NATURAL LANGUAGE PROCESSING MINI PROJECT REPORT

On

Feature Extraction using Zernike Moments

By

Ishan Raskar

Sakshi Pawar

Sakshi Dhavale

Under the guidance of

Mrs. Mayuri Narkhede

“ येथे बहुतांचे हित ”

Department of Computer Engineering

Marathwada Mitra Mandal’s College of Engineering

Acknowledgement

I feel great pleasure in expressing my deepest sense of gratitude and sincere thanks to my guide **Mrs. Mayuri Narkhede** for her valuable guidance during the Project work, without which it would have been a very difficult task. I have no words to express my sincere thanks for valuable guidance, extreme assistance and cooperation extended to all the **Staff Members** of my Department.

This acknowledgement would be incomplete without expressing my special thanks to **Prof. K. S Thakre**, Head of the Department (Computer Engineering) for their support during the work.

I would also like to extend my heartfelt gratitude to my **Principal, Dr. V N Gohokar** who provided a lot of valuable support, mostly being behind the veils of college bureaucracy.

Last but not least I would like to thank all the Teaching, Non- Teaching staff members of my Department, my parents and my colleagues who helped me directly or indirectly for completing this Project successfully.

Ishan Raskar
Sakshi Pawar
Sakshi Dhavale

Contents

1. ABSTRACT

Zernike moments possess valuable mathematical properties that make them highly suitable as image features for shape classification tasks. Their rotational invariance ensures that the descriptors remain consistent regardless of the orientation of the shape. Furthermore, with additional techniques, Zernike moments can be made scale and translational invariant, enhancing their robustness for different image scales and positions. However, the correct application of Zernike moments requires careful consideration of various factors. This paper explores several techniques to optimize the usage of Zernike moments as shape descriptors, and the experimental results have demonstrated exceptional accuracy, reaching up to 100% in certain cases. These findings highlight the effectiveness and potential of Zernike moments in shape classification problems.

2. INTRODUCTION

In the field of image processing and pattern recognition, feature extraction plays a crucial role in identifying and distinguishing objects within an image. One of the widely used techniques for shape representation is the use of Zernike Moments (ZMs), which provide a robust and rotation-invariant descriptor of image features. Zernike Moments are a set of complex polynomials defined on the unit disk that form an orthogonal basis, allowing for efficient shape representation with minimal redundancy. This project explores the process of extracting features from images using Zernike Moments. The implementation involves computing the Zernike moments of an image, analyzing their effectiveness in shape characterization, and demonstrating their applications in pattern recognition tasks. The results provide insights into the efficiency and accuracy of Zernike Moments in distinguishing different shapes and structures.

Problem definition

Feature extraction is a crucial step in image processing and pattern recognition, but traditional methods often struggle with rotation sensitivity, noise, and redundancy. This project focuses on extracting shape-based features using Zernike Moments (ZMs), which provide a robust, rotation-invariant, and noise-resistant representation of images. The goal is to implement and analyze Zernike Moment-based feature extraction for improved shape recognition and classification. This method finds applications in areas like optical character recognition (OCR), medical imaging, and object detection.

3. SOFTWARE & HARDWARE REQUIREMENTS

3.1 Software Requirements:

1. **Operating System**: Windows, Linux, or macOS
2. **Programming Language**: Python (preferred for implementation)
3. **Libraries & Tools**:
    - NumPy (for numerical computations)
    - OpenCV (for image processing)
    - Matplotlib (for visualization)
    - Scipy (for mathematical functions)
    - Zernike Moments library (if using prebuilt functions)
4. **Jupyter Notebook / Anaconda** (for interactive coding and debugging)

3.2 Hardware Requirements:

1. **Processor**: Intel Core i5 or higher
2. **RAM**: Minimum 4GB (8GB or more recommended for large image datasets)
3. **Storage**: At least 10GB free space for dataset storage and software installation

4. WORKING

The project "Feature Extraction using Zernike Moments" involves detecting and extracting shape-based features from images using Zernike Moments. The workflow is as follows:

