Theory Question

Lecture 15 presented two very famous algorithms for image segmentation: The Otsu Algorithm and the Watershed Algorithm. These algorithms are as different as night and day. Present in your own words the strengths and the weaknesses of each. (Note that the Watershed algorithm uses the morphological operators that we discussed in Lecture 14.)

Otsu's Algorithm is a simple and fast algorithm and performs well when the histogram of an image has a bimodal distribution with a deep and sharp valley between the two peaks. It performs badly when there is lot of noise, when object sizes are small, when lighting is inhomogeneous and when intra-class is larger than inter-class variance.

Watershed Algorithm, on the other hand, generates a topological surface of the intensity of pixels where high intensity denotes peaks and hills while low intensity denotes valleys. It deals with noise better than Otsu's algorithm since it has a large variation in the gradient levels of the image. However, it is possible to oversegment an image and to avoid that Watershed algorithm requires a lot of markers.

Task 1

For Task 1, two variants of the Otsu algorithm were implemented on images shown below.

(a) Image 1 (b) Image 2

Figure 1: Input Images

Otsu's Algorithm: Otsu's algorithm calculates a threshold that divides pixels of an image into two classes: foreground and background. This threshold is determined by maximizing inter-class variance or by minimizing the intra-class variance. The inter-class variance is given by:

$$\sigma_b^2 = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

where $\omega_0(t)$ and $\omega_1(t)$ are probabilities of the two classes separated by threshold t and are calculated from L bins of histogram. N is the total number of pixels in the image, and n_i is the number of pixels at i^{th} grayscale level:

$$p(i) = \frac{n_i}{N}$$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

and $\mu_0(t)$ and $\mu_1(t)$ are class mean, calculated as:

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\mu_0(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

Task 1.1: Image segmentation using RGB values

Background

Brief Summary of the process:

1. The image was separated into three color channels: Blue, Green and Red
2. Each channel was then passed through Otsu's algorithm to find the threshold to divide the channel into foreground and background.
3. The threshold was used to create the mask of the foreground for each channel.
4. The masks for all three channels were then combined with logical "AND" operation.

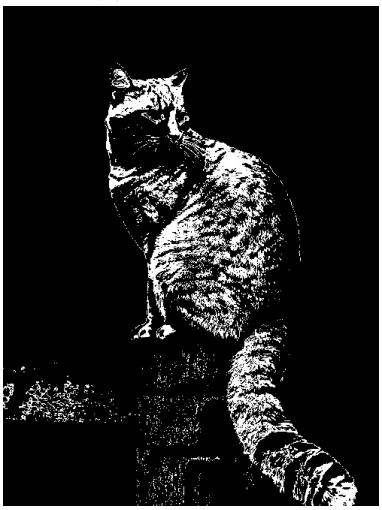
Outputs

(a) Image 1

(b) Blue Mask

(c) Green Mask

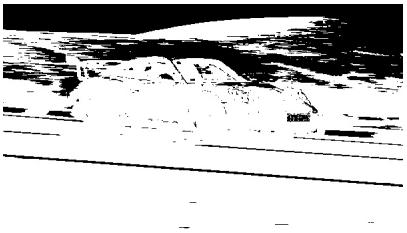
(d) Red Mask



(e) Combined Mask

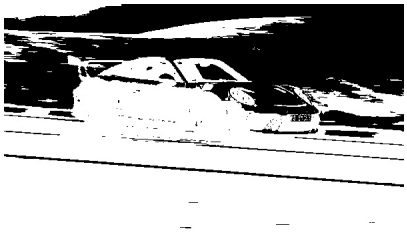
Figure 2: BGR Segmentation masks

(a) Image 1

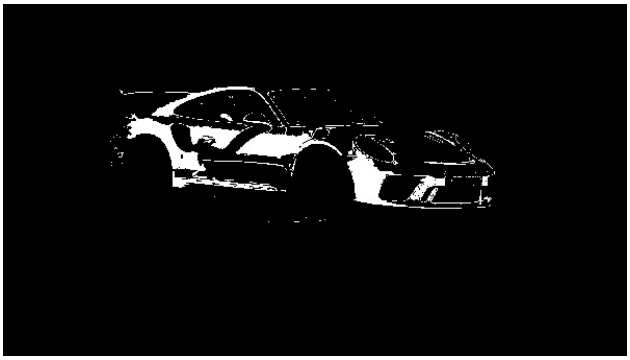


(b) Blue Mask

(c) Green Mask



(d) Red Mask



(e) Combined Mask

Figure 3: BGR Segmentation masks

Parameters:
Image: Cat
Channels: Blue, Green, Red
Iterations: 1,1,1
Flip: 0,0,1

Image: Car
Channels: Blue, Green, Red
Iterations: 1,1,1
Flip: 0,1,0

Task 1.2: Texture-based segmentation

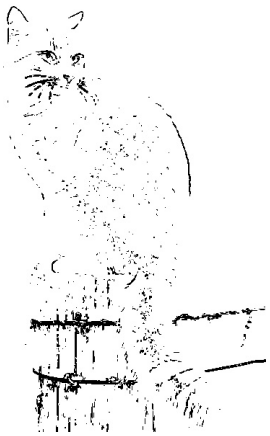
Background

Brief Summary of the process:

1. The image is converted to gray scale.
2. For the sliding window approach, three window sizes are selected (eg. 5, 7, 9)
3. The window of size N x N is then placed at each pixel and the intensity variance within the window is calculated as the texture feature of that center pixel.
4. This creates a texture feature mask which is then passed through the Otsu's algorithm to divide it into foreground and background.
5. The masks for all three window sizes are then combined with logical "AND" operation.

Outputs

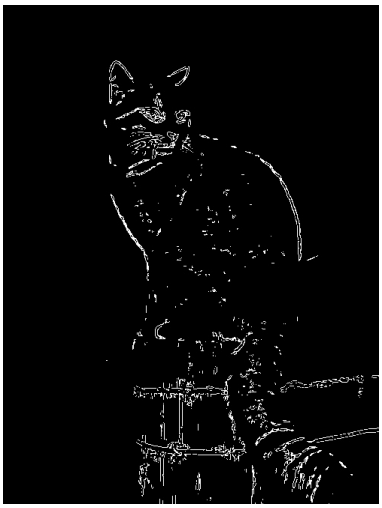
(a) Image 1



(b) N = 3

(c) N = 5

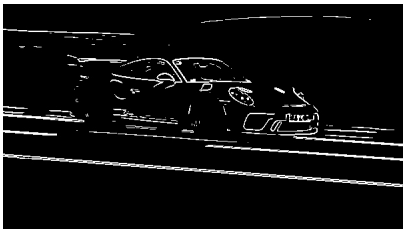
(d) N = 7



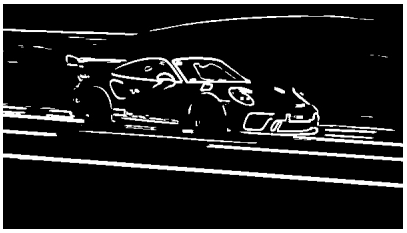
(e) Combined Mask

Figure 4: Texture Segmentation masks

(a) Image 1

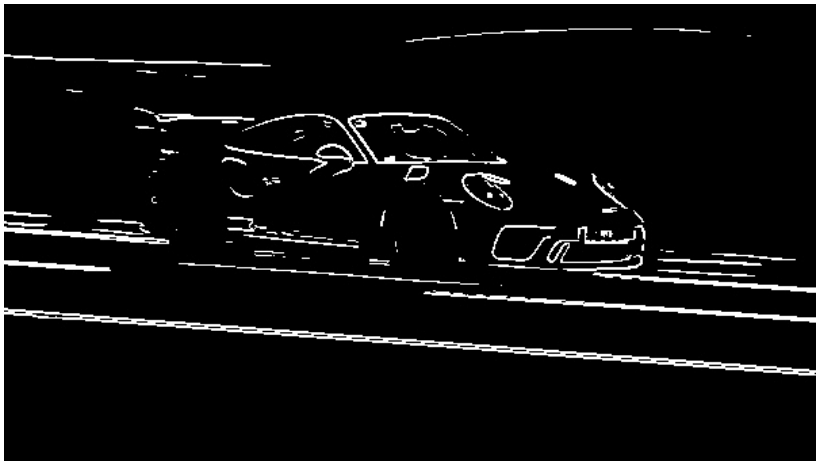


(b) N = 3



(c) N = 5

(d) N = 7



(e) Combined Mask

Figure 5: Texture Segmentation masks

Parameters:
Image: Cat
Window Size: 3,5,7
Iterations: 1,1,1

Image: Car
Window Size: 3,5,7
Iterations: 1,1,1

Task 1.3: Contour Extraction

Background

Brief Summary of the process:

1. Once the masks from BGR Segmentation and Texture Segmentations are computed, Dilation and Erosion is performed on them to check if the quality of the foreground vs background increases or not.

2. For each mask, "Opening" and "Closing" Morphological operations are performed followed by contour extraction.

3. For contour extraction, each pixel with value 1 is checked if it has at least one of its eight neighbours has a value 0, it is copied onto the contour mask.

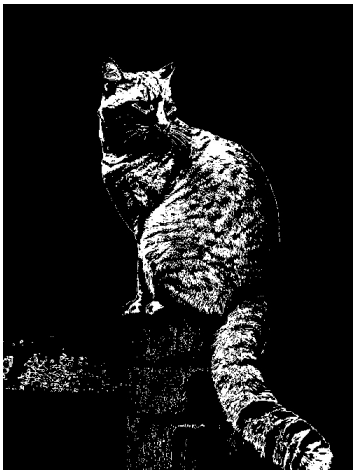
Dilation: It is a morphological operation which computes the maximum intensity over the area of the image that overlaps with the kernel and then sets the intensity of the pixel under the anchor point of the kernel with this maximum value. This results in bright regions expanding in size.

