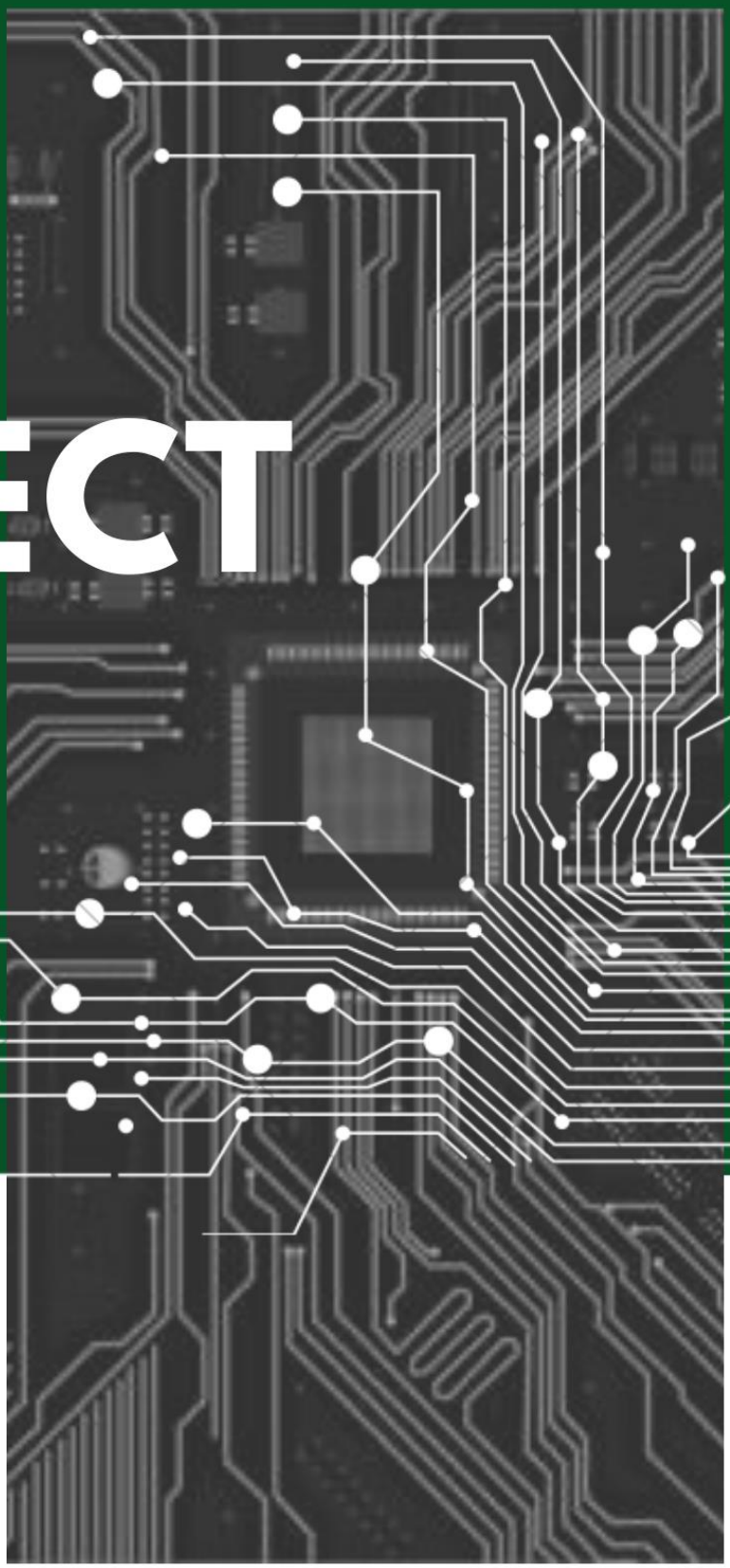
FINAL PROJECT REPORT FALL 2021



SCHOOL OF ENGINEERING

MCE410-31
JAWAD AL KHOURY

15/2/2021

Iyad Baz 201901551
Khaled Al Madani 201902362
Karim Kaouk 201900327
Reda Atoui 201905047

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Project Description

We developed a sorting system which detects certain objects, their shape, and their color then sorts them according to the user's preference. In total there are 6 objects that are used in the sorting process, and 3 different shapes being detected by our system: 2 circles, 2 triangles, and 2 squares. Furthermore, 2 colors are being detected: green and yellow, where each shape has 1 of each color.

When the system starts, the user receives a window prompt asking whether to sort the given object according to shape or color. After the user chooses, the conveyor belt starts moving. When the object is directly under the camera, the conveyor belt stops and the camera detects either its shape or color based on the previous user prompt.

