# Capstone Design

### Assignment OpenCV basics

Task explanations:



gray\_car.png (given original image)

1. From image gray\_car.png manually find the position of an SUV car and draw a rectangle as follows:

import cv2

# task 1: draw a rectangle

original = cv2.imread('gray\_car.png')

image = cv2.imread('gray\_car.png')

image = cv2.rectangle(image, (215, 104), (365, 245), (0, 0, 255), 3)

cv2.imwrite('task1.png', image)

In this task, the first thing I did was import the OpenCV library by import cv2

Then loading the given gray\_car.png from a local drive by

original = cv2.imread('gray\_car.png')

The reason why I loaded the same image twice and assigned them to different variables was after drawing a rectangle to the image, I cannot access the original image later for completing other tasks.

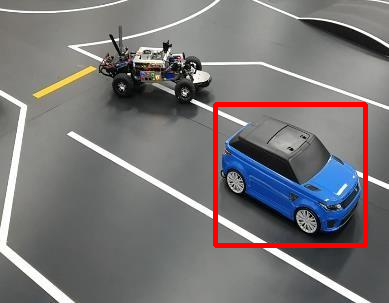
Before drawing the rectangle to the image, I found the top-left and bottom-right pixels where the SUV car was located in a photo editing tool. Then, I can draw the rectangle with these pixels by

image = cv2.rectangle(image, (215, 104), (365, 245), (0, 0, 255), 3)

The first parameter is the path of an image to draw the rectangle, followed by top-left and bottom-right pixels, then the color of the rectangle border and thickness.

Lastly, I saved the resulted image:

cv2.imwrite('task1.png', image)



task1.png

1. Using ROI separate gray\_car.png image into two different images as follows:

Continuation of the previous code to complete this task:

# task 2: separate two cars (using ROI)

cv2.imwrite('task2\_car1.png', original[104:245, 215:365])

cv2.imwrite('task2\_car2.png', original[18:110, 100:212])

To set ROI on images I used the following syntax where the first set of pixels for Y axes and others for X axes:

original[104:245, 215:365]

Here are the cropped images:

task2\_car1.png task2\_car2.png

1. Create blurred and sharpen images from ***gray\_car.png*** image. Save resulted images

For this task, we need the NumPy library to work with arrays:

import numpy as np

. . .

# task 3: create blurred and sharpened images

size = 15

kernel\_motion\_blur = np.zeros((size, size))

kernel\_motion\_blur[int((size-1)/2), :] = np.ones(size)

kernel\_motion\_blur = kernel\_motion\_blur / size

kernel\_sharpen = np.array(

[[-1, -1, -1],

[-1, 9, -1],

[-1, -1, -1]])

blurred = cv2.filter2D(original, -1, kernel\_motion\_blur)

sharpened = cv2.filter2D(original, -1, kernel\_sharpen)

cv2.imwrite('task3\_blurred.png', blurred)

cv2.imwrite('task3\_sharpened.png', sharpened)

The first thing I did was generate the kernels, starting with blurring. The following code will generate a 15 x 15 array of zeros:

size = 15

kernel\_motion\_blur = np.zeros((size, size))

Then I made the 7th row of the array all 1s and divide the array by 15 in order not to make the complete blur:

kernel\_motion\_blur[int((size-1)/2), :] = np.ones(size)

kernel\_motion\_blur = kernel\_motion\_blur / size

The kernel for sharpening is as follows:

kernel\_sharpen = np.array(

[[-1, -1, -1],

[-1, 9, -1],

[-1, -1, -1]])

Lastly, I applied the filters to the original image and save it to local disk:

blurred = cv2.filter2D(original, -1, kernel\_motion\_blur)

sharpened = cv2.filter2D(original, -1, kernel\_sharpen)

cv2.imwrite('task3\_blurred.png', blurred)

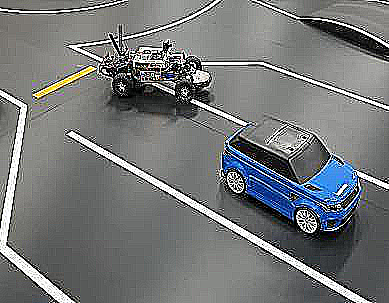
cv2.imwrite('task3\_sharpened.png', sharpened)

Here are the parameters for cv2.filter2D(src, ddepth, kernel)

* **src**: The source image on which to apply the filter. It is a matrix that represents the image in pixel intensity values.
* **ddepth**: It is the **desired depth** of the destination image. Value -1 represents that the resulting image will have the same depth as the source image.
* **kernel**: kernel is the filter matrix applied to the image.



task3\_blurred.png



task3\_sharpened.png

1. Using histogram equalization increase the brightness of the ***gray\_car.png*** image. Save resulted images

Here, each channel of the R, G, and B represents the intensity of the related color, not the intensity/brightness of the image as a whole. And so, running HE on these color channels is not the proper way. So, I have to convert the colorspace from RGB to YCbCr >> Run HE on the Y channel by equalizing it (this channel represents brightness) >> Convert back the colorspace to RGB:

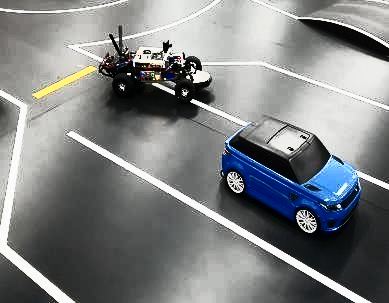
# task 4: increase brightness

img\_yuv = cv2.cvtColor(original, cv2.COLOR\_BGR2YUV)

img\_yuv[:, :, 0] = cv2.equalizeHist(img\_yuv[:, :, 0])

brighter = cv2.cvtColor(img\_yuv, cv2.COLOR\_YUV2BGR)

cv2.imwrite('task4.png', brighter)



task4.png