we wrote in our program.

```
answer_key = ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'A']
```

And this is the detected output

Name :

Roll No. :

- | | | | | |
|-----|---|---|---|--|
| 1. |  |  |  |  |
| 2. |  |  |  |  |
| 3. |  |  |  |  |
| 4. |  |  |  |  |
| 5. |  |  |  |  |
| 6. |  |  |  |  |
| 7. |  |  |  |  |
| 8. |  |  |  |  |
| 9. |  |  |  |  |
| 10. |  |  |  |  |

Where green circles are empty circles and red circles are filled circles.

```
1: ['A']
2: ['B']
3: ['B', 'C']
4: ['A']
5: ['B']
6: []
7: ['B']
8: ['C']
9: ['C', 'D']
10: ['A']
{'total_marks': 4.5, 'detailed_results': {1: {'status': 'Correct', 'marks': 1, 'filled_option': 'A'}, 2: {'status': 'Correct', 'marks': 1, 'filled_option': 'B'}, 3: {'status': 'Multiple options', 'penalty': -0.25, 'filled_options': ['B', 'C']}, 4: {'status': 'Correct', 'marks': 1, 'filled_option': 'A'}, 5: {'status': 'Correct', 'marks': 1, 'filled_option': 'B'}, 6: {'status': 'Not attempted', 'marks': 0}, 7: {'status': 'Incorrect', 'marks': 0, 'filled_option': 'B'}, 8: {'status': 'Incorrect', 'marks': 0, 'filled_option': 'C'}, 9: {'status': 'Multiple options', 'penalty': -0.25, 'filled_options': ['C', 'D']}, 10: {'status': 'Correct', 'marks': 1, 'filled_option': 'A'}}}
PS D:\assignments\semester 7\cv\project>
```

This is the detailed result breakdown of every question. It correctly identifies wrong answers, correct answers, unfilled answers, and multiple filled answers. The total marks is 4.5 marks as is the correct result. The penalty for multiple answers is 0.25.