Future Vehicle Education Workshop

Subject: Computer Vision

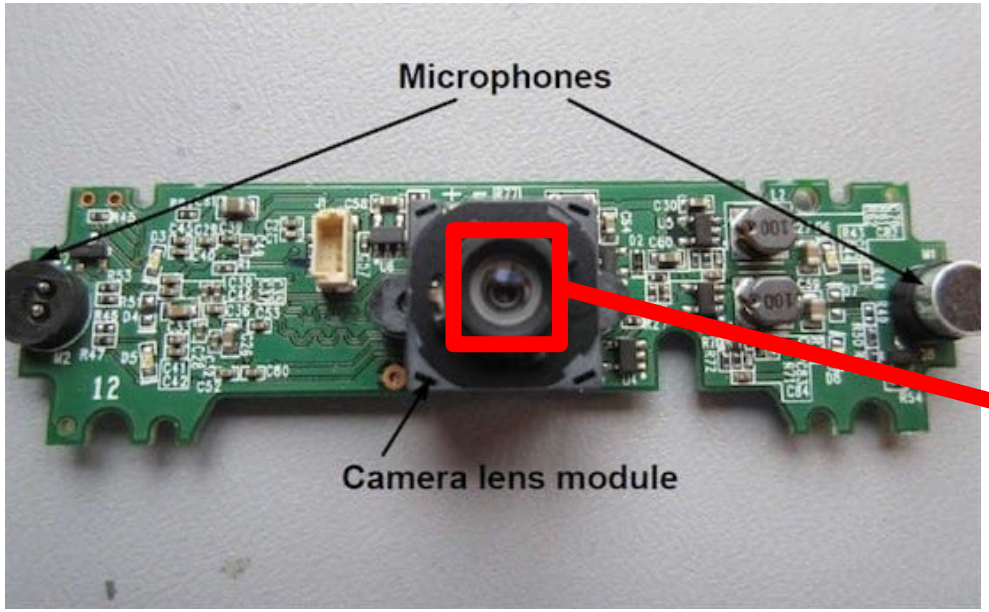
Automation Lab

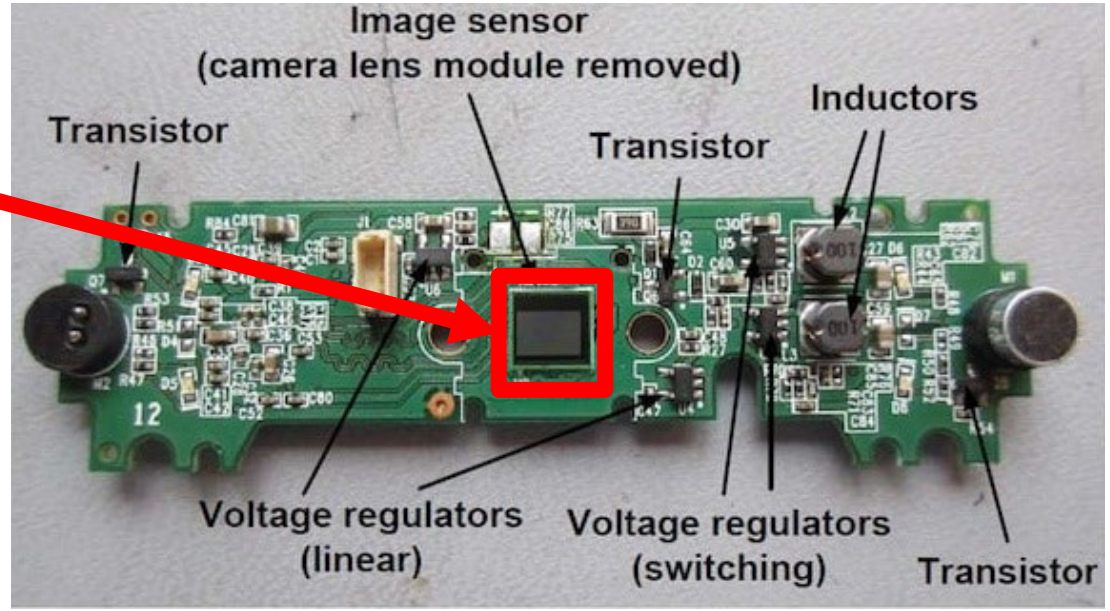
Introduction

- Logitech C920 HD PRO Webcam

Introduction

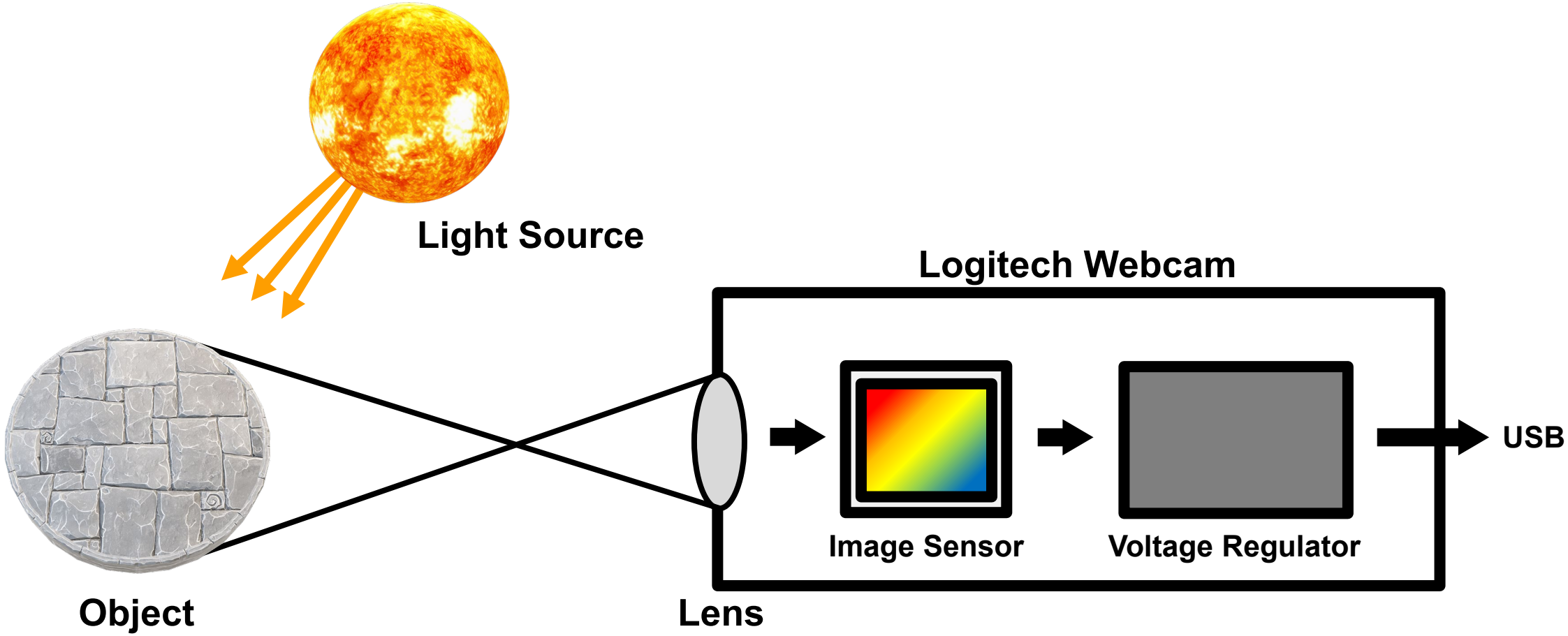
Webcam Internal Structure





Introduction

Image Formation

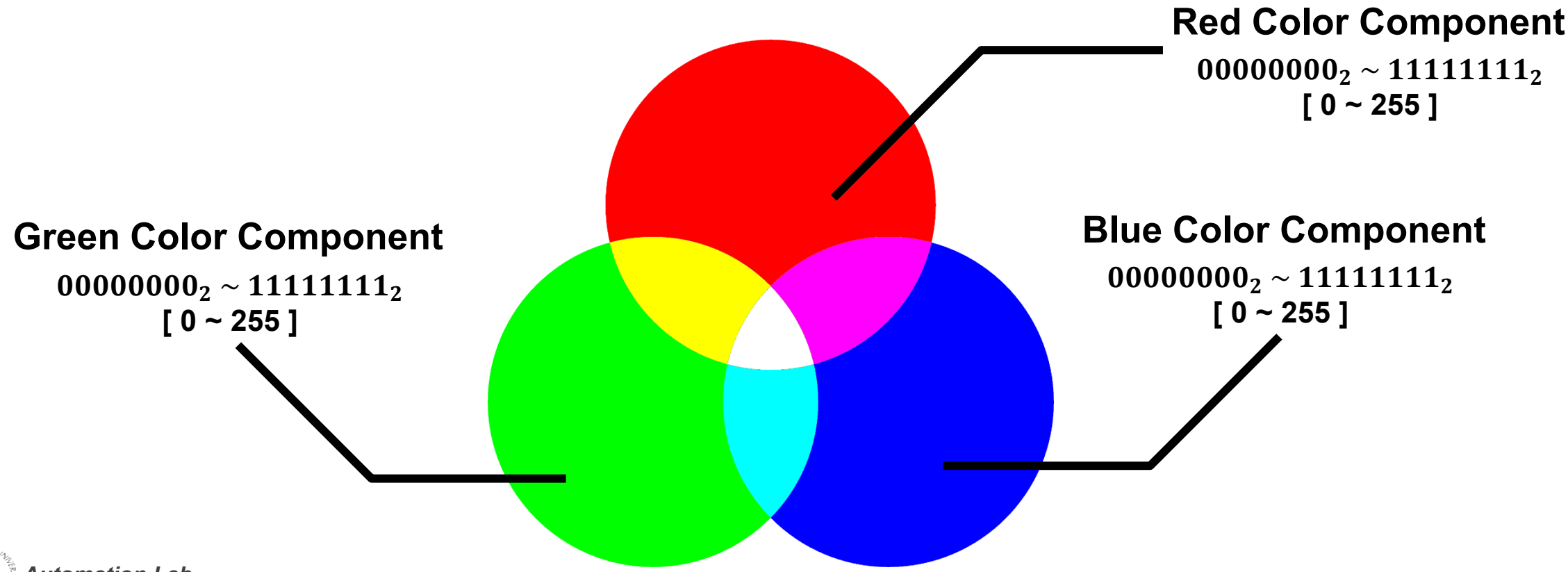


Introduction

■ 3-type Primary Color [RGB Color]