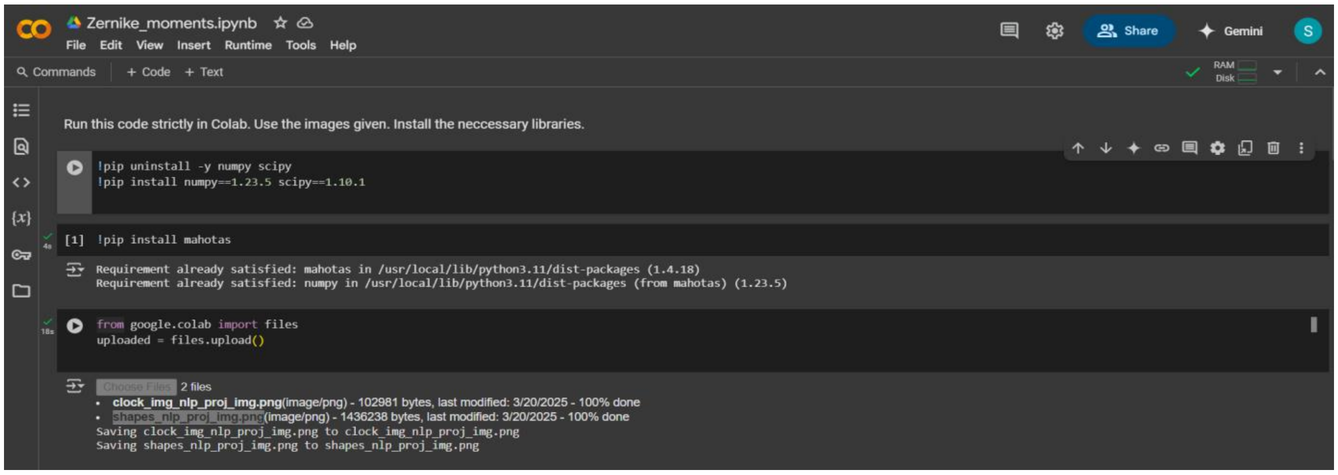
1. Image Preprocessing

- Load the input image using OpenCV.
- Convert the image to grayscale and apply Gaussian blur to remove noise.
- Use thresholding to create a binary image for contour detection.
- Perform morphological operations (dilation & erosion) to enhance object boundaries.

2. Contour Detection & Region of Interest (ROI) Extraction
   - Detect contours in the processed image.
   - For each detected contour, create a mask and extract the bounding box around the object.
3. Zernike Moment Computation
   - Compute Zernike Moments for each ROI using the Mahotas library.
   - Store these computed moments as feature vectors, which uniquely represent the shape.
4. Feature Matching & Object Recognition
   - Compute the Euclidean distance between the extracted shape features and a reference object.
   - Identify the object with the smallest distance, indicating the closest match.
5. Visualization & Results
   - Draw bounding boxes around detected objects.
   - Highlight the matched object with a distinct color and display results using OpenCV.

This structured approach ensures efficient and accurate feature extraction, making it useful for pattern recognition, shape classification, and object detection applications.

4. CODE SNIPPETS



Run this code strictly in Colab. Use the images given. Install the neccessary libraries.