Erosion: It is a morphological operation which computes the minimum intensity over the area of the image that overlaps with the kernel and sets the pixel under the anchor point of the kernel to the minimum intensity value. This results in bright regions shrinking in size.

Opening: This is when Erosion is performed first on the image, followed by Dilation. It is used to remove internal noises.

Closing: This is when Dilation is performed first on the image, followed by Erosion. It is used for smoothening of the image and fusing of narrow breaks.

Outputs



(a) Combined Mask (CM) from BGR Segmentation

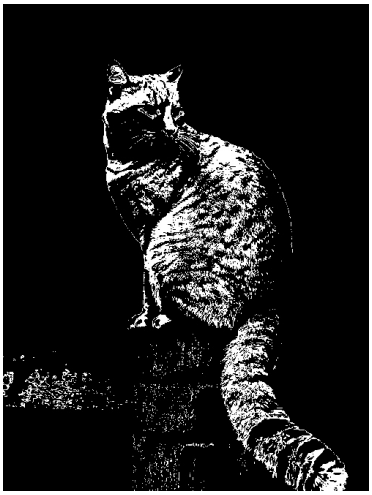
(b) CM -> Erosion

(c) CM -> Erosion -> Dilation

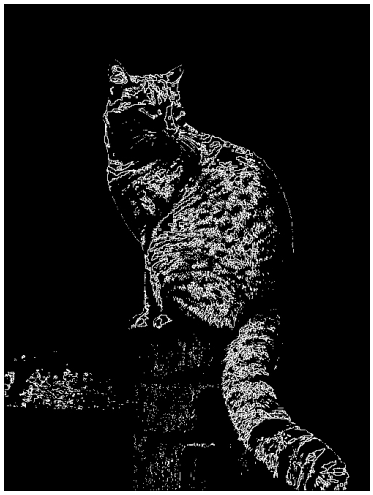
(d) CM -> Dilation

(e) CM -> Dilation -> Erosion

Figure 6: Opening (b and c) and Closing (d and e) Morphological Operations



(a) Combined Mask (CM)



(b) Combined Mask Contour

(c) Opening Mask

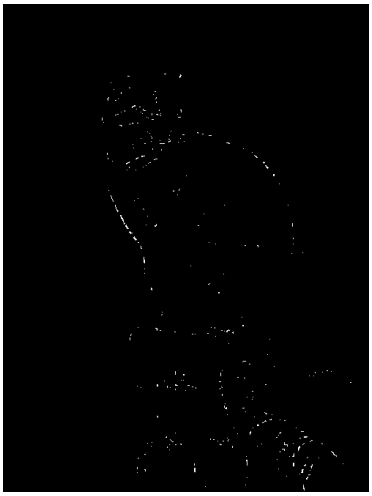
(d) Contour after Opening

(e) Closing Mask

(f) Contour after Closing

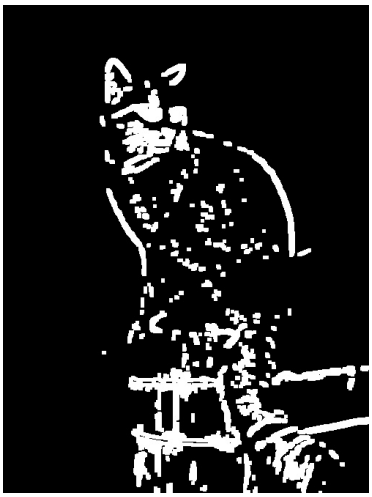
Figure 7: Final Contours for different Masks

(a) Combined Mask (CM) from Texture Segmentation

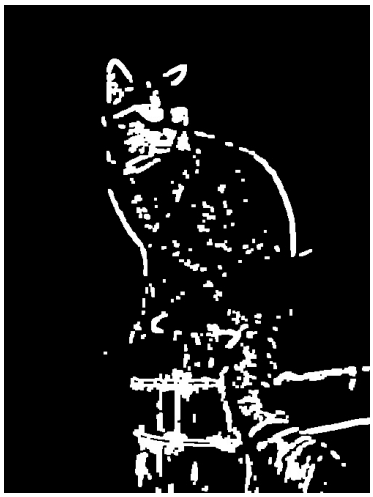


(b) CM -> Erosion

(c) CM -> Erosion -> Dilation

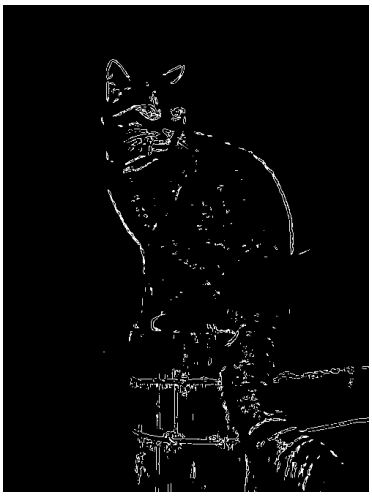


(d) CM -> Dilation

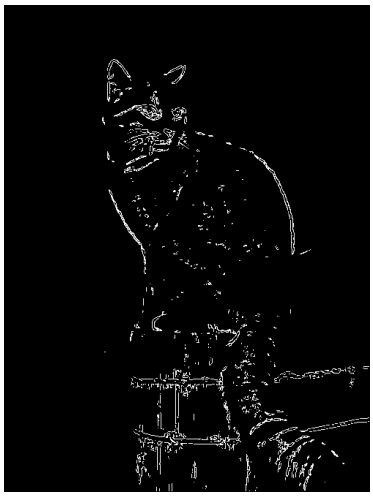


(e) CM -> Dilation -> Erosion

Figure 8: Opening (b and c) and Closing (d and e) Morphological Operations

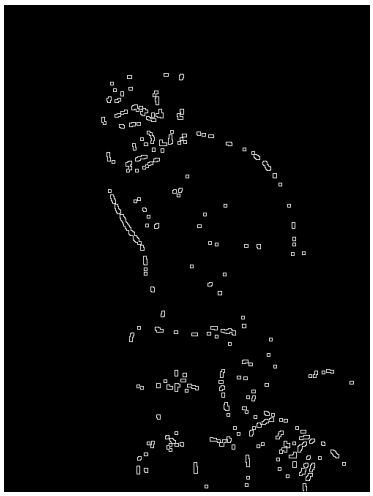


(a) Combined Mask (CM)

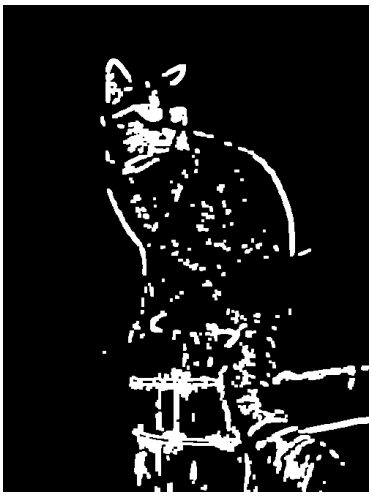


(b) Combined Mask Contour

(c) Opening Mask



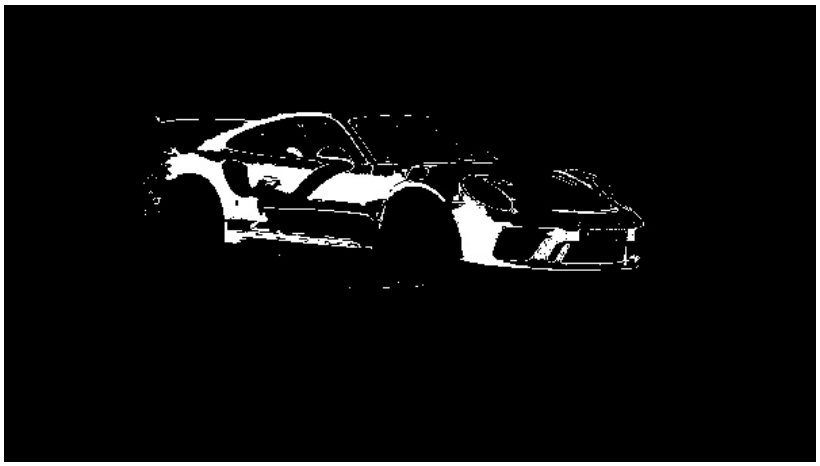
(d) Contour after Opening



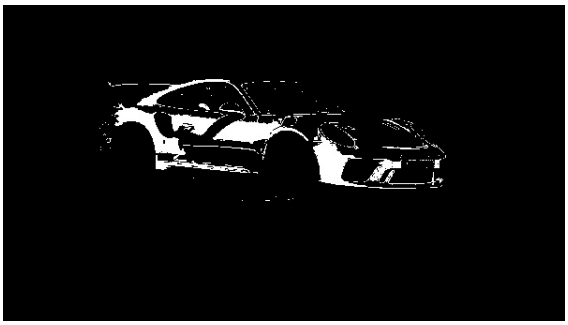
(e) Closing Mask

(f) Contour after Closing

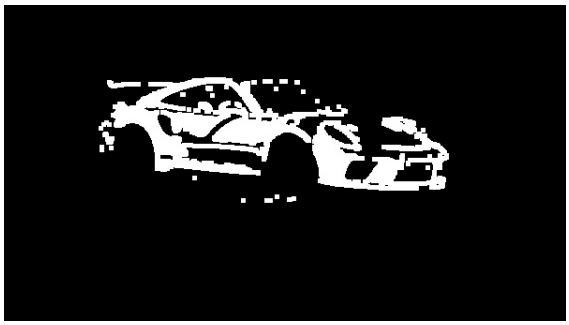
Figure 9: Final Contours for different Masks



