Assignment 2: Documentation

Jonah Bertolino, Hunter Burnham, Joseph Kirby

Questions:

2a.

a. An OpenCV file is stored in memory by pixels, and each pixel contains three 24-bit numbers, which are the colors. The first number represents how much blue is in the image, the second number represents how much green is in the image, and the last number represents how much red is in the image. Since they are 24-bits, the largest value they can have is 255. I find this interesting because I always think of images as RGB values, instead of BGR values.
b. The “image.shape” command returns the dimensions of the array. In this case, the variable “image” is having its dimensions returned.
c. The detector parameters are set with the PTR<DetectorParameters> argument to the detect marker function. To set the parameters, you create an instance of this class that holds the values for many different parameters used in the detection function. The member of this value in this class that gives the minimum perimeter for marker contour to be detected is the minMarkerPerimeterRate. This value is a rate and not an absolute value because it is relative to the size of the input image. If a marker is too small with respect to this parameter, the detect function will not count it as a marker.

2b.

a. There are 7 morphological transformations that can be performed on an image. These include erosion, dilation, opening, closing, morphological gradient, top hat, and black hat.
b. Erosion cuts away at the edges of a shape or white space on an image, leaving a skinnier version of the original shape. This is useful for eliminating noise that is outside the shape or white space (when combined with dilation after).
c. Dilation adds to the edges of a shape or white space in an image, leaving a thicker version of the original shape. This is useful for eliminating noise that is inside the shape or white space (when combined with erosion after).

1a Code:

Python

```python
from smbus2 import SMBus
from time import sleep

# I2C address of the Arduino, set in Arduino sketch
ARD_ADDR = 8

# Initialize SMBus library with I2C bus 1
i2c = SMBus(1)
```

```
# Do in a loop
while(True):

    # Get user input for offset
    offset = int(input("Enter an offset (7 to quit): "))

    # Provide an exit key
    if(offset == 7):
        break

    # Get user input for command
    string = input("Enter a string of 32 characters or less:")

    # Write a byte to the i2c bus
    command = [ord(character) for character in string]

    try:
        #ask the arduino to take on encoder reading
        i2c.write_i2c_block_data(ARD_ADDR,offset,command)
    except IOError:
        print("Could not write data to the to the Arduino.")
```

Arduino

```
#include <Wire.h>
#define MY_ADDR 8
// Global variables to be used for I2C communication
volatile uint8_t offset = 0;

volatile uint8_t instruction[32] = {0};
volatile uint8_t msgLength = 0;
volatile uint8_t reply = 0;

void setup() {
  Serial.begin(115200);
  // We want to control the built-in LED (pin 13)
  pinMode(LED_BUILTIN, OUTPUT);
  // Initialize I2C
  Wire.begin(MY_ADDR);
  // Set callbacks for I2C interrupts
  Wire.onReceive(receive);
}

void loop() {
  // If there is data on the buffer, read it
```

```
  if (msgLength > 0) {
    if (offset==1) {
    digitalWrite(LED_BUILTIN,instruction[0]);
    }

  printReceived();
  msgLength = 0;
  }
}

// printReceived helps us see what data we are getting from the leader
void printReceived() {
  // Print on serial console
  Serial.print("Offset received: ");
  Serial.println(offset);
  Serial.print("Message Length: ");
  Serial.println(msgLength);
  Serial.print("Message: ");
  for(int i = 0; i < msgLength; i++){
    Serial.print(char(instruction[i]));
  }
  Serial.println();
  Serial.print("Instruction received: ");
  for (int i=0;i<msgLength;i++) {
      Serial.print(String(instruction[i])+"\t");
    }
    Serial.println("");
}

// function called when an I2C interrupt event happens
void receive() {
  // Set the offset, this will always be the first byte.
  offset = Wire.read();
  // If there is information after the offset, it is telling us more about the
command.
  while (Wire.available()) {
    instruction[msgLength] = Wire.read();
    msgLength++;
  }
}
```

1b Code:

Python

```
rom smbus2 import SMBus
from time import sleep
import board
import adafruit_character_lcd.character_lcd_rgb_i2c as character_lcd

# Modify this if you have a different sized Character LCD
lcd_columns = 16
lcd_rows = 2

# Initialise I2C bus.
i2c = board.I2C()

# Initialise the LCD class
lcd = character_lcd.Character_LCD_RGB_I2C(i2c, lcd_columns, lcd_rows)

# I2C address of the Arduino, set in Arduino sketch
ARD_ADDR = 8

# Initialize SMBus library with I2C bus 1
i2c = SMBus(1)

#get user input for integer, ensure between 0 and 100
while(True):
    integer = int(input("Enter an integer between 0 and 100: "))
    if integer < 0 or integer > 100:
        print("integer was not between 0 and 100")
    else:
        break

#send to arduino
command = integer

try:
    #ask the arduino to take on encoder reading
    i2c.write_byte_data(ARD_ADDR,0,command)
except IOError:
    print("Could not write data to the to the Arduino.")

sleep(0.1)

#requet byte from arduino
reply = i2c.read_byte_data(ARD_ADDR, 0)
print("Recieved from Arduino: " +str(reply))

#Display on LCD
```

```
lcd.clear()
lcd.message = "Recieved: " +str(reply)
```

Arduino

```
#include <Wire.h>
#define MY_ADDR 8
// Global variables to be used for I2C communication
volatile uint8_t offset = 0;

volatile uint8_t instruction[32] = {0};
volatile uint8_t msgLength = 0;
volatile uint8_t reply = 0;

void setup() {
  Serial.begin(115200);
  // We want to control the built-in LED (pin 13)
  pinMode(LED_BUILTIN, OUTPUT);
  // Initialize I2C
  Wire.begin(MY_ADDR);
  // Set callbacks for I2C interrupts
  Wire.onReceive(receive);
  Wire.onRequest(request);
}

void loop() {
  // If there is data on the buffer, read it
  if (msgLength > 0) {
    if (offset==1) {
    digitalWrite(LED_BUILTIN,instruction[0]);
    }
  printReceived();
  msgLength = 0;
  }
}

// printReceived helps us see what data we are getting from the leader
void printReceived() {
  Serial.print("Recieved: ");
  Serial.println(instruction[0]);
  Serial.print("Reply: ");
  reply = int(instruction[0]) + 100;
  Serial.println(reply);
}
// function called when an I2C interrupt event happens
void receive() {
```

```
  // Set the offset, this will always be the first byte.
  offset = Wire.read();
  // If there is information after the offset, it is telling us more about the
command.
  while (Wire.available()) {
    instruction[msgLength] = Wire.read();
    msgLength++;
  }
}

void request() {
  // According to the Wire source code, we must call write() within the
requesting ISR
  // and nowhere else. Otherwise, the timing does not work out. See line 238:
  // https://github.com/arduino/ArduinoCore-
avr/blob/master/libraries/Wire/src/Wire.cpp
  Wire.write(reply);
  Serial.println("Request Recieved");
  reply = 0;
}
```

2a Code:

```
import cv2
from cv2 import aruco
import numpy as np
from time import sleep
import time
import board
import adafruit_character_lcd.character_lcd_rgb_i2c as character_lcd

# Initialise I2C bus.
i2c = board.I2C()

# Initialize LCD
lcd_columns = 16
lcd_rows = 2
lcd = character_lcd.Character_LCD_RGB_I2C(i2c, lcd_columns, lcd_rows)

#create instance of aruco library
aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_6X6_50)

# initialize the camera. If channel 0 doesn't work, try channel 1
camera = cv2.VideoCapture(0)
```

```
#set dimensions
camera.set(cv2.CAP_PROP_FRAME_WIDTH,640)
camera.set(cv2.CAP_PROP_FRAME_HEIGHT,480)

#initialize variables
preMessage = ""

# Let the camera warmup
sleep(0.1)

# Get an image from the camera stream
while(True):
    ret, frame = camera.read()
    grey = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY) # Make the image greyscale for
ArUco detection
    cv2.imshow("overlay",grey)
    #take pictures until keypress
    k = cv2.waitKey(1) & 0xFF
    if k == ord("q"):
        break

    #detect markers
    corners,ids,rejected = aruco.detectMarkers(grey,aruco_dict)

    #create message to display
    if len(corners) > 0:
        numIds = len(ids)
        message = "Aruco ID's found:\n"
        for i in range(numIds):
            message += str(ids[i][0])
            message += " "
    else:
        message = "No aruco markers detected"


    # Set LCD color to red and display message
    lcd.color = [100, 0, 0]

    #clear lcd if message is different
    if message != preMessage:
        lcd.clear()
    preMessage = message

    #display message
    lcd.message = message
```

```
cv2.destroyAllWindows()
lcd.clear()
```

2b Code:

```
from time import sleep
import numpy as np
import cv2

#create file name for image
fileName = input("File Name: ")

#initialize camera
camera = cv2.VideoCapture(0)

#capture image
ret, image = camera.read()

# Convert the captured image to HSV
imageHSV = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)

#set color bounds (only captures green colors)
upperred = np.array([83,255,150])
lowerred = np.array([17,127,56])

#create mask using image and bounds, this is a binary array with 1's at desired
pixels
mask = cv2.inRange(imageHSV,lowerred,upperred)

#bitwise and the image and the mask to extract desired pixels (green ones) into
result image
greenImg = cv2.bitwise_and(image, image, mask=mask)

#create kernel matrix for morphology
kernel = np.ones((5,5),np.uint8)

#use morphology functions to reduce noise inside and outside green circle
morphOpenImg = cv2.morphologyEx(greenImg, cv2.MORPH_OPEN, kernel)
finalImg = cv2.morphologyEx(morphOpenImg, cv2.MORPH_CLOSE, kernel)

#display final image
cv2.imshow("Final Image",finalImg)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite(fileName, finalImg)
```

```
#handle errors
if not ret:
    print("Could not capture image from camera!")
    quit()

else:
    print("Saving image "+fileName)
    try:
        cv2.imwrite(fileName, finalImg)
    except:
        print("Could not save "+fileName)
        pass
```