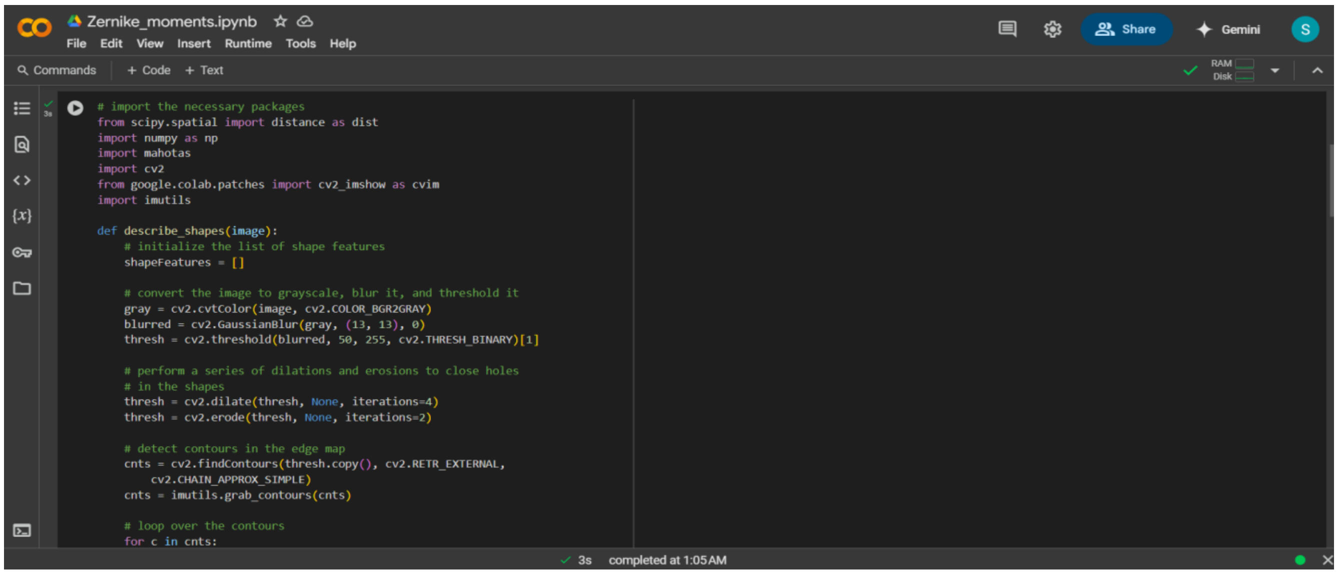
```
!pip uninstall -y numpy scipy
!pip install numpy==1.23.5 scipy==1.10.1
```

```
[1] !pip install mahotas
```

```
Requirement already satisfied: mahotas in /usr/local/lib/python3.11/dist-packages (1.4.18)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from mahotas) (1.23.5)
```

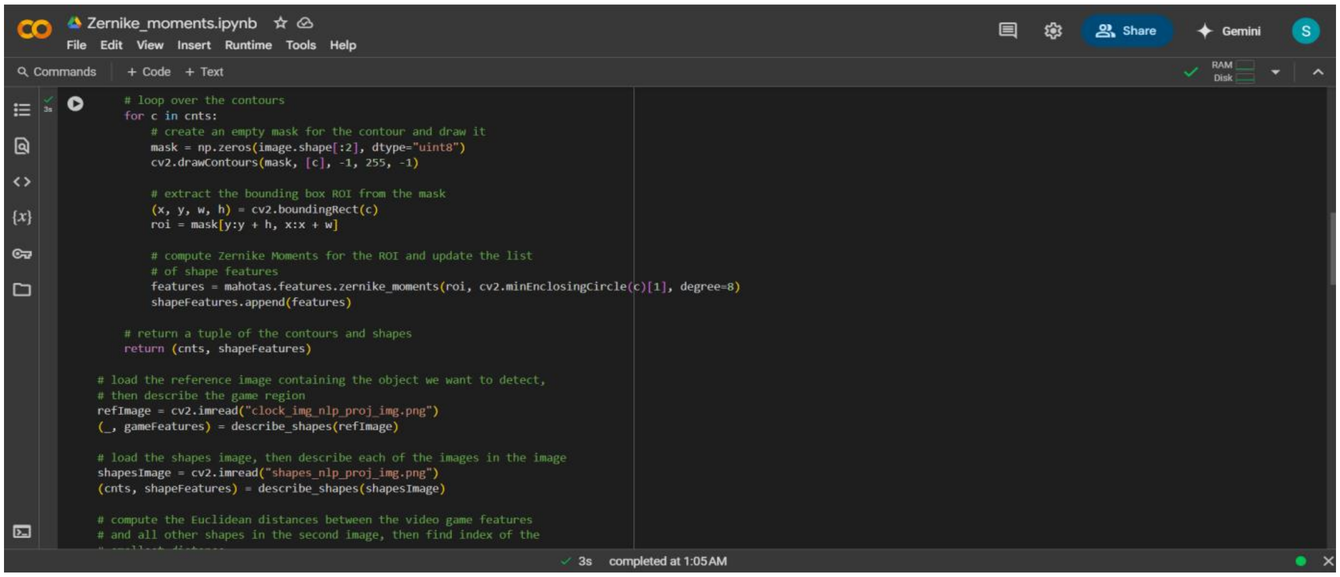
```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files 2 files
• clock_img_nlp_proj_img.png(image/png) - 102981 bytes, last modified: 3/20/2025 - 100% done
• shapes_nlp_proj_img.png(image/png) - 1436238 bytes, last modified: 3/20/2025 - 100% done
Saving clock_img_nlp_proj_img.png to clock_img_nlp_proj_img.png
Saving shapes_nlp_proj_img.png to shapes_nlp_proj_img.png
```