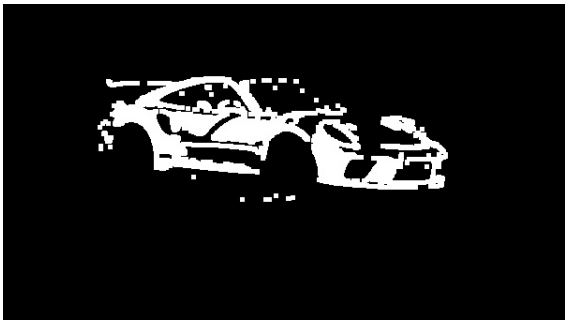
(a) Combined Mask (CM) from BGR Segmentation



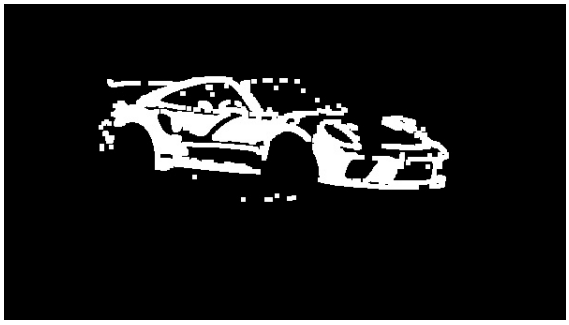
(b) CM -> Erosion



(c) CM -> Erosion -> Dilation

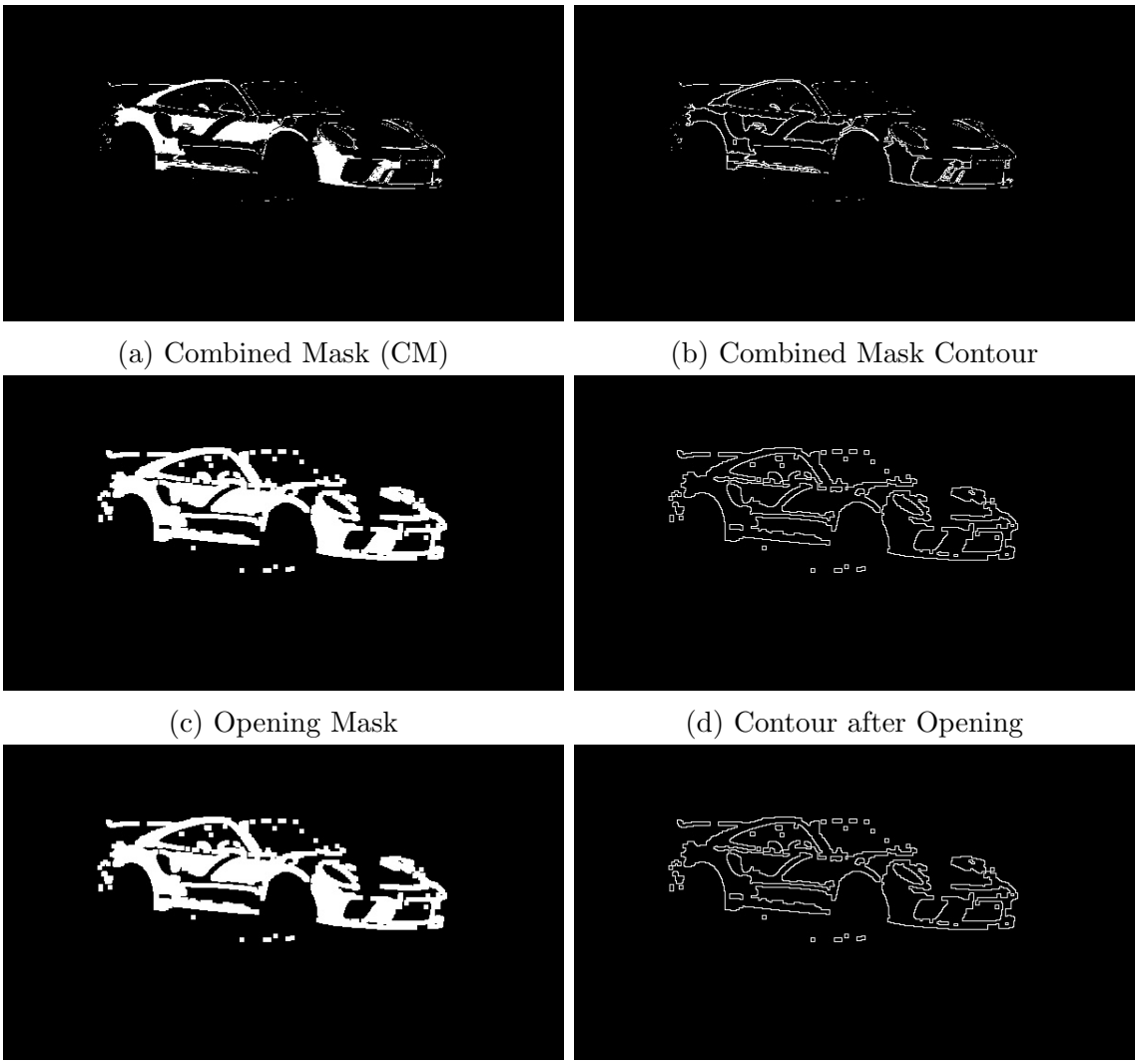


(d) CM -> Dilation



(e) CM -> Dilation -> Erosion

Figure 10: Opening (b and c) and Closing (d and e) Morphological Operations



(a) Combined Mask (CM)