→ 8-bit Color Depth: How computers represent colors (Most Common Method)

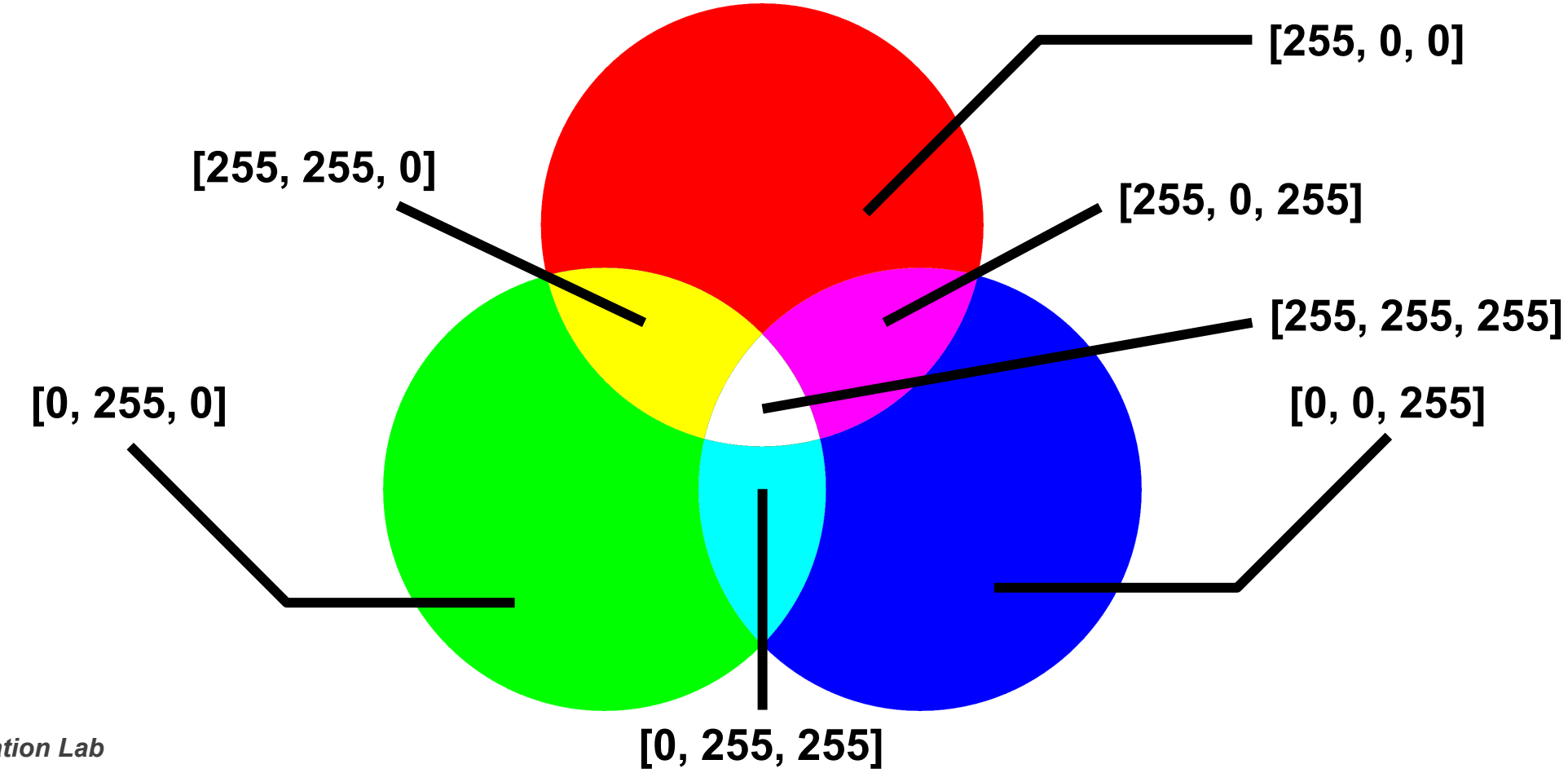
→ "256×256×256=16,777,216" Color representation possible



Introduction

■ 3-type Primary Color [RGB Color]

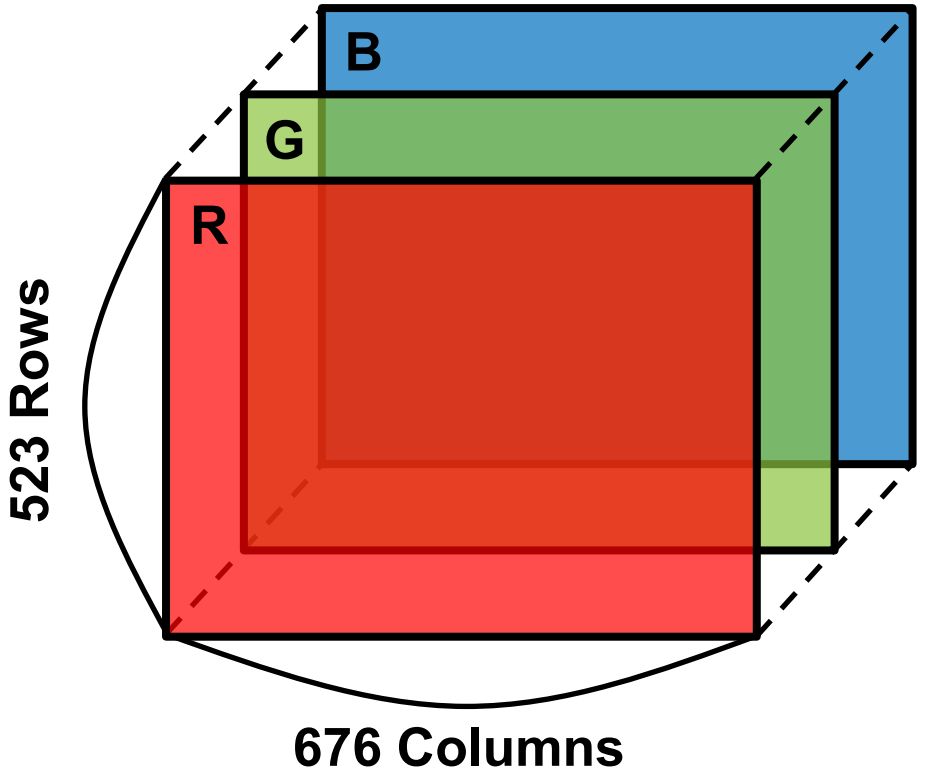
→ Digital Value allows to express multiple colors [Red, Greed, Blue]



Introduction

- RGB Image Matrix

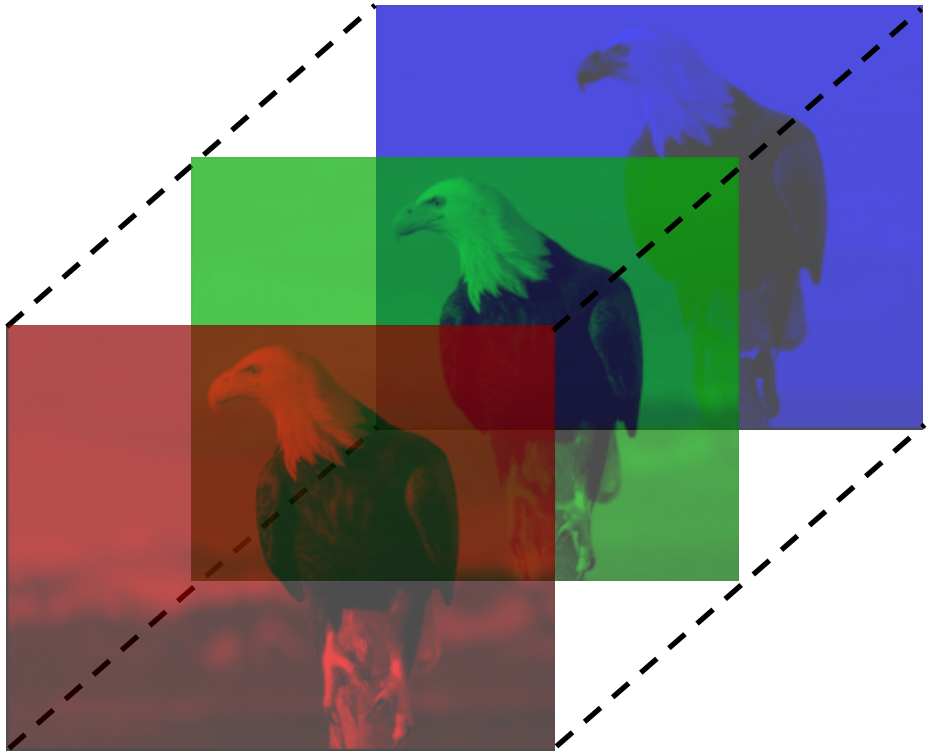
676 × 523 pixels



Introduction

RGB Image Matrix

676 × 523 pixels



Introduction

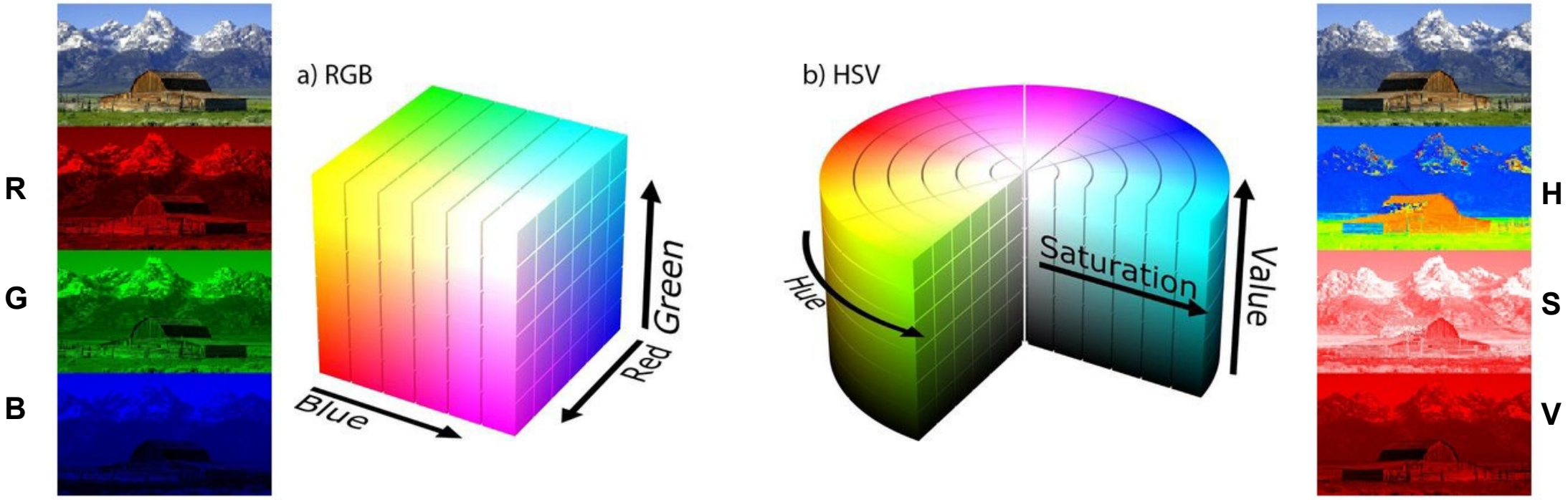
RGB Image Matrix (Python)

676 × 523 pixels

```
Image Red Component Matrix (row=0):
[145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 14
 142 142 142 143 143 143 143 143 143 143 143 145 145 145 145 145 145 14
 143 143 143 143 143 143 143 143 143 143 143 143 143 143 143 143 143 14
 143 143 143 143 143 143 143 143 143 142 142 143 143 143 143 143 143 14
 144 144 144 144 144 144 144 144 144 144 144 144 145 145 145 145 145 14
 144 144 145 145 146 146 146 145 145 145 145 144 144 144 144 144 144 14
 144 144 144 144 144 144 144 144 143 143 144 144 144 144 144 144 144 14
 144 144 144 144 144 144 145 145 145 145 144 145 145 145 145 145 145 14
 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 14
 146 146 146 146 146 146 146 146 146 146 146 146 146 145 145 145 145 14
 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 14
 144 144 144 145 145 145 145 144 144 144 144 144 144 144 144 144 144 14
 144 145 145 146 145 145 145 145 145 145 145 145 145 146 146 146 146 14
 147 147 147 147 147 147 147 147 147 147 147 147 147 147 147 147 147 14
 149 149 148 148 148 148 148 148 148 147 148 148 148 148 148 148 148 14
 147 147 147 147 147 148 148 148 148 148 148 148 148 148 149 149 150 15
 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 148 14
 146 146 146 146 146 146 146 146 146 146 146 146 146 146 146 146 146 14
 148 148 148 148 148 148 148 148 148 148]
```

HSV Color Format

HSV Color Format



HSV Color Format

■ Why prefer HSV format?

Human: Can see objects in the dark and recognize red

Computer: Computers that use RGB channels recognize only the R value slightly higher in dark images and videos. → Inaccurate method

HSV Color Format

- HSV - Hue/Saturation/Value

Original

```
h, s, v = cv2.split(hsv_img)
```

HSV H S V

```
hsv_image = cv2.merge([h, s, v])
```

Color Filtering

■ “color_filtering()” Description

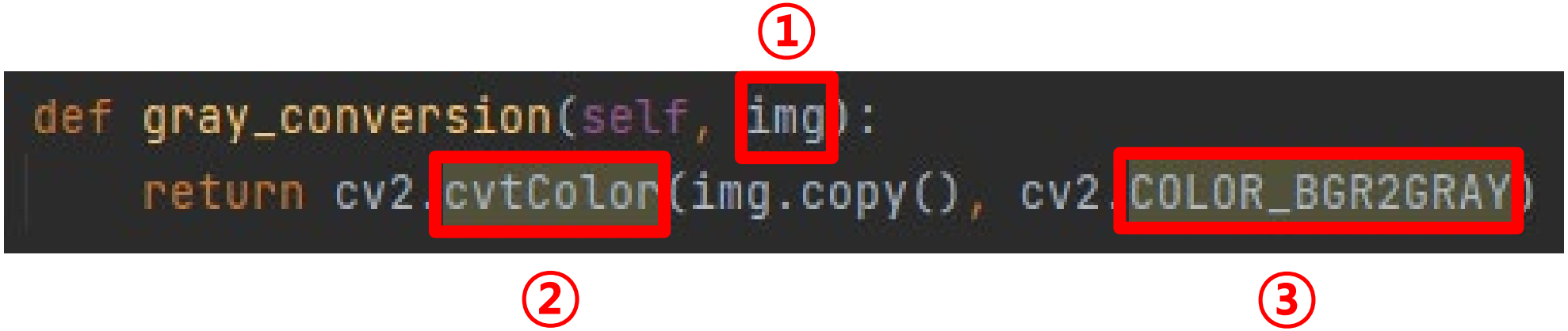
→Functions to detect the color of a traffic light

```
def color_filtering(self, img, roi=None, print_enable=False):
    self.row, self.col, self.dim = img.shape

    hsv_img = self.hsv_conversion(img)
    h, s, v = cv2.split(hsv_img)

    s_cond = s > SATURATION
    if roi is RED:
        h_cond = (h < HUE_THRESHOLD[roi][0]) | (h > HUE_THRESHOLD[roi][1])
    else:
        h_cond = (h > HUE_THRESHOLD[roi][0]) & (h < HUE_THRESHOLD[roi][1])

    v[~h_cond], v[~s_cond] = 0, 0
    hsv_image = cv2.merge([h, s, v])
    result = cv2.cvtColor(hsv_image, cv2.COLOR_HSV2BGR)

    if print_enable:
        self.image_show(result)

    return result
```

Gray Conversion

■ “gray_conversion( )” Description

→ RGB function to convert an image from a channel into a single channel, a monochrome image



① Input: Color image data (3D array) input you want to separate colors from

② Function: Use cvtColor to change the color space of an image

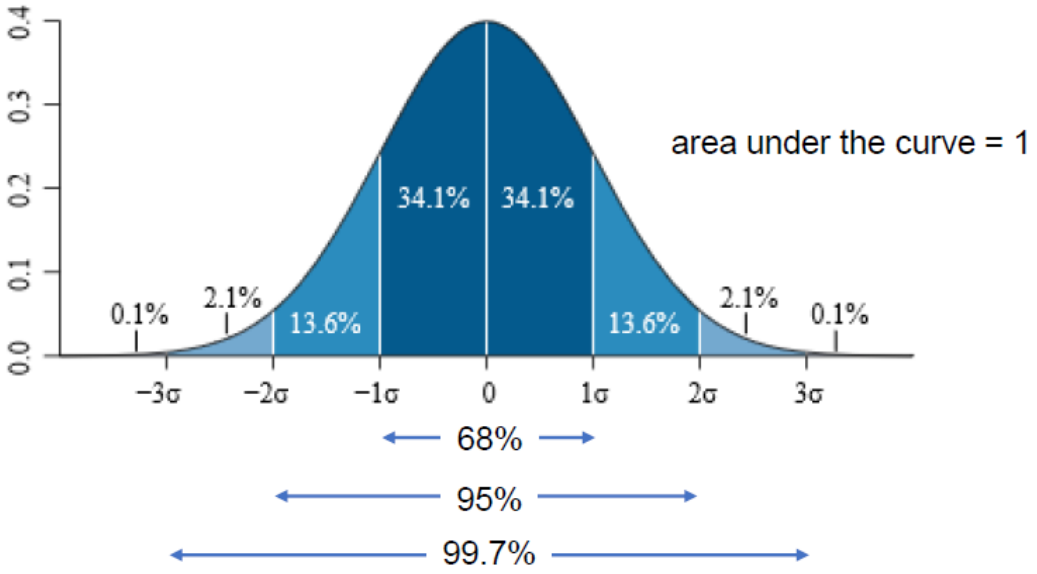
③ Output: Returning from a color image to a black and white image

※BGR: RGB color channel, and, byte Reverse order

Gaussian Blur

■ “gaussian_blurring( )” Description

→ Function to remove noise from the original image with blur effect

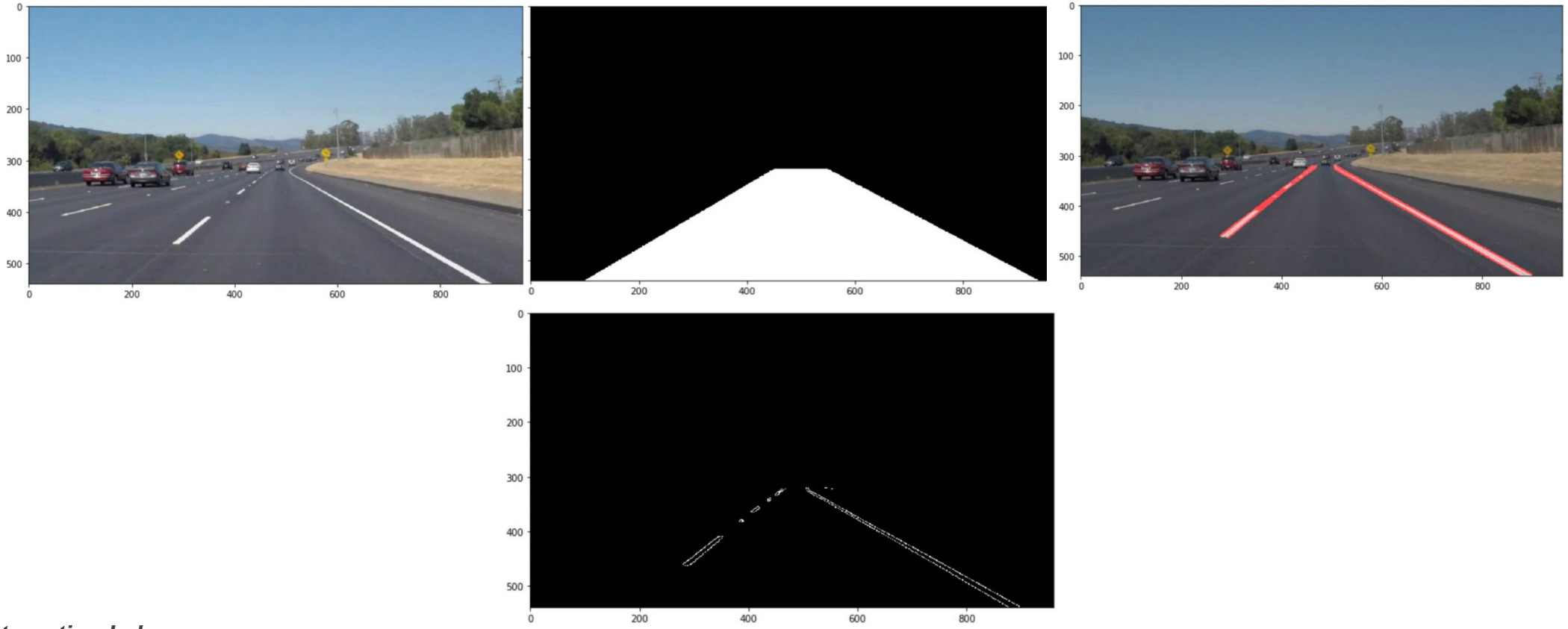


$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Canny Detection

■ “canny_edge( )” Description

→Functions to detect edges for shape recognition in hough_transform (Seperate area of interest)



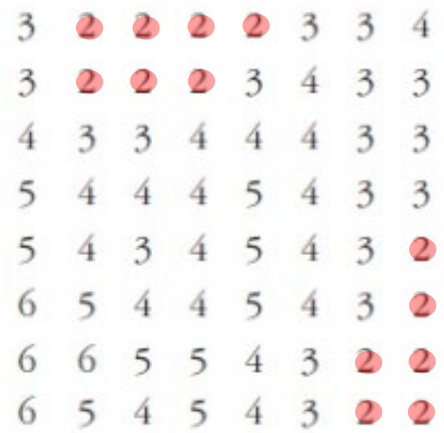
Histogram Equalization

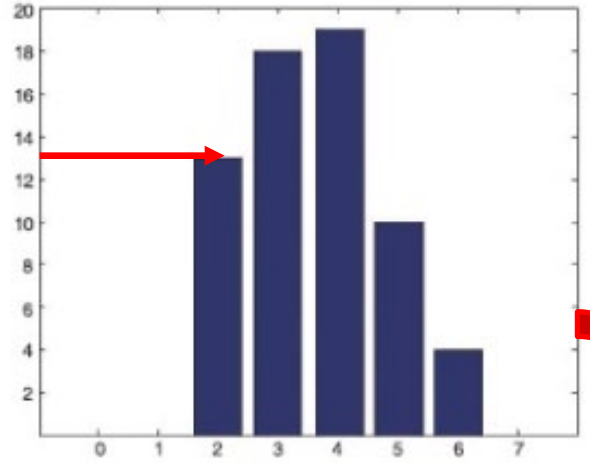
■ “histogram_equalization( )” Description

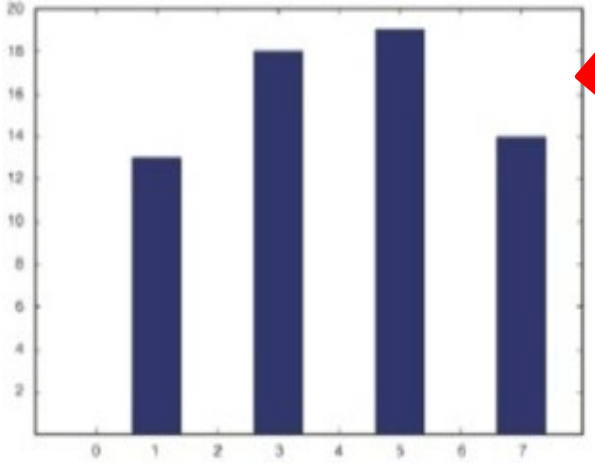
→A function that uses the cumulative number of pixel values to improve the image

원본

평활화 후







① ② ③ ④ ⑤

l_{in}	$h(l_{in})$	$c(l_{in})$	$c(l_{in}) \times 7$	l_{out}
0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0
2	0.203	0.203	1.421	1
3	0.281	0.484	3.388	3
4	0.297	0.781	5.467	5
5	0.156	0.937	6.559	7
6	0.063	1.0	7.0	7
7	0.0	1.0	7.0	7

① Gray Level은 영상의 픽셀값이다.
h = (0,0,13,18,19,10,4,0)

명암값이 2인 화소가 13개 ↔ h(2)=13

② 전체 빈도수의 합인 64로 나눠 정규화한다.
h'(2)=13/64=0.203

③ 2열의 누적합을 구한다.

④ $\frac{1}{7}$ 비율로 매핑하기 위해 곱한다.

⑤ 4열의 값을 반올림한다.
(13개의 2를 1로 매핑한다.)

Hough Transform

■ “hough_transform( )” Description

→ Functions that recognize shapes such as line or circle

```
def hough_transform(self, img, rho=None, theta=None, threshold=None, mll=None, mlg=None, mode="lineP"):
    if mode == "line":                                                          ①
        return cv2.HoughLines(img.copy(), rho, theta, threshold)
    elif mode == "lineP":                                                       ②
        return cv2.HoughLinesP(img.copy(), rho, theta, threshold, lines=np.array([]),
                               minLineLength=mll, maxLineGap=mlg)
    elif mode == "circle":                                                      ③
        return cv2.HoughCircles(img.copy(), cv2.HOUGH_GRADIENT, dp=1, minDist=80,
                                param1=200, param2=10, minRadius=40, maxRadius=100)
```

① HoughLines : The threshold is based on the point where it meets, so if it is small, it detects many lines, and if the number is large, the accuracy increases.

② HoughLinesP : **To find a straight line using random points rather than targeting all points**

③ HoughCircles - Hough_Gradient : How to detect a circle

- dp=1 : Set the same resolution as the input image
- minDist=80 : Minimum distance between detected circle
- param1=200 : **canny edge, higher threshold value / param2 : If it's too small, false circles will be detected**
- max,min radius : The minimum and maximum radius of the circle to be detected (however, if the size is unknown, it is specified as 0)

Morphology

■ Morphology

→ Function that detectreduce noise, fill holes, connect broken lines

```
def morphology(self, img, kernel_size=(None, None), mode="opening"):
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, kernel_size)

①  if mode == "opening":
        dst = cv2.erode(img.copy(), kernel)
        return cv2.dilate(dst, kernel)
②  elif mode == "closing":
        dst = cv2.dilate(img.copy(), kernel)
③       return cv2.erode(dst, kernel)
    elif mode == "gradient":
        return cv2.morphologyEx(img.copy(), cv2.MORPH_GRADIENT, kernel)
```

① (opening : erosion+dilation) Elimination of brighter noise than the surrounding area, independent object separation

② (closing : dilation+erosion) Remove noise values that are darker than the surroundings, connect broken objects, and fill holes

③ Use cv2.MORPH_GRADIENT to detect edges

Object Detection

■ “object_detection()” Description

→Functions to detect the circular shape of a traffic light

main.py

```
""" Exercise 3: Object Detection (Traffic Light Circle) """
color = env.object_detection(frame0, sample=16, print_enable=True)
```

Function_Library.py

```
for color in (RED, YELLOW, GREEN):                                    ①
    extract = self.color_filtering(img, roi=color, print_enable=True) # return : RGB 파일
    gray = self.gray_conversion(extract) # gray scale 변환
    circles = self.hough_transform(gray, mode=mode)                   ②  # mode : circle
    if circles is not None:                                           ③
        for circle in circles[0]:
            center, count = (int(circle[0]), int(circle[1])), 0
            hsv_img = self.hsv_conversion(img)
            h, s, v = cv2.split(hsv_img)
```

① color_filtering color : Red, Yellow, Green input → return RGB file

② process gray scale image to mode=="circle"

③ Calculate the median value of the hough_transform results, convert them to HSV, and separate them into h, s, and v.

Object Detection

■ “object_detection()” Description

→ Searching the surrounding pixels

```
for res in range(sample):
    x, y = int(center[1] - sample / 2), int(center[0] - sample / 2)
    s_cond = s[x][y] > SATURATION                                          ①
    if color is RED:
        h_cond = (h[x][y] < HUE_THRESHOLD[color][0]) | (h[x][y] > HUE_THRESHOLD[color][1])
        count += 1 if h_cond and s_cond else count
    else:
        h_cond = (h[x][y] > HUE_THRESHOLD[color][0]) & (h[x][y] < HUE_THRESHOLD[color][1])
        count += 1 if h_cond and s_cond else count

if count > sample / 2:
    result = COLOR[color]
    cv2.circle(replica, center, int(circle[2]), (0, 0, 255), 2)            ②
```

① Set the same as the traffic light detection function

② Draw a red circle with a line thickness of 2, the same as the circle you detected.

Edge Detection

■ “edge_detection( )” Description

→ A function that can detect a specific Edge Line to recognize the lane direction

```
def edge_detection(self, img, width=0, height=0, gap=0, threshold=0, print_enable=False):
```

① ② ③ ④ ⑤ ⑥

① Input1: Input the frame data received from the camera

② Input2: Input the maximum horizontal length of the region of interest [ROI]

③ Input3: Input the minimum vertical length of the region of interest [ROI]

④ Input4: Input the distance difference from the comparison target in pixel analysis

⑤ Input5: Input the length condition for distinguishing a specific edge line in pixel analysis

⑥ Input6: Input whether to display the image and direction value for the output result

Edge Detection

```
lines = self.hough_transform(canny, 1, np.pi/180, 50, 10, 20, mode="lineP")

if lines is not None:
    new_lines, real_lines = [], []
    for line in lines:
        xa, ya, xb, yb = line[0]

⑦      if np.abs(yb - ya) > height and np.abs(xb - xa) < width:
           if self.point_analyze(blurring, line[0], gap, threshold):
               for idx in range(len(new_lines)):
                   if np.abs(new_lines[:][idx][1] - ya) < VARIANCE:
                       if np.abs(new_lines[:][idx][3] - yb) < VARIANCE:

                           grad = (xb - xa) / -(yb - ya)

                           if np.abs(grad) < FORWARD_THRESHOLD:
⑧                              prediction = FORWARD
                           elif grad > 0:
                               prediction = RIGHT
                           elif grad < 0:
                               prediction = LEFT
```

⑦ point_analyze: Returns True if the average difference in y-values is greater than the threshold length for detecting a specific edge line.

⑧ Output: Predicts the direction of the lane based on the slope of the straight line for a specific edge line and returns the result.→ Forward: FORWARD(0) / Left: LEFT(1) / Right: RIGHT(2)

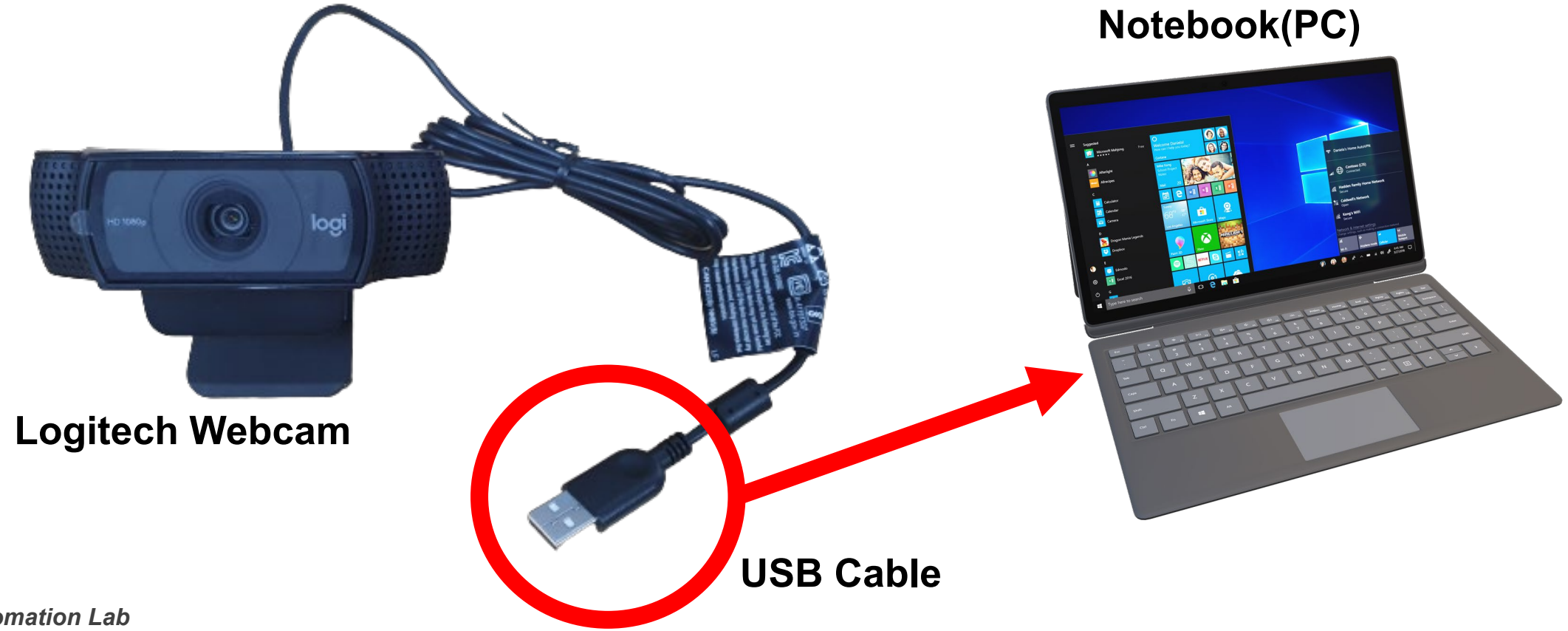
Exercise

Automation Lab.

Exercise 1

■ Webcam Hardware Setting

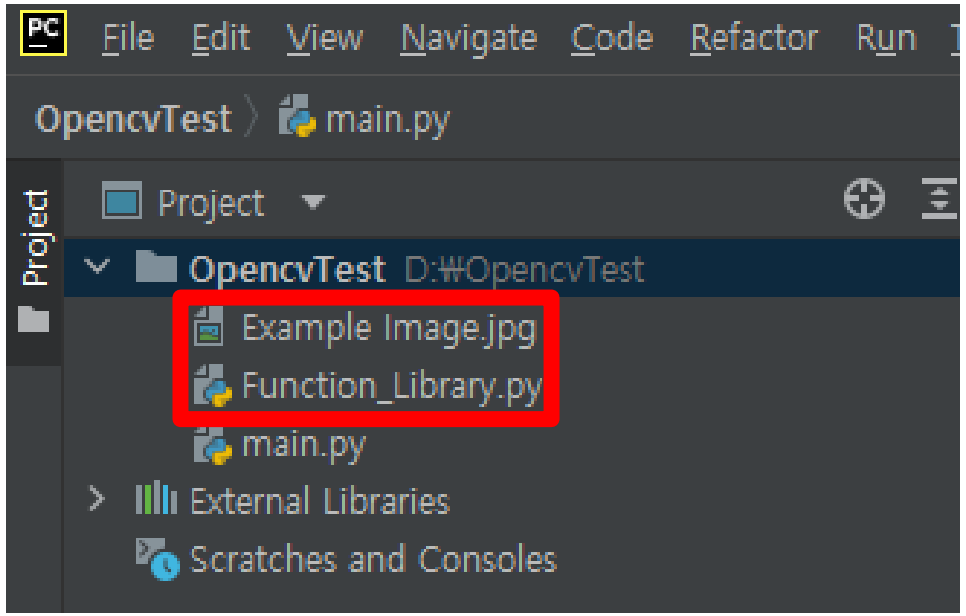
→ Connect the Webcam USB cable directly to the laptop (PC) USB port



Exercise 1

■ Execute Pycharm

→ Insert “Function_Library.py”, “Example Image.jpg” into Python Project

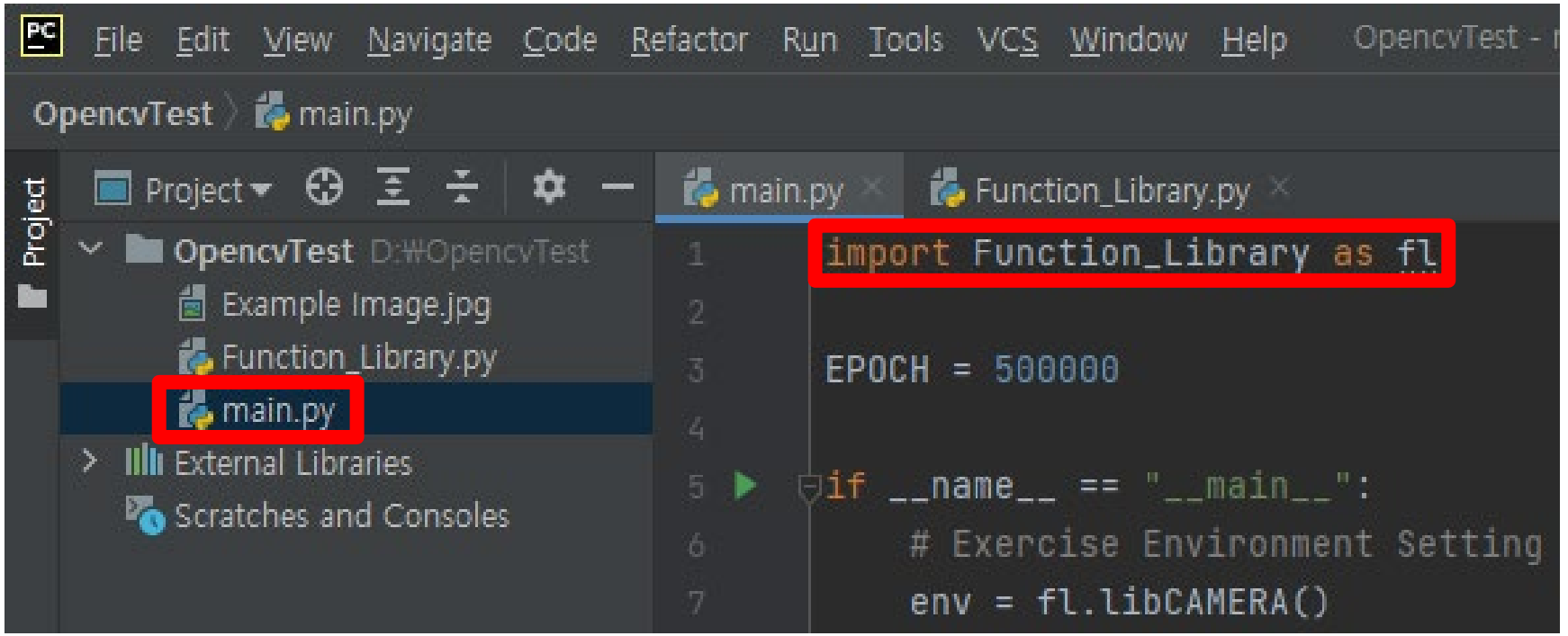


Example Image.jpg

Exercise 1

■ Import Library

→ Create Main Code, Load “Function_Library.py”

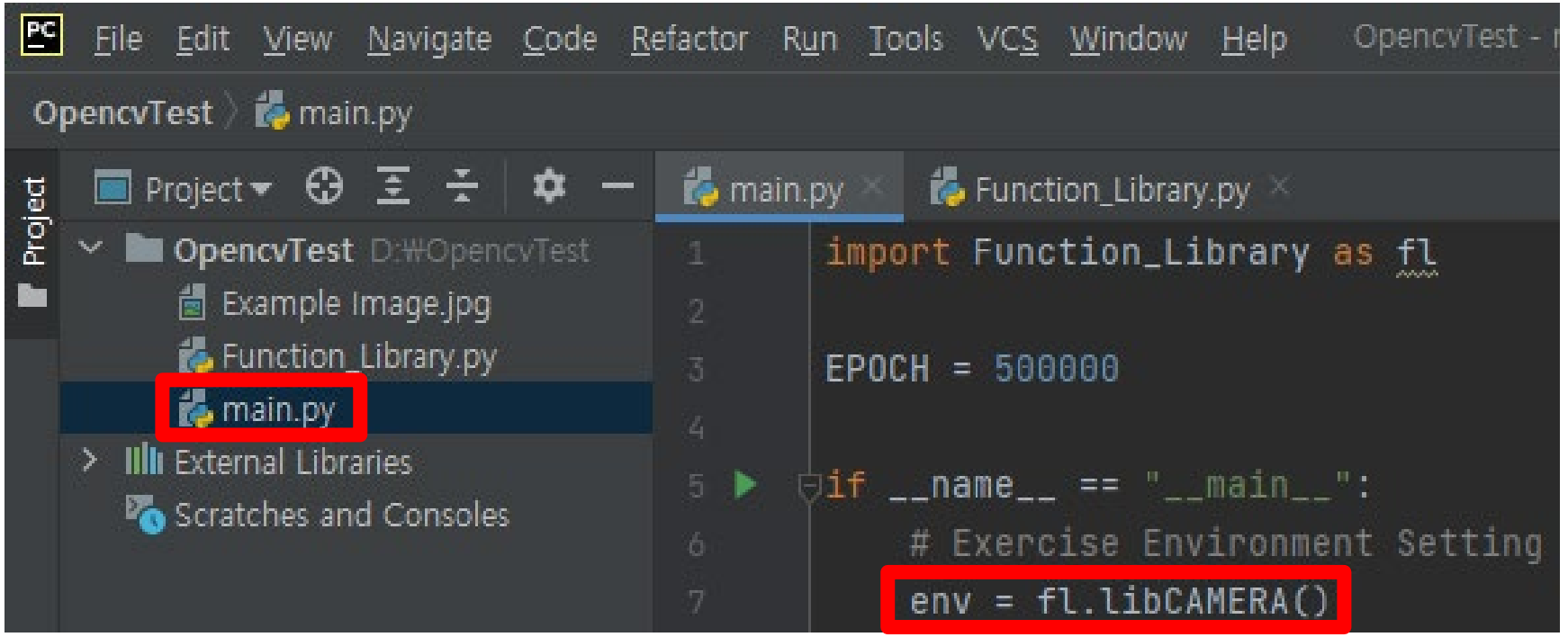


Exercise 1

Declare Environment

→ Load libCAMERA() Class in “Function_Library.py”

→ libCAMERA( ) Class is a collection of all the functions needed for practice



Exercise 1

■ RGB Color Value Extracting

→ Extract Red/Green/Blue values for example samples

```
if __name__ == "__main__":
    # Exercise Environment Setting
    env = fl.libCAMERA()

    """ Exercise 1: RGB Color Value Extracting """
    ################ YOU MUST EDIT ONLY HERE ################
    example = env.file_read("./Example Image.jpg")
    R, G, B = env.extract_rgb(example, print_enable=True)
    quit()
    ########################################################
```

Exercise 1

■ “file_read( )” Description

→ A function that loads the desired image file as a digital value from a specified path

```
def file_read(self, img_path):
```

①

②

```
return np.array(cv2.imread(img_path))
```

① Input: Enter the desired file path

② Output: For color image files, output digital values (three-dimensional arrays)
→ (Size: **Column × Row × Channel**, In this case, Channel corresponds to three types of R/G/B)

Exercise 1

■ “extract_rgb( )” Description

→ A function that separates the values corresponding to the red, green, and blue components from image data composed of digital values

```
def extract_rgb(self, img, print_enable=False):
```

① ②

③

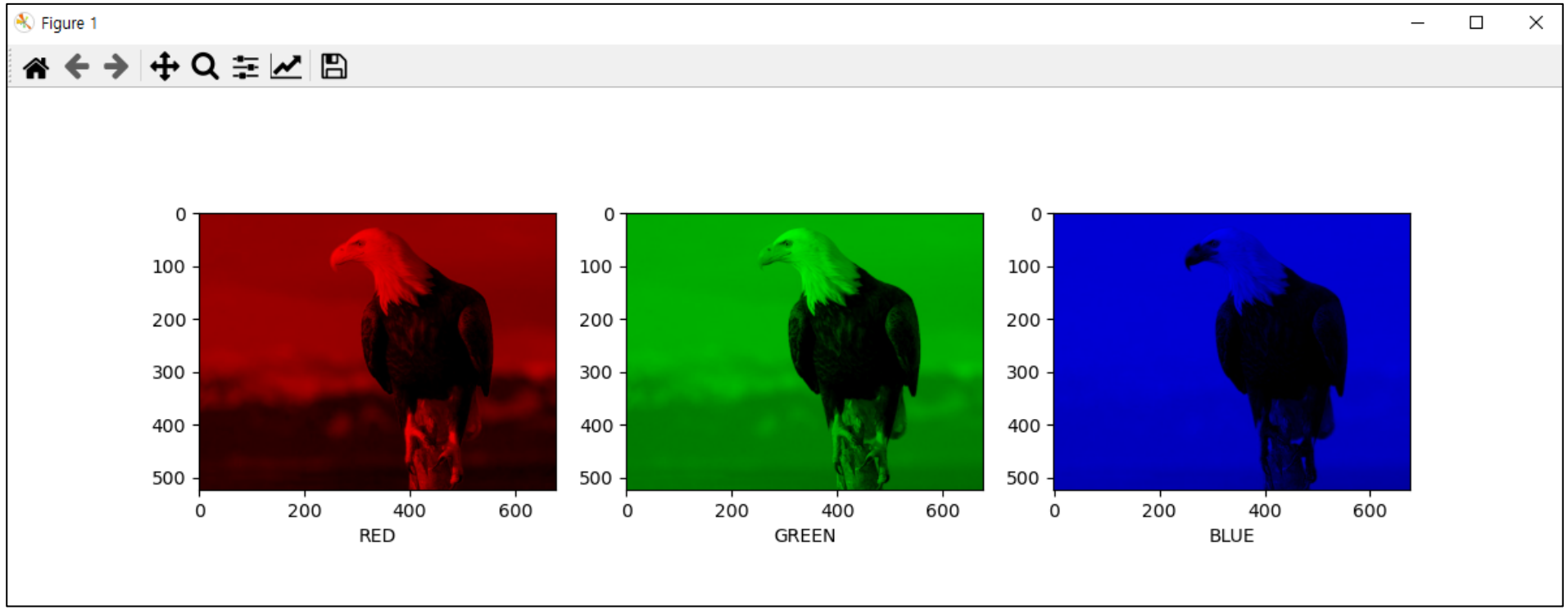
```
return img_red[:, :, RED], img_green[:, :, GREEN], img_blue[:, :, BLUE]
```

① Input1: Input color image data (3-dimensional array) from which you want to separate colors
② Input2: Input whether you want to display images separated by color
③ Output: Output color values separated by channel from the color image (Red/Green/Blue)

Exercise 1

■ Exercise 1 Result

→ When you run the code, the following result window will be displayed.



Exercise 2

■ Webcam Real-time Reading

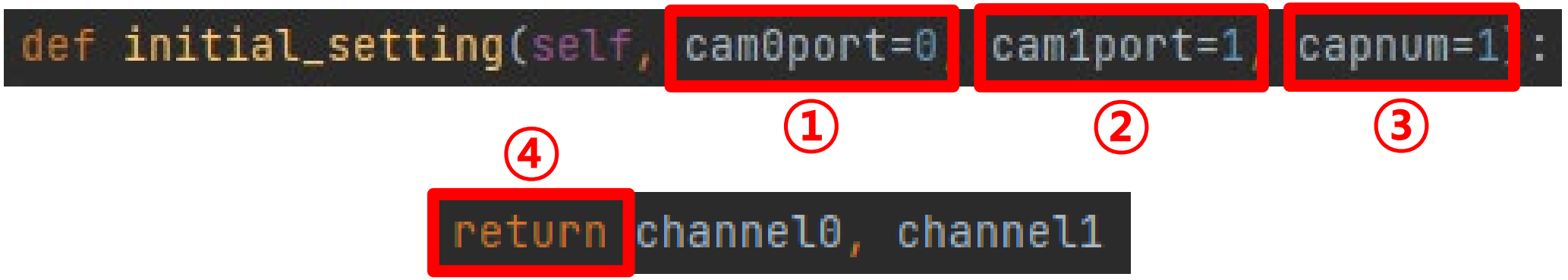
→ Connect Logitech Webcam and Pycharm to output video in real time

```
if __name__ == "__main__":
    # Exercise Environment Setting
    env = fl.libCAMERA()

    """ Exercise 1: RGB Color Value Extracting """
    ############### YOU MUST EDIT ONLY HERE ###############
    # example = env.file_read("./Example Image.jpg")
    # R, G, B = env.extract_rgb(example, print_enable=True)
    # quit()
    ########################################################

    # Camera Initial Setting
    ch0, ch1 = env.initial_setting(capnum=2)

    # Camera Reading..
    for i in range(EPOCH):
        _, frame0, _, frame1 = env.camera_read(ch0, ch1)

        """ Exercise 2: Webcam Real-time Reading """
        ############### YOU MUST EDIT ONLY HERE ###############
        env.image_show(frame0, frame1)
        ########################################################
```

Exercise 2

■ “initial_setting( )” Description

→ Function to link cameras connected to a PC with Pycharm (hardware connection settings)



① Input1: Enter the physical port number for camera 0.

② Input2: Enter the physical port number for camera 1.

③ Input3: Enter the number of cameras connected to the PC (always fixed at 2).

④ Output: Output the channel object information for the cameras whose hardware settings have been completed.

Exercise 2

■ “camera_read( )” Description

→ A function that uses a webcam (camera) to retrieve the frame of the current moment as a digital value.

```
def camera_read(self, cap1 cap2=None):
```

① ②

```
for idx in range(0, self.capnum):
    ret, frame = capset[idx].read()
    result.extend([ret, frame])

return result
```

③

① Input1: Input channel object information for camera 0

② Input2: Input channel object information for camera 1

③ Output: Captures the current frame for each camera channel according to capnum
→ If there are two cameras, four outputs are generated (ret0, frame0, ret1, frame1)

Exercise 2

■ “image_show( )” Description

→ A function that uses a webcam (camera) to retrieve the frame of the current moment as a digital value.

① ②

```
def image_show(self, frame0, frame1=None):
```

```
if frame1 is None:
    cv2.imshow('frame0', frame0)
else:
    cv2.imshow('frame0', frame0)
    cv2.imshow('frame1', frame1)
```

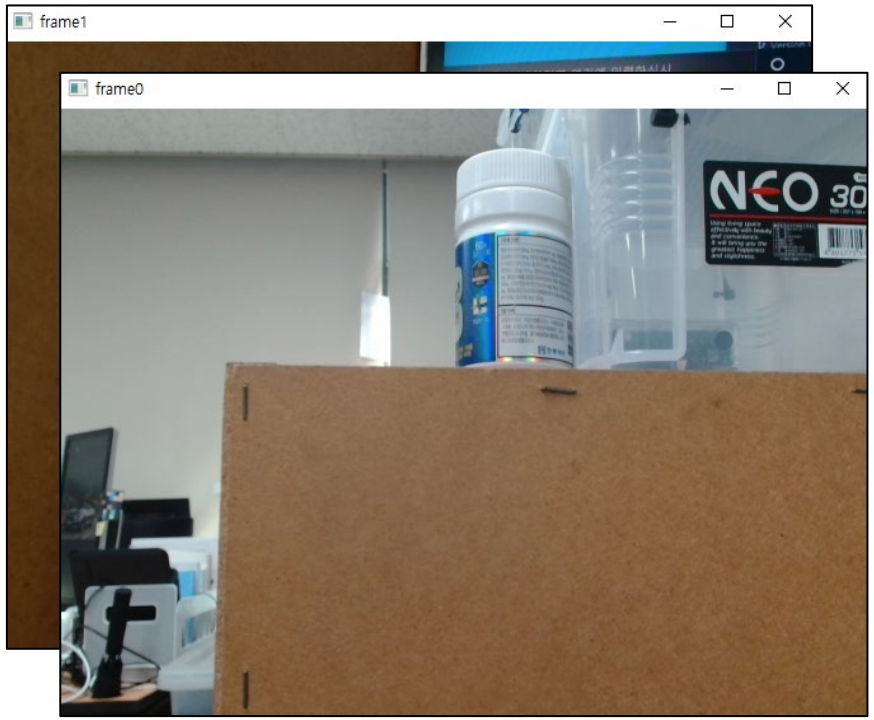
① Input1: Input moment frame data for camera 0

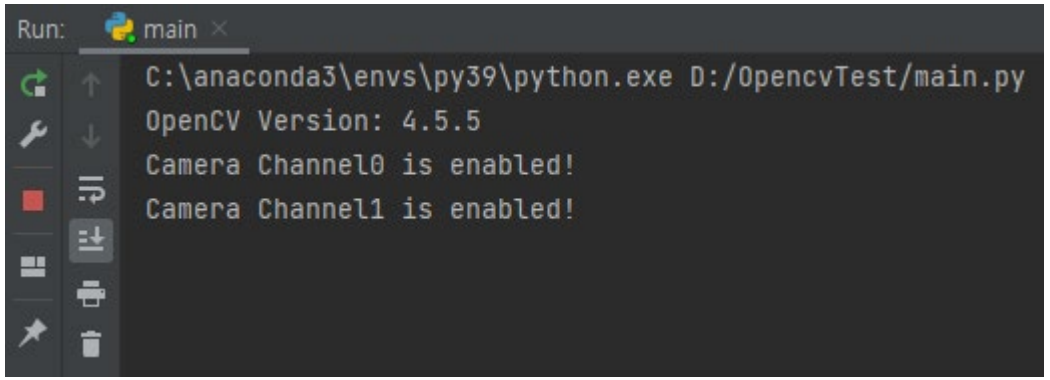
② Input2: Input moment frame data for camera 1→ No separate output (moment frames are output in real time through the figure window)

Exercise 2

Exercise 2 Result

→ When you run the code, you can see the following results
→ During video output, you can exit the program by typing the lowercase letter 'q'.





Exercise 3

■ Object Detection (Traffic Light Circle)

→ Output a sample of the traffic light example to ensure that the webcam recognizes it accurately

```
# Camera Reading..
for i in range(EPOCH):
    _, frame0, _, frame1 = env.camera_read(ch0, ch1)

    """ Exercise 2: Webcam Real-time Reading """
    ############### YOU MUST EDIT ONLY HERE ###############
    # env.image_show(frame0, frame1)
    ########################################################

    """ Exercise 3: Object Detection (Traffic Light Circle) """
    ##################### YOU MUST EDIT ONLY HERE ####################
    color = env.object_detection(frame0, sample=16, print_enable=True)
    ##################################################################
```

Exercise 3

■ “object_detection( )” Description

→ Function that can detect only objects (objects) of a specific color (traffic light color recognition)

```
def object_detection(self, img, sample=0, mode="circle", print_enable=False):
```

① ② ③ ④

① Input1: Input of frame data received from the camera

② Input2: Number of samples used to recognize traffic light colors (Hyper-parameter)→ Can be changed to desired value (preferably use pre-set value)

③ Input3: Enter the Mode value for the Hough transform→ Set to “circle” Mode to detect circular objects of traffic lights

④ Input4: Enter whether to display the image and color values of the output results

Exercise 3

■ “object_detection( )” Description

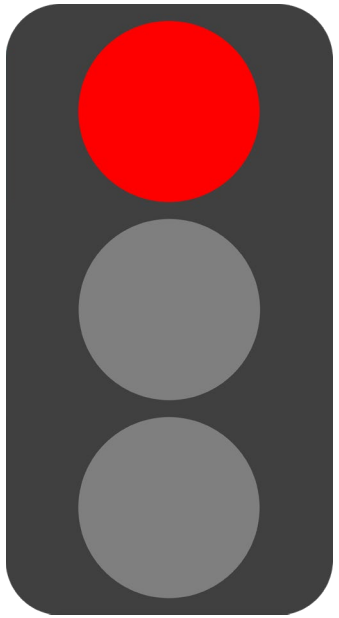
→ Function that can detect only objects (objects) of a specific color (traffic light color recognition)

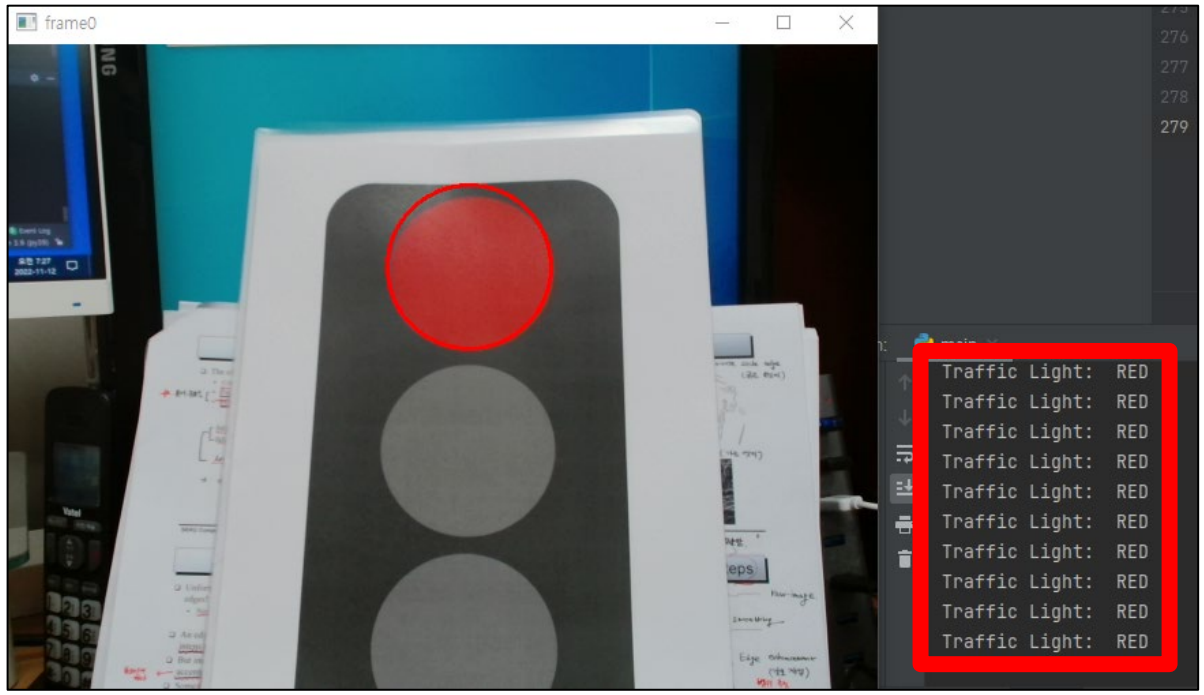
⑤ Output: Outputs the final value (Red/Yellow/Green) determined during the object detection process→ Each color is set as Red: 0 / Yellow: 3 / Green: 1

Exercise 3

■ Exercise 3 Result

→ The following results can be obtained for the traffic light example sample.





Exercise 4

■ Specific Edge Detection (Traffic Line)

→ Print out the lane example sample to check if the webcam recognizes it correctly.

```
# Camera Reading..
for i in range(EPOCH):
    _, frame0, _, frame1 = env.camera_read(ch0, ch1)

    """ Exercise 2: Webcam Real-time Reading """
    ############### YOU MUST EDIT ONLY HERE ###############
    # env.image_show(frame0, frame1)
    ########################################################

    """ Exercise 3: Object Detection (Traffic Light Circle) """
    ##################### YOU MUST EDIT ONLY HERE #####################
    # color = env.object_detection(frame0, sample=16, print_enable=True)
    ###################################################################

    """ Exercise 4: Specific Edge Detection (Traffic Line) """
    ##################### YOU MUST EDIT ONLY HERE #####################
    direction = env.edge_detection(frame0, width=500, height=120,
                                   gap=40, threshold=150, print_enable=True)
    ###################################################################
```

Exercise 4

■ “edge_detection( )” Description

→ Function that can detect specific edge lines (lane direction recognition)

```
def edge_detection(self, img, width=0, height=0, gap=0, threshold=0, print_enable=False):
```

① ② ③ ④ ⑤ ⑥

① Input1: Input the frame data received from the camera

② Input2: Input the maximum horizontal length of the region of interest [ROI]

③ Input3: Input the minimum vertical length of the region of interest [ROI]

④ Input4: Input the distance difference from the comparison target in pixel analysis

⑤ Input5: Input the length condition for distinguishing a specific edge line in pixel analysis

⑥ Input6: Input whether to display the image and direction value for the output result

Exercise 4

■ “edge_detection( )” Description

→ A function that can detect a specific Edge Line (lane direction recognition)

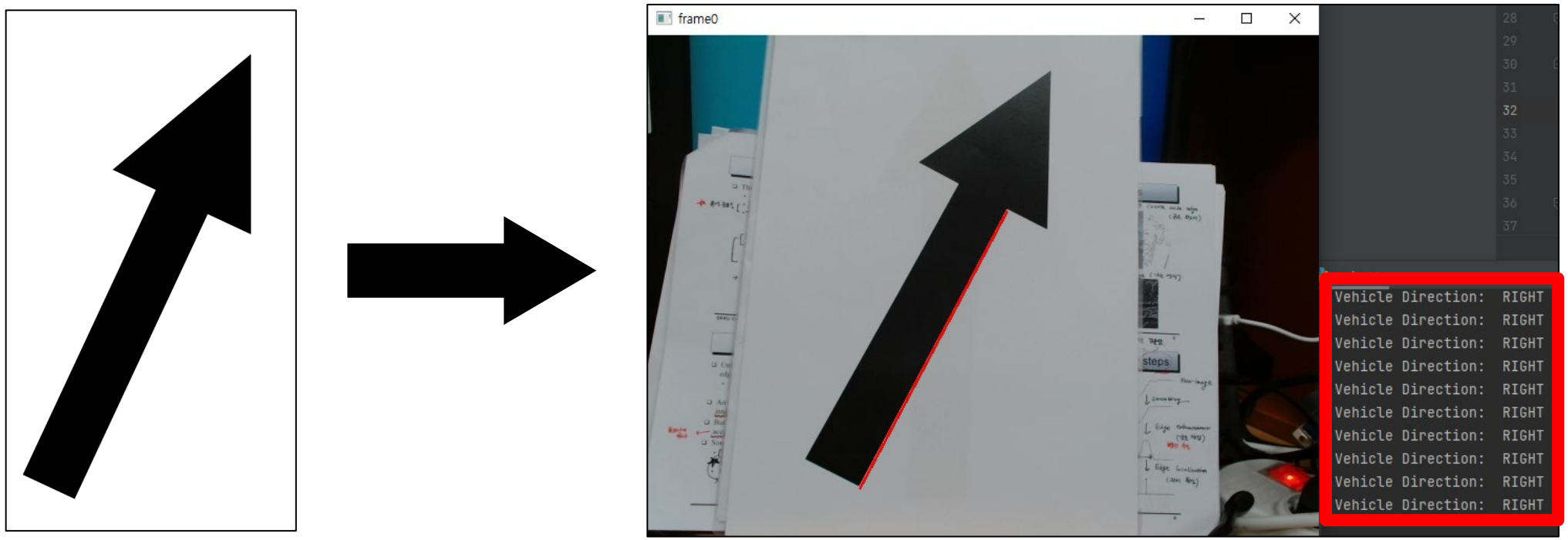
```
                if np.abs(grad) < FORWARD_THRESHOLD:
                    prediction = FORWARD
                elif grad > 0:
                    prediction = RIGHT
                elif grad < 0:
                    prediction = LEFT

                # real_lines.append([xa, ya, xb, yb])
                cv2.line(replica, (xa, ya), (xb, yb), color=[0, 0, 255], thickness=2)
            new_lines.append([xa, ya, xb, yb])
    if print_enable:
        if prediction is not None:
            print("Vehicle Direction: ", DIRECTION[prediction])
        self.image_show(replica)

⑦  return prediction
```

⑦ Output: Predicts the direction of the lane based on the slope of a straight line for a specific edge line→ Forward: FORWARD (0) / Right: RIGHT (2) / Left: LEFT (1)

Exercise 4

■ Exercise 4 Result (1)

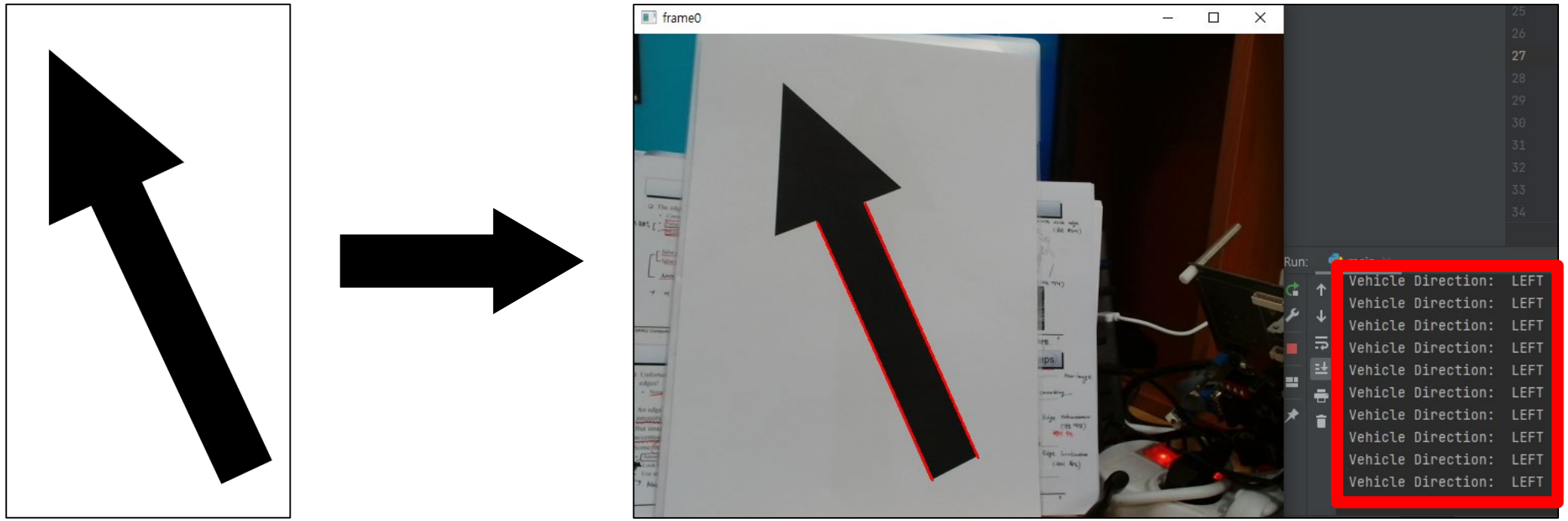
→ The following results can be obtained for the lane example sample



Exercise 4

■ Exercise 4 Result (2)

→ The following results can be obtained for the lane example sample



Thank You!

Automation Lab