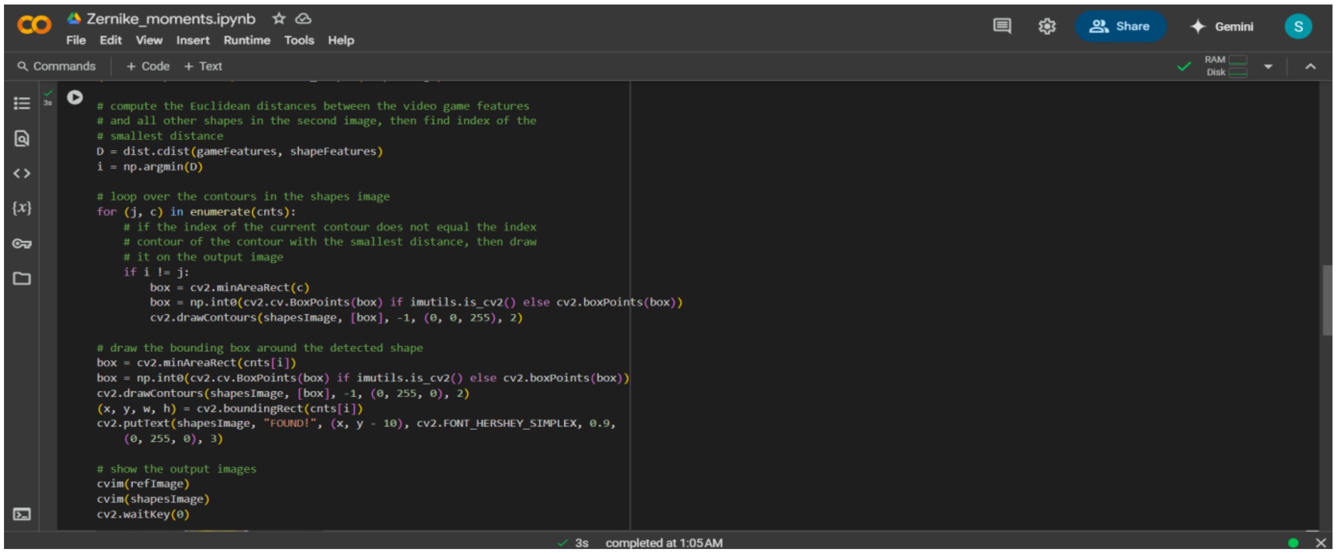
```python
# import the necessary packages
from scipy.spatial import distance as dist
import numpy as np
import mahotas
import cv2
from google.colab.patches import cv2_imshow as cvim
import imutils

def describe_shapes(image):
    # initialize the list of shape features
    shapeFeatures = []

    # convert the image to grayscale, blur it, and threshold it
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (13, 13), 0)
    thresh = cv2.threshold(blurred, 50, 255, cv2.THRESH_BINARY)[1]

    # perform a series of dilations and erosions to close holes
    # in the shapes
    thresh = cv2.dilate(thresh, None, iterations=4)
    thresh = cv2.erode(thresh, None, iterations=2)

    # detect contours in the edge map
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

    # loop over the contours
    for c in cnts:
```