(b) Combined Mask Contour

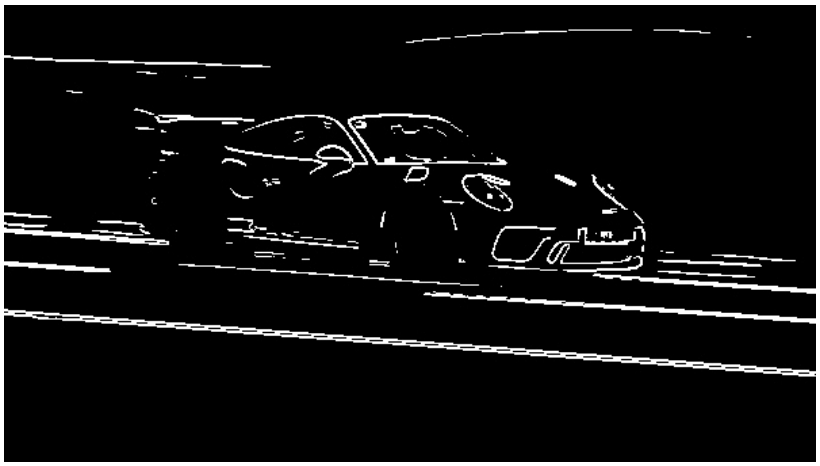
(c) Opening Mask

(d) Contour after Opening

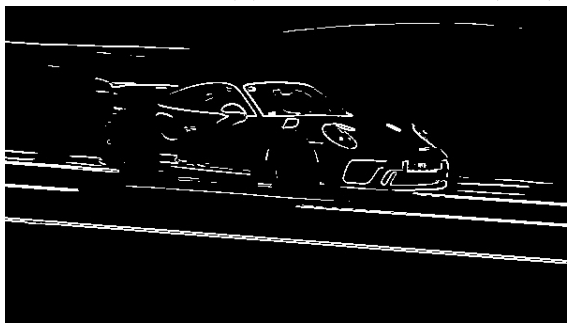
(e) Closing Mask

(f) Contour after Closing

Figure 11: Final Contours for different Masks



(a) Combined Mask (CM) from Texture Segmentation



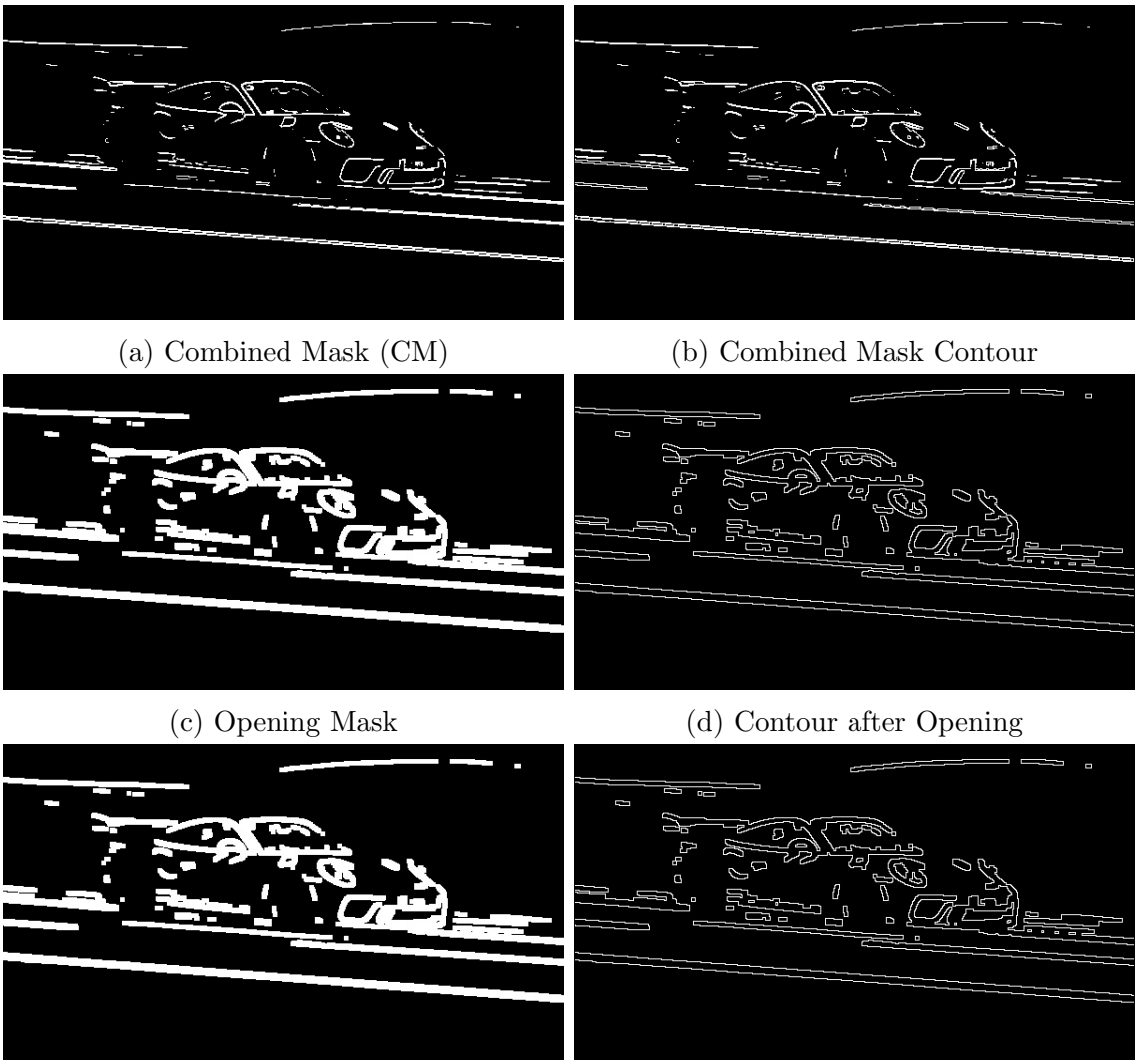
(b) CM -> Erosion

(c) CM -> Erosion -> Dilation

(d) CM -> Dilation

(e) CM -> Dilation -> Erosion

Figure 12: Opening (b and c) and Closing (d and e) Morphological Operations



(a) Combined Mask (CM)

(b) Combined Mask Contour

(c) Opening Mask

(d) Contour after Opening

(e) Closing Mask

(f) Contour after Closing

Figure 13: Final Contours for different Masks

Parameters:
Image: Cat
Erosion Size: 2
Erosion iterations: 1
Dilation Size: 3
Dilation Iterations: 2

Image: Car
Erosion Size: 2
Erosion Iterations: 1
Dilation Size: 3
Dilation Iterations: 2

Task 2

For Task 2, two variants of the Otsu algorithm were implemented on images shown below.

(a) Image 1

(b) Image 2

Figure 14: Input Images

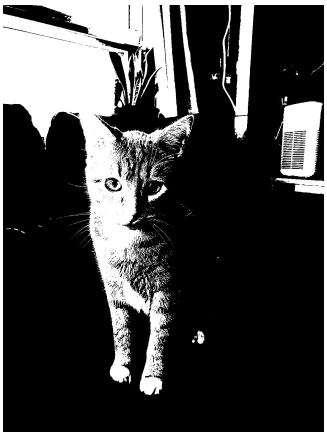
Outputs

Task 2.1: Image segmentation using RGB values

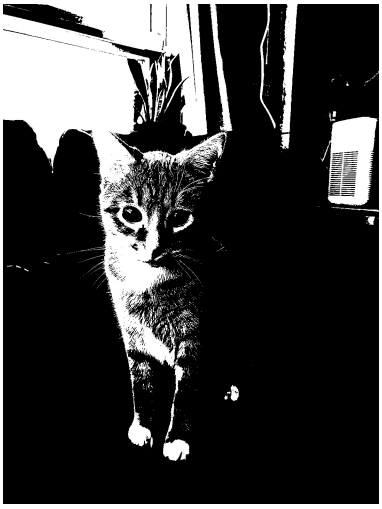
(a) Image 1

(b) Blue Mask

(c) Green Mask



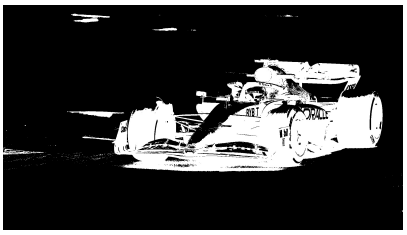
(d) Red Mask



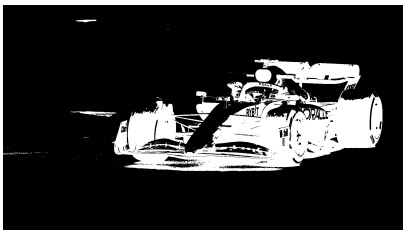
(e) Combined Mask

Figure 15: BGR Segmentation masks

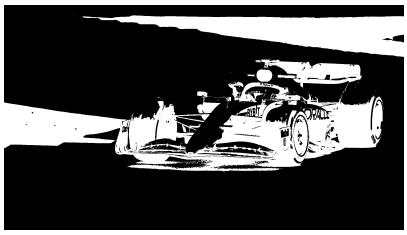
(a) Image 1



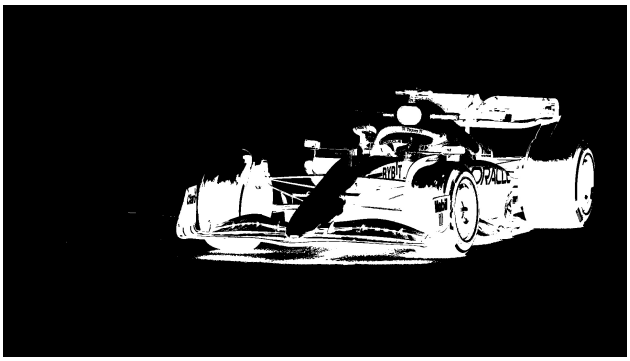
(b) Blue Mask



(c) Green Mask



(d) Red Mask



(e) Combined Mask

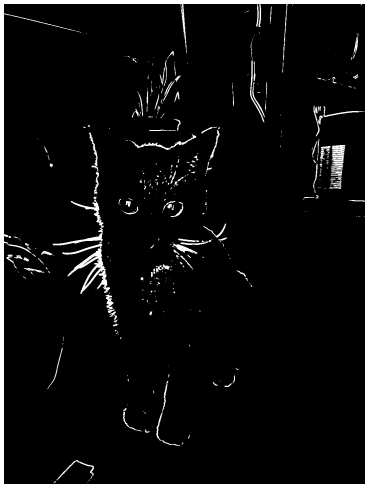
Figure 16: BGR Segmentation masks

Parameters:
Image: Cat
Channels: Blue, Green, Red
Iterations: 1,1,2
Flip: 1,1,1

Image: Car
Channels: Blue, Green, Red
Iterations: 2,2,2
Flip: 0,0,0

Task 2.2: Texture-based segmentation

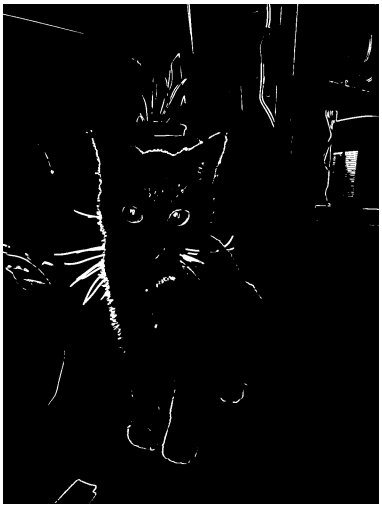
(a) Image 1



(b) N = 3

(c) N = 5

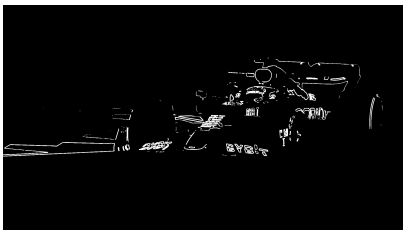
(d) N = 7



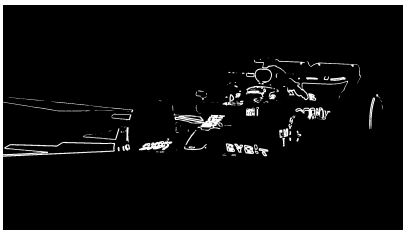
(e) Combined Mask

Figure 17: Texture Segmentation masks

(a) Image 1

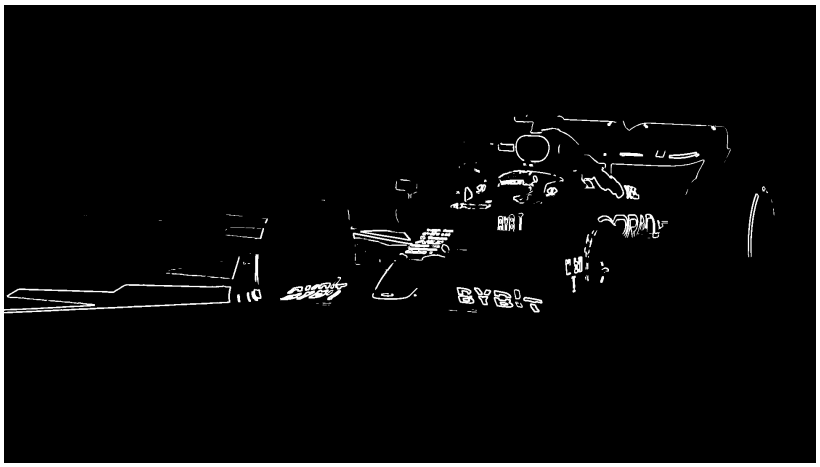


(b) N = 3



(c) N = 5

(d) N = 7



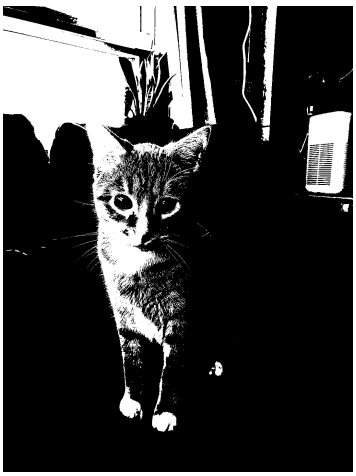
(e) Combined Mask

Figure 18: Texture Segmentation masks

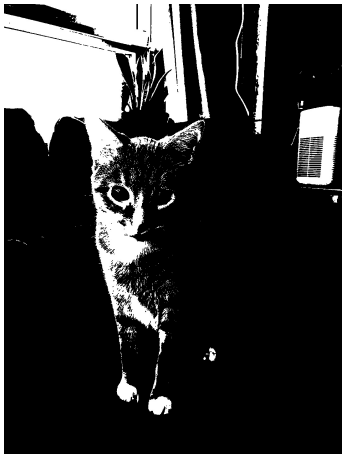
Parameters:
Image: Cat
Window Size: 5,7,9
Iterations: 1,1,1

Image: Car
Window Size: 5,7,9
Iterations: 1,1,1

Task 2.3: Contour Extraction

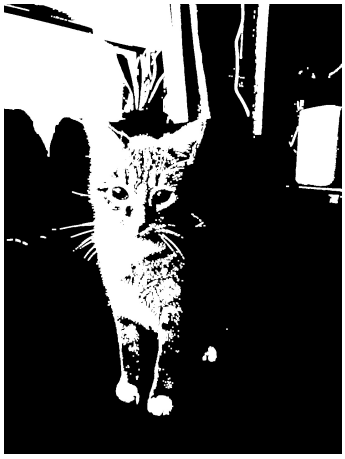


(a) Combined Mask (CM) from BGR Segmentation

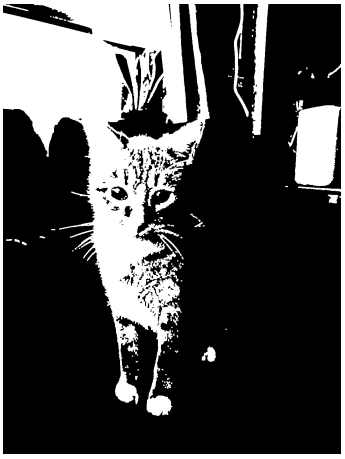


(b) CM -> Erosion

(c) CM -> Erosion -> Dilation

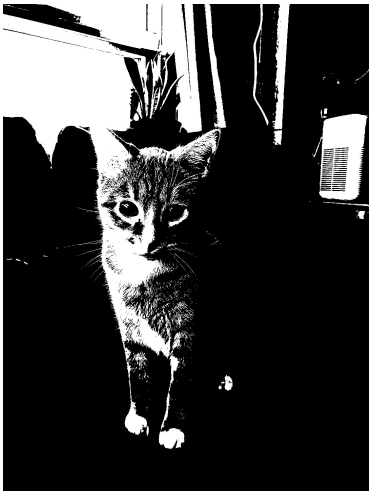


(d) CM -> Dilation

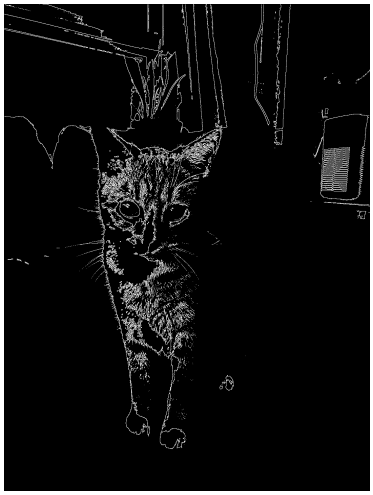


(e) CM -> Dilation -> Erosion

Figure 19: Opening (b and c) and Closing (d and e) Morphological Operations



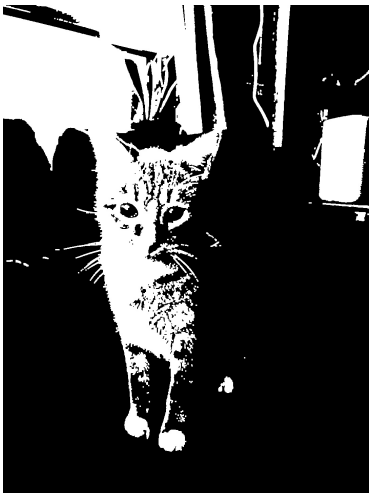
(a) Combined Mask (CM)



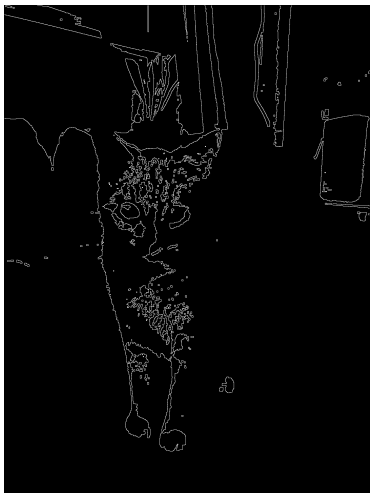
(b) Combined Mask Contour

(c) Opening Mask

(d) Contour after Opening

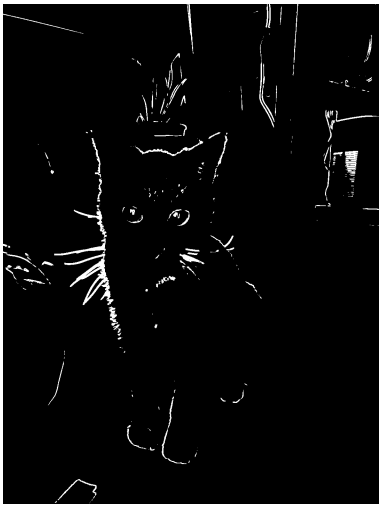


(e) Closing Mask

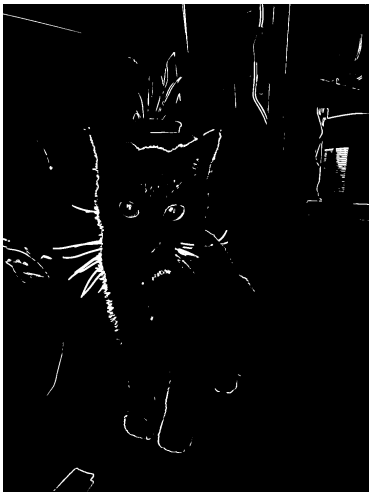


(f) Contour after Closing

Figure 20: Final Contours for different Masks



(a) Combined Mask (CM) from Texture Segmentation



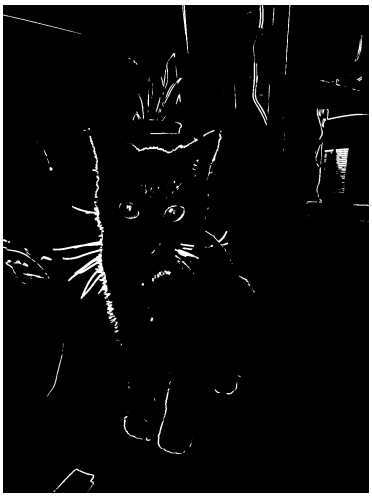
(b) CM -> Erosion

(c) CM -> Erosion -> Dilation

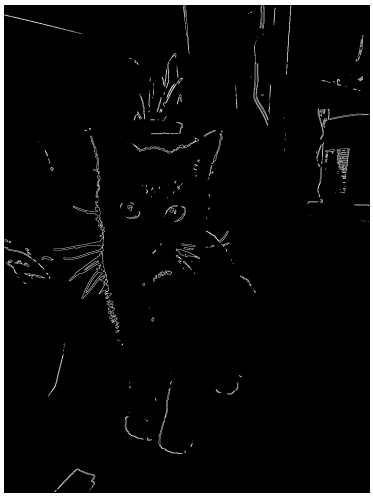
(d) CM -> Dilation

(e) CM -> Dilation -> Erosion

Figure 21: Opening (b and c) and Closing (d and e) Morphological Operations



(a) Combined Mask (CM)



(b) Combined Mask Contour

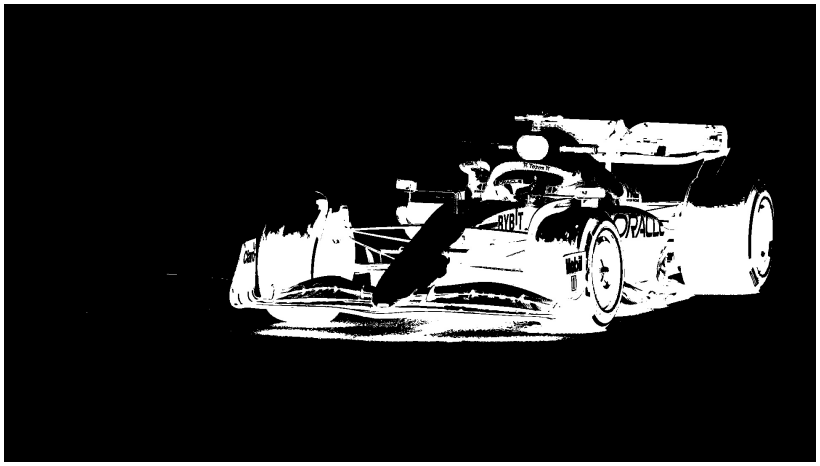
(c) Opening Mask

(d) Contour after Opening

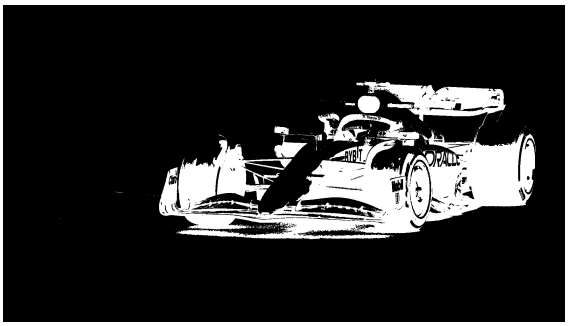
(e) Closing Mask

(f) Contour after Closing

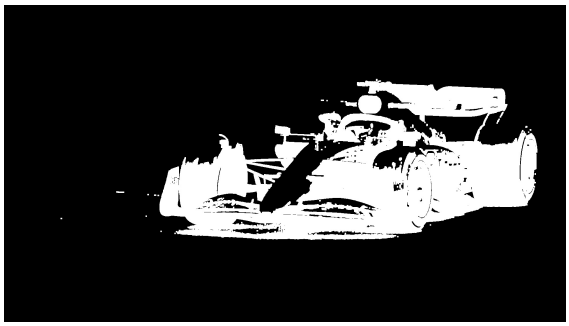
Figure 22: Final Contours for different Masks



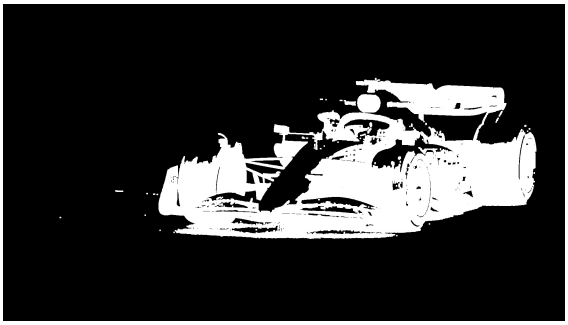
(a) Combined Mask (CM) from BGR Segmentation



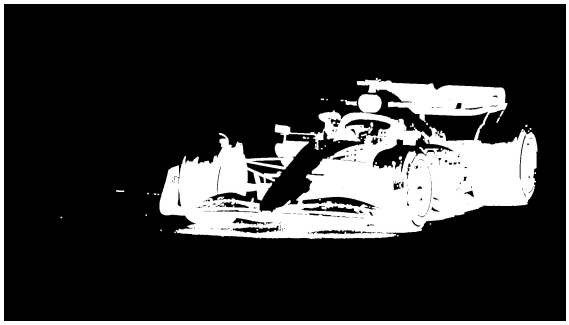
(b) CM -> Erosion



(c) CM -> Erosion -> Dilation

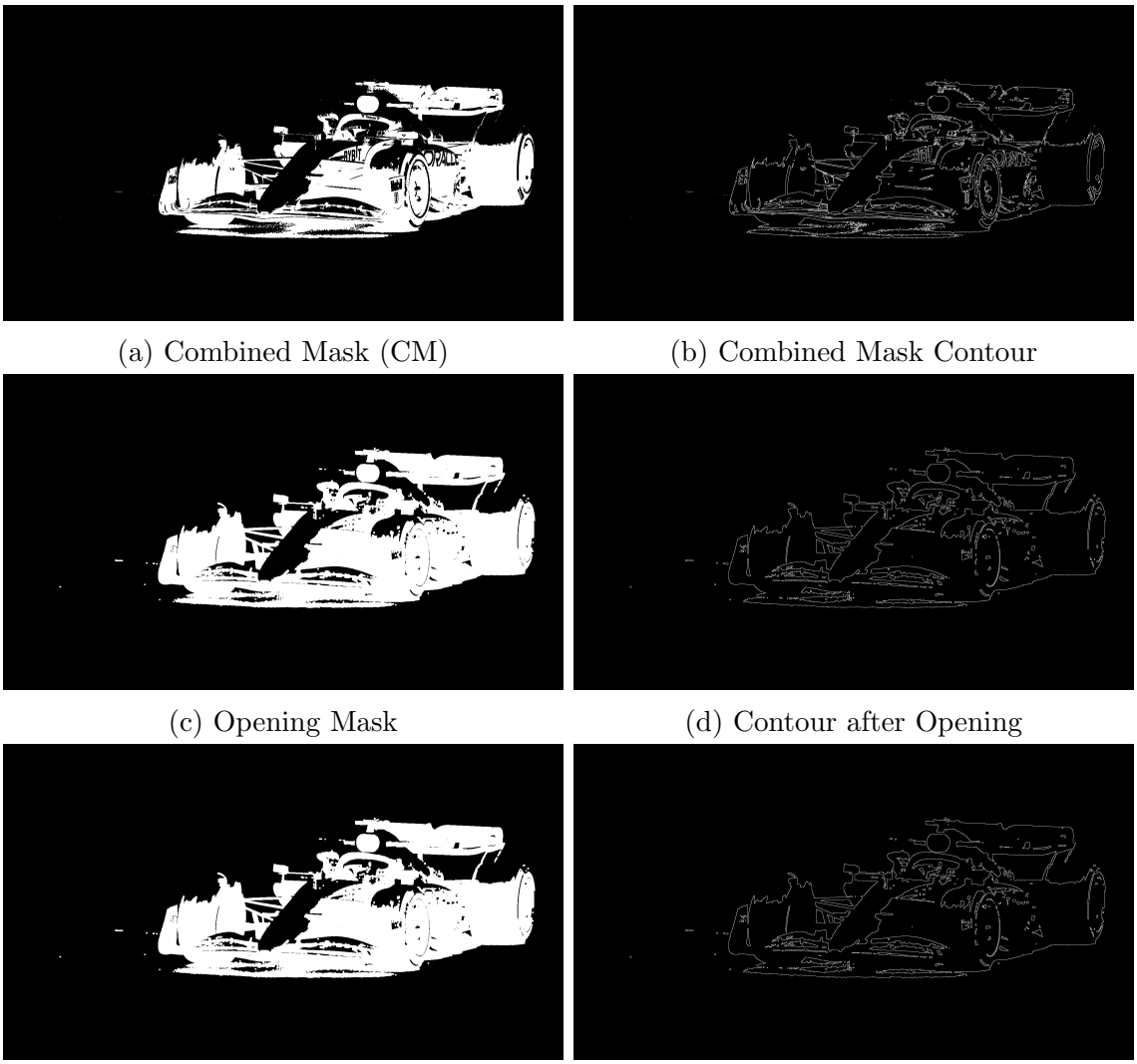


(d) CM -> Dilation



(e) CM -> Dilation -> Erosion

Figure 23: Opening (b and c) and Closing (d and e) Morphological Operations



(a) Combined Mask (CM)

(b) Combined Mask Contour

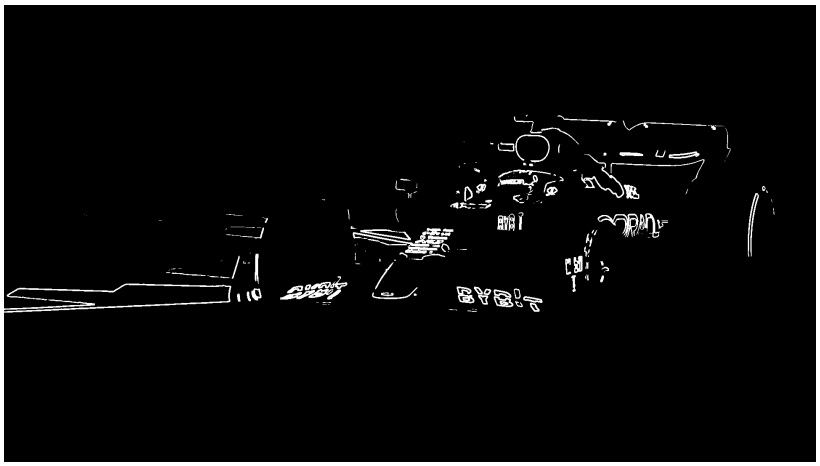
(c) Opening Mask

(d) Contour after Opening

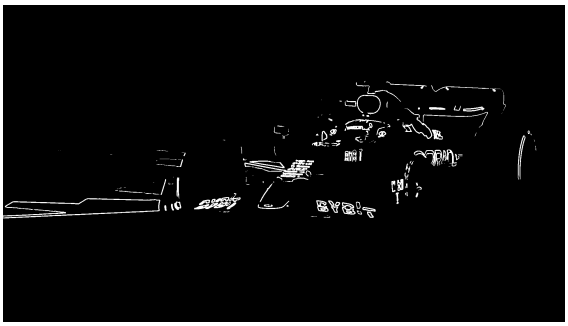
(e) Closing Mask

(f) Contour after Closing

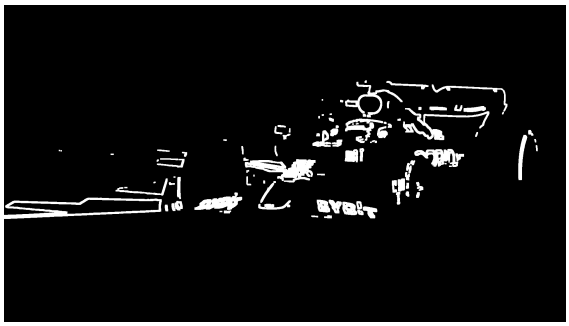
Figure 24: Final Contours for different Masks



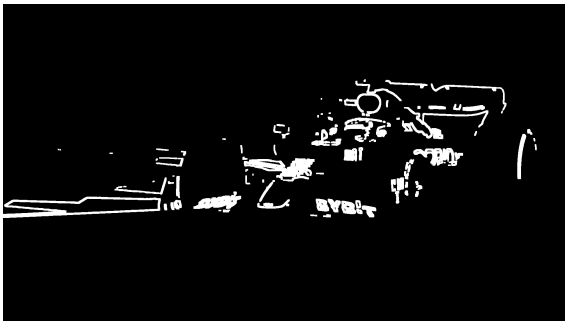
(a) Combined Mask (CM) from Texture Segmentation



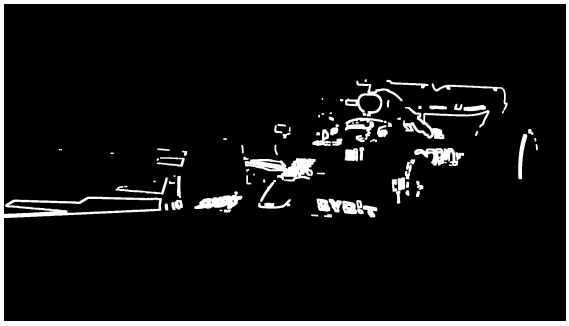
(b) CM -> Erosion



(c) CM -> Erosion -> Dilation

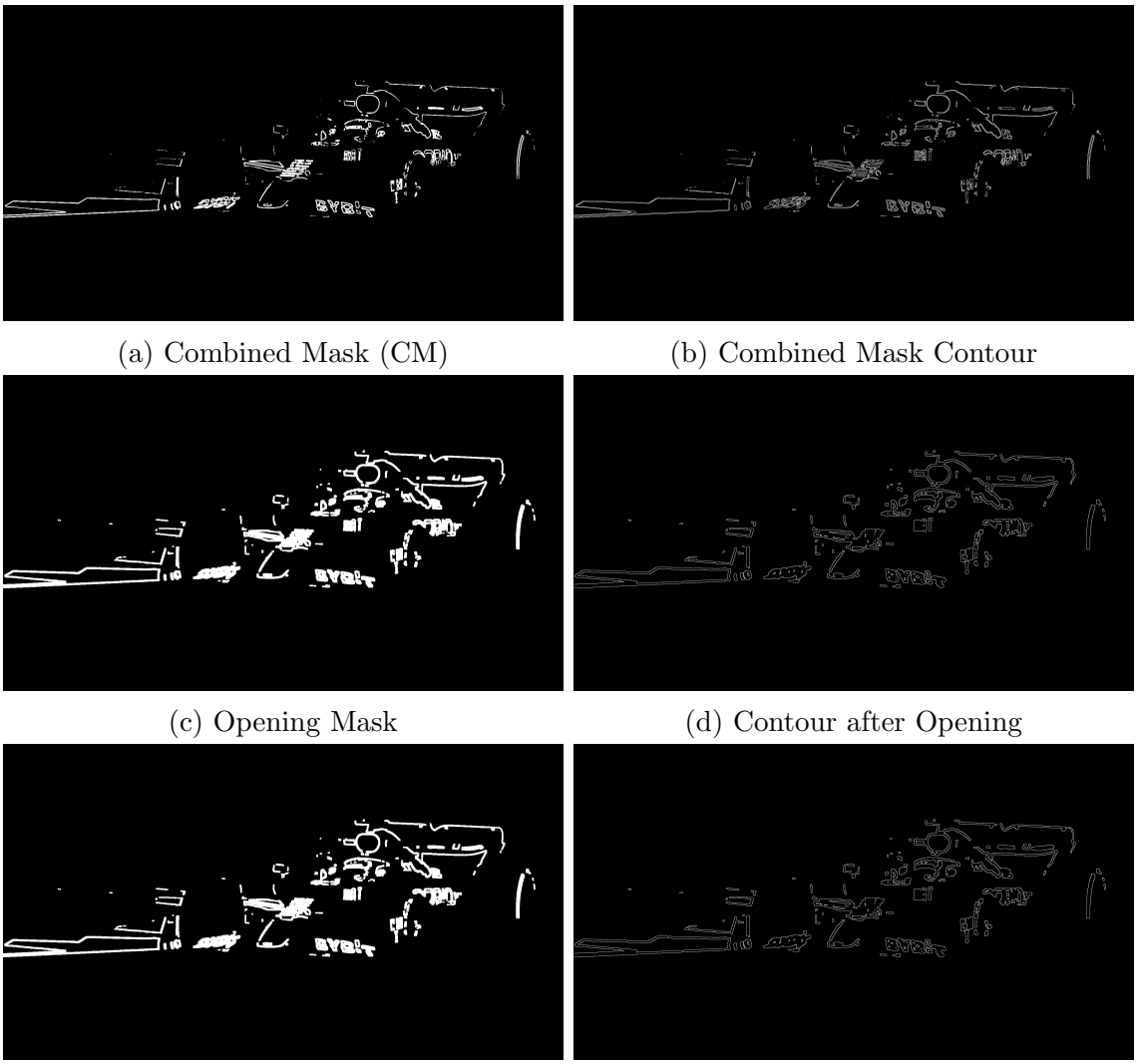


(d) CM -> Dilation



(e) CM -> Dilation -> Erosion

Figure 25: Opening (b and c) and Closing (d and e) Morphological Operations



(a) Combined Mask (CM)

(b) Combined Mask Contour

(c) Opening Mask

(d) Contour after Opening

(e) Closing Mask

(f) Contour after Closing

Figure 26: Final Contours for different Masks

Parameters:
Image: Cat
Erosion Size: 1
Erosion iterations: 1
Dilation Size: 3
Dilation Iterations: 2

Image: Car
Erosion Size: 1
Erosion Iterations: 1
Dilation Size: 3
Dilation Iterations: 2

Observation

1. For each image, parameters were tuned to get the best results
2. In cases when the combined mask was dense, performing erosion followed by dilation helped clean the foreground mask and get a better contour.
3. In cases when the combined was not dense, performing erosion led to lose of details, and hence dilation was performed first followed by erosion to get a better result.
4. As can be seen, sometimes BGR Segmentation performed better than Texture Segmentation at separating the foreground from the background.
5. If the image consisted of colors such as cyan, pink and magenta, the BGR segmentation failed to separate the colors properly.

Source Code

```
import numpy as np
import cv2
import math

#--------------------------Functions-----------------------------#
def RGBImageSegmentation(image, iters, flip, imgName):
    img = image.copy()

    # Get RGB layers of the img
    channels = cv2.split(img) # B - G - R
    # Create Masks array to store mask for each of the 3 layers
    masks = []
    mask_all = np.ones(channels[0].shape).astype(np.uint8)
    for c, channel_data in enumerate(channels):
        # Update mask
        mask = getMask(channel_data, iters[c], flip[c])
        masks.append(mask)
        mask_all = np.logical_and(mask_all, mask)
    # Foreground with blue, gree, red mask
    mask_blue = np.uint8(masks[0]*255)
    mask_green = np.uint8(masks[1]*255)
    mask_red = np.uint8(masks[2]*255)
    # Foreground with merged masks
    mask_comb_img = np.uint8(mask_all*255)

    # Save Images
    cv2.imwrite('HW6/'+imgName+'_blue_img.jpeg', channels[0])
    cv2.imwrite('HW6/'+imgName+'_green_img.jpeg', channels[1])
    cv2.imwrite('HW6/'+imgName+'_red_img.jpeg', channels[2])

    cv2.imwrite('HW6/'+imgName+'_blue_mask.jpeg', mask_blue)
    cv2.imwrite('HW6/'+imgName+'_green_mask.jpeg', mask_green)
    cv2.imwrite('HW6/'+imgName+'_red_mask.jpeg', mask_red)
```

```
    cv2.imwrite('HW6/'+imgName+'_mask_all.jpeg', mask_comb_img)

    return mask_blue, mask_green, mask_red, mask_all


def getMask(img_data, iters, flip):
    data = img_data.flatten()

    for i in range(iters):
        # Get the threshold using Otsu's Algorithm
        threshold = OtsuThresh(data)

        # Create a mask to update
        mask = np.zeros(img_data.shape, dtype = np.uint8)

        # Using the threshold, update the mask
        if flip:
            mask[img_data > threshold] = 1
            data = [i for i in data if i>threshold]
        else:
            mask[img_data <= threshold] = 1
            data = [i for i in data if i<threshold]

        data = np.asarray(data)
    return mask

def OtsuThresh(img):
    # get histogram
    hist, bins = np.histogram(img, bins = 256, range = (0,256))
    sum_t = sum(hist*bins[:-1])
    back = 0
    sumback = 0
    best_fn = -1
    threshold = 0
    for i in range(256):
        back, fore = sum(hist[:i]), sum(hist[i+1:])
        if back == 0 or fore == 0:
            continue
        sumback += i*hist[i]
        sumfore = np.int32(sum_t - sumback)
        fn = back*fore*(sumback/back - sumfore/fore)**2
        if fn>=best_fn:
            threshold = i
            best_fn = fn

    print(threshold)
    return threshold

def textureSegmentation(image, window, iters, flip, imgName):
    img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    layers = []
    masks = []
    mask_all = np.ones(img.shape).astype(np.uint8)
    for i, N in enumerate(window):
```

```
        layer = getTexturelayer(img, N)
        layers.append(layer)
        mask = getMask(layer, iters[i], flip[i])
        masks.append(mask)
        mask_all = np.logical_and(mask_all, mask)
    # Foreground with blue, gree, red mask
    mask_1 = np.uint8(masks[0]*255)
    mask_2 = np.uint8(masks[1]*255)
    mask_3 = np.uint8(masks[2]*255)
    # Foreground with merged masks
    mask_comb_img = np.uint8(mask_all*255)

    # Save Images
    cv2.imwrite('HW6/'+imgName+'_textLayer1.jpeg', layers[0])
    cv2.imwrite('HW6/'+imgName+'_textLayer2.jpeg', layers[1])
    cv2.imwrite('HW6/'+imgName+'_textLayer3.jpeg', layers[2])
    cv2.imwrite('HW6/'+imgName+'_layer1Mask.jpeg', mask_1)
    cv2.imwrite('HW6/'+imgName+'_layer2Mask.jpeg', mask_2)
    cv2.imwrite('HW6/'+imgName+'_layer3Mask.jpeg', mask_3)
    cv2.imwrite('HW6/'+imgName+'_textureMask_all.jpeg', mask_comb_img)

    return mask_1, mask_2, mask_3, mask_all

def getTexturelayer(img, N):
    varLayer = np.zeros(img.shape)
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            mid = int((N-1)/2)
            u = max(0, y-mid)
            d = min(img.shape[0], y + mid + 1)
            l = max(0, x-mid)
            r = min(img.shape[1], x + mid + 1)
            window = img[u:d, l:r]
            varLayer[y][x] = np.var(window)
    varLayer = np.uint8(np.round(255 * varLayer / (np.max(varLayer)-np.min
                                        (varLayer)) ))
    return varLayer

def opening(imgName, mask, erosion, dilation, eros_size, eros_iter,
                                    dil_size, dil_iter):
     # Use dilation and Erosion to clean the mask
    if erosion:
        k = np.ones((eros_size,eros_size), np.uint8)
        mask = cv2.erode(np.float32(mask), k, iterations=eros_iter)
        # cv2.imshow('frame1', mask)
        # cv2.waitKey(0)
        # cv2.destroyAllWindows()
        cv2.imwrite('HW6/'+imgName+'_opening_eros_img.jpeg', mask*255)
    if dilation:
        k = np.ones((dil_size, dil_size), np.uint8)
        mask = cv2.dilate(np.float32(mask), k, iterations=dil_iter)
        # cv2.imshow('frame2', mask)
        # cv2.waitKey(0)
        # cv2.destroyAllWindows()
```

```
        cv2.imwrite('HW6/'+imgName+'_opening_dil_eros_img.jpeg', mask*255)
    return mask

def closing(imgName, mask, erosion, dilation, eros_size, eros_iter,
                                        dil_size, dil_iter):
     # Use dilation and Erosion to clean the mask
    if dilation:
        k = np.ones((dil_size, dil_size), np.uint8)
        mask = cv2.dilate(np.float32(mask), k, iterations=dil_iter)
        # cv2.imshow('frame2', mask)
        # cv2.waitKey(0)
        # cv2.destroyAllWindows()
        cv2.imwrite('HW6/'+imgName+'_closing_dilation_img.jpeg', mask*255)
    if erosion:
        k = np.ones((eros_size,eros_size), np.uint8)
        mask = cv2.erode(np.float32(mask), k, iterations=eros_iter)
        # cv2.imshow('frame1', mask)
        # cv2.waitKey(0)
        # cv2.destroyAllWindows()
        cv2.imwrite('HW6/'+imgName+'_closing_eros_dil_img.jpeg', mask*255)
    return mask

def getContour(imgName, mask):
    contour = np.zeros(mask.shape, dtype = np.uint8)
    for y in range(1,mask.shape[0]-1):
        for x in range(1,mask.shape[1]-1):
            if mask[y][x] == 0:
                continue
            window = mask[y-1:y+2, x-1:x+2]
            if sum(window.flatten())<9:
                contour[y][x] = 1
    contour_img = np.uint8(contour*255)

    # Save Image
    cv2.imwrite('HW6/'+imgName+'_contour_img.jpeg', contour_img)

    return contour_img


#-----------------------------Task 1-----------------------------#

# Input Images
cat1 = cv2.imread('HW6/cat.jpg')
car1 = cv2.imread('HW6/car.jpg')
cat2 = cv2.imread('HW6/cat2.jpg')
car2 = cv2.imread('HW6/car2.jpg')


#----------------Task 1-------------------#

#----------Cat1 Image-----------#

# BGR Segmentation and Contouring
```

```
blue_mask, green_mask, red_mask, mask_all_bgrSeg = RGBImageSegmentation(
                                    cat1, iters = [1,1,1], flip = [0,0,1]
                                    , imgName="cat1")
mask_all_opening = closing("cat1_bgrSeg", mask_all_bgrSeg, erosion=1,
                                    dilation=1, eros_size=2, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all_closing = opening("cat1_bgrSeg", mask_all_bgrSeg, erosion=1,
                                    dilation=1, eros_size=2, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all = getContour("cat1_bgrSeg", mask_all_bgrSeg)
mask_all = getContour("cat1_bgrSeg_opening", mask_all_opening)
mask_all = getContour("cat1_bgrSeg_closing", mask_all_closing)

# Texture Segmentation and Contouring
layer3Mask, layer5Mask, layer7Mask, textureMask_all = textureSegmentation(
                                    cat1, window = [3,5,7], iters = [1,1,
                                    1], flip = [0,1,1], imgName="cat1")
mask_all_closing = closing("cat1_texture", textureMask_all, erosion=1,
                                    dilation=1, eros_size=2, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all_opening = opening("cat1_texture", textureMask_all, erosion=1,
                                    dilation=1, eros_size=2, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all = getContour("cat1_texture", textureMask_all)
mask_all = getContour("cat1_texture_opening", mask_all_opening)
mask_all = getContour("cat1_texture_closing", mask_all_closing)


#----------Car1 Image-----------#

# BGR Segmentation and Contouring
blue_mask, green_mask, red_mask, mask_all_seg = RGBImageSegmentation(car1,
                                     iters = [1,1,1], flip=[0,1,0],
                                    imgName="car1")
mask_all_closing = closing("car1_bgrSeg", mask_all_seg, erosion=1,
                                    dilation=1, eros_size=1, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all_opening = opening("car1_bgrSeg", mask_all_seg, erosion=1,
                                    dilation=1, eros_size=1, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all = getContour("car1_bgrSeg", mask_all_seg)
mask_all = getContour("car1_bgrSeg_opening", mask_all_opening)
mask_all = getContour("car1_bgrSeg_closing", mask_all_closing)

layer1Mask, layer2Mask, layer3Mask, textureMask_all = textureSegmentation(
                                    car1, window = [3,5,7], iters = [1,1,
                                    1], flip = [0,0,1], imgName="car1")
mask_all_closing = closing("car1_texture", textureMask_all, erosion=1,
                                    dilation=1, eros_size=1, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all_opening = opening("car1_texture", textureMask_all, erosion=1,
                                    dilation=1, eros_size=1, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all = getContour("car1_texture", textureMask_all)
```

```
mask_all = getContour("car1_texture_closing", mask_all_closing)
mask_all = getContour("car1_texture_opening", mask_all_opening)


#----------------Task 2-------------------#

#----------Cat2 Image-----------#

# BGR Segmentation and Contouring
blue_mask, green_mask, red_mask, mask_all_bgrSeg = RGBImageSegmentation(
                                    cat2, iters = [1,1,2], flip = [1,1,1]
                                    , imgName="cat2")
mask_all_opening = closing("cat2_bgrSeg", mask_all_bgrSeg, erosion=1,
                                    dilation=1, eros_size=2, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all_closing = opening("cat2_bgrSeg", mask_all_bgrSeg, erosion=1,
                                    dilation=1, eros_size=2, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all = getContour("cat2_bgrSeg", mask_all_bgrSeg)
mask_all = getContour("cat2_bgrSeg_opening", mask_all_opening)
mask_all = getContour("cat2_bgrSeg_closing", mask_all_closing)

# Texture Segmentation and Contouring
layer3Mask, layer5Mask, layer7Mask, textureMask_all = textureSegmentation(
                                    cat2, window = [5,7,9], iters = [1,1,
                                    1], flip = [1,1,1], imgName="cat2")
mask_all_closing = closing("cat2_texture", textureMask_all, erosion=1,
                                    dilation=1, eros_size=2, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all_opening = opening("cat2_texture", textureMask_all, erosion=1,
                                    dilation=1, eros_size=1, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all = getContour("cat2_texture", textureMask_all)
mask_all = getContour("cat2_texture_opening", mask_all_opening)
mask_all = getContour("cat2_texture_closing", mask_all_closing)


#----------Car2 Image-----------#
# BGR Segmentation and Contouring
blue_mask, green_mask, red_mask, mask_all_seg = RGBImageSegmentation(car2,
                                     iters = [2,2,2], flip=[0,0,0],
                                    imgName="car2")
mask_all_closing = closing("car2_bgrSeg", mask_all_seg, erosion=1,
                                    dilation=1, eros_size=1, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all_opening = opening("car2_bgrSeg", mask_all_seg, erosion=1,
                                    dilation=1, eros_size=1, eros_iter=1,
                                     dil_size=3, dil_iter=2)
mask_all = getContour("car2_bgrSeg", mask_all_seg)
mask_all = getContour("car2_bgrSeg_opening", mask_all_opening)
mask_all = getContour("car2_bgrSeg_closing", mask_all_closing)

# Texture Segmentation and Contouring
```

```
layer1Mask, layer2Mask, layer3Mask, textureMask_all = textureSegmentation(
                                        car2, window = [5,7,9], iters = [1,1,
                                        1], flip = [1,1,1], imgName="car2")
mask_all_closing = closing("car2_texture", textureMask_all, erosion=1,
                                        dilation=1, eros_size=1, eros_iter=1,
                                         dil_size=3, dil_iter=2)
mask_all_opening = opening("car2_texture", textureMask_all, erosion=1,
                                        dilation=1, eros_size=1, eros_iter=1,
                                         dil_size=3, dil_iter=2)
mask_all = getContour("car2_texture", textureMask_all)
mask_all = getContour("car2_texture_closing", mask_all_closing)
mask_all = getContour("car2_texture_opening", mask_all_opening)
```