3s completed at 1:05 AM



```python
    # loop over the contours
    for c in cnts:
        # create an empty mask for the contour and draw it
        mask = np.zeros(image.shape[:2], dtype="uint8")
        cv2.drawContours(mask, [c], -1, 255, -1)

        # extract the bounding box ROI from the mask
        (x, y, w, h) = cv2.boundingRect(c)
        roi = mask[y:y + h, x:x + w]

        # compute Zernike Moments for the ROI and update the list
        # of shape features
        features = mahotas.features.zernike_moments(roi, cv2.minEnclosingCircle(c)[1], degree=8)
        shapeFeatures.append(features)

    # return a tuple of the contours and shapes
    return (cnts, shapeFeatures)

# load the reference image containing the object we want to detect,
# then describe the game region
refImage = cv2.imread("clock_img_nlp_proj_img.png")
(_, gameFeatures) = describe_shapes(refImage)

# load the shapes image, then describe each of the images in the image
shapesImage = cv2.imread("shapes_nlp_proj_img.png")
(cnts, shapeFeatures) = describe_shapes(shapesImage)

# compute the Euclidean distances between the video game features
# and all other shapes in the second image, then find index of the
```

3s completed at 1:05 AM



```python
# compute the Euclidean distances between the video game features
# and all other shapes in the second image, then find index of the
# smallest distance
D = dist.cdist(gameFeatures, shapeFeatures)
i = np.argmin(D)

# loop over the contours in the shapes image
for (j, c) in enumerate(cnts):
    # if the index of the current contour does not equal the index
    # contour of the contour with the smallest distance, then draw
    # it on the output image
    if i != j:
        box = cv2.minAreaRect(c)
        box = np.int0(cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box))
        cv2.drawContours(shapesImage, [box], -1, (0, 0, 255), 2)

# draw the bounding box around the detected shape
box = cv2.minAreaRect(cnts[i])
box = np.int0(cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box))
cv2.drawContours(shapesImage, [box], -1, (0, 255, 0), 2)
(x, y, w, h) = cv2.boundingRect(cnts[i])
cv2.putText(shapesImage, "FOUND!", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
    (0, 255, 0), 3)

# show the output images
cvim(refImage)
cvim(shapesImage)
cv2.waitKey(0)
```

3s completed at 1:05 AM

Code:

```
!pip install numpy –upgrade

!pip install mahotas

# import the necessary packages
from scipy.spatial import distance as dist
import numpy as np
import mahotas
import cv2
from google.colab.patches import cv2_imshow as cvim
import imutils

def describe_shapes(image):
   # initialize the list of shape features
   shapeFeatures = []

   # convert the image to grayscale, blur it, and threshold it
   gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
   blurred = cv2.GaussianBlur(gray, (13, 13), 0)
   thresh = cv2.threshold(blurred, 50, 255, cv2.THRESH_BINARY)[1]

   # perform a series of dilations and erosions to close holes
   # in the shapes
   thresh = cv2.dilate(thresh, None, iterations=4)
   thresh = cv2.erode(thresh, None, iterations=2)
```

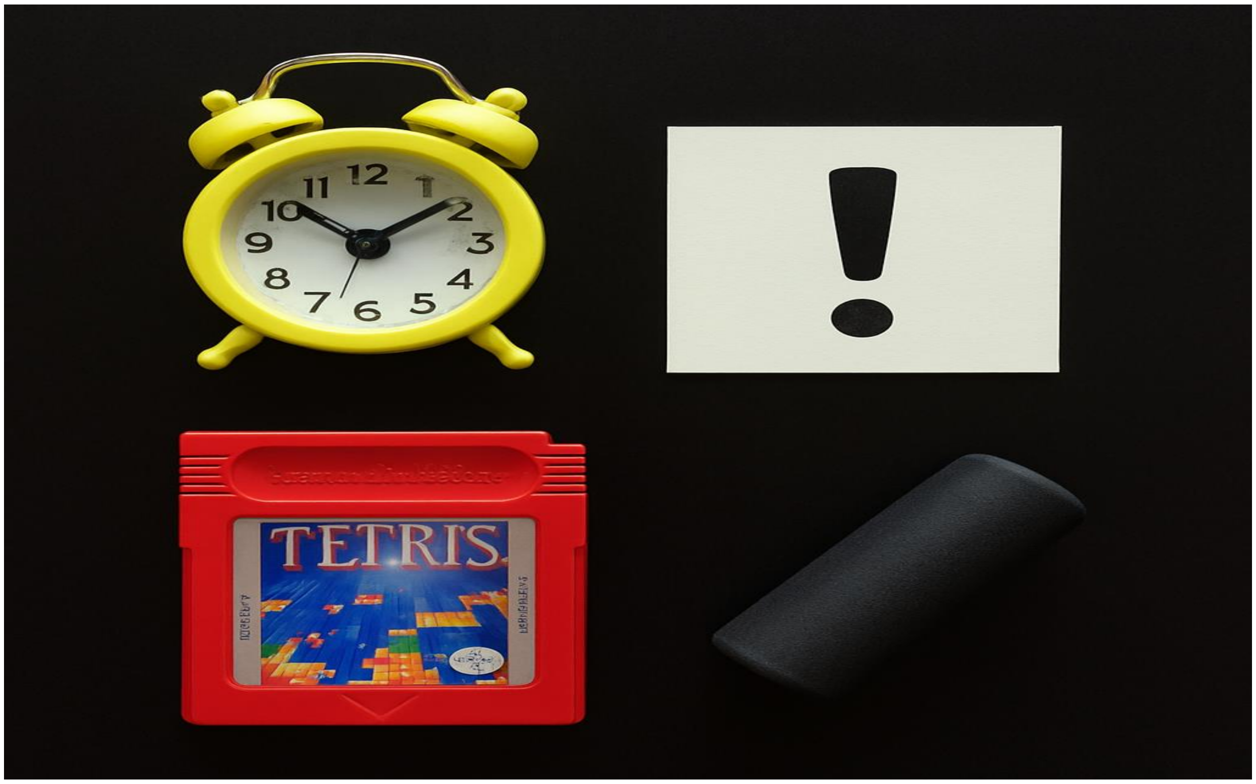
```
    # detect contours in the edge map
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

    # loop over the contours
    for c in cnts:
        # create an empty mask for the contour and draw it
        mask = np.zeros(image.shape[:2], dtype="uint8")
        cv2.drawContours(mask, [c], -1, 255, -1)

        # extract the bounding box ROI from the mask
        (x, y, w, h) = cv2.boundingRect(c)
        roi = mask[y:y + h, x:x + w]

        # compute Zernike Moments for the ROI and update the list
        # of shape features
        features = mahotas.features.zernike_moments(roi,
cv2.minEnclosingCircle(c)[1], degree=8)
        shapeFeatures.append(features)

    # return a tuple of the contours and shapes
    return (cnts, shapeFeatures)

# load the reference image containing the object we want to detect,
# then describe the game region
```

```
refImage = cv2.imread("/content/pokemon_red.png")
(_, gameFeatures) = describe_shapes(refImage)

# load the shapes image, then describe each of the images in the image
shapesImage = cv2.imread("/content/shapes.png")
(cnts, shapeFeatures) = describe_shapes(shapesImage)

# compute the Euclidean distances between the video game features
# and all other shapes in the second image, then find index of the
# smallest distance
D = dist.cdist(gameFeatures, shapeFeatures)
i = np.argmin(D)

# loop over the contours in the shapes image
for (j, c) in enumerate(cnts):
    # if the index of the current contour does not equal the index
    # contour of the contour with the smallest distance, then draw
    # it on the output image
    if i != j:
        box = cv2.minAreaRect(c)
        box = np.int0(cv2.cv.BoxPoints(box) if imutils.is_cv2() else
cv2.boxPoints(box))
        cv2.drawContours(shapesImage, [box], -1, (0, 0, 255), 2)

# draw the bounding box around the detected shape
box = cv2.minAreaRect(cnts[i])
box = np.int0(cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box))
```

```
cv2.drawContours(shapesImage, [box], -1, (0, 255, 0), 2)
(x, y, w, h) = cv2.boundingRect(cnts[i])
cv2.putText(shapesImage, "FOUND!", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9,
    (0, 255, 0), 3)

# show the output images
cvim(refImage)
cvim(shapesImage)
cv2.waitKey(0)
```

5. RESULTS

Input:



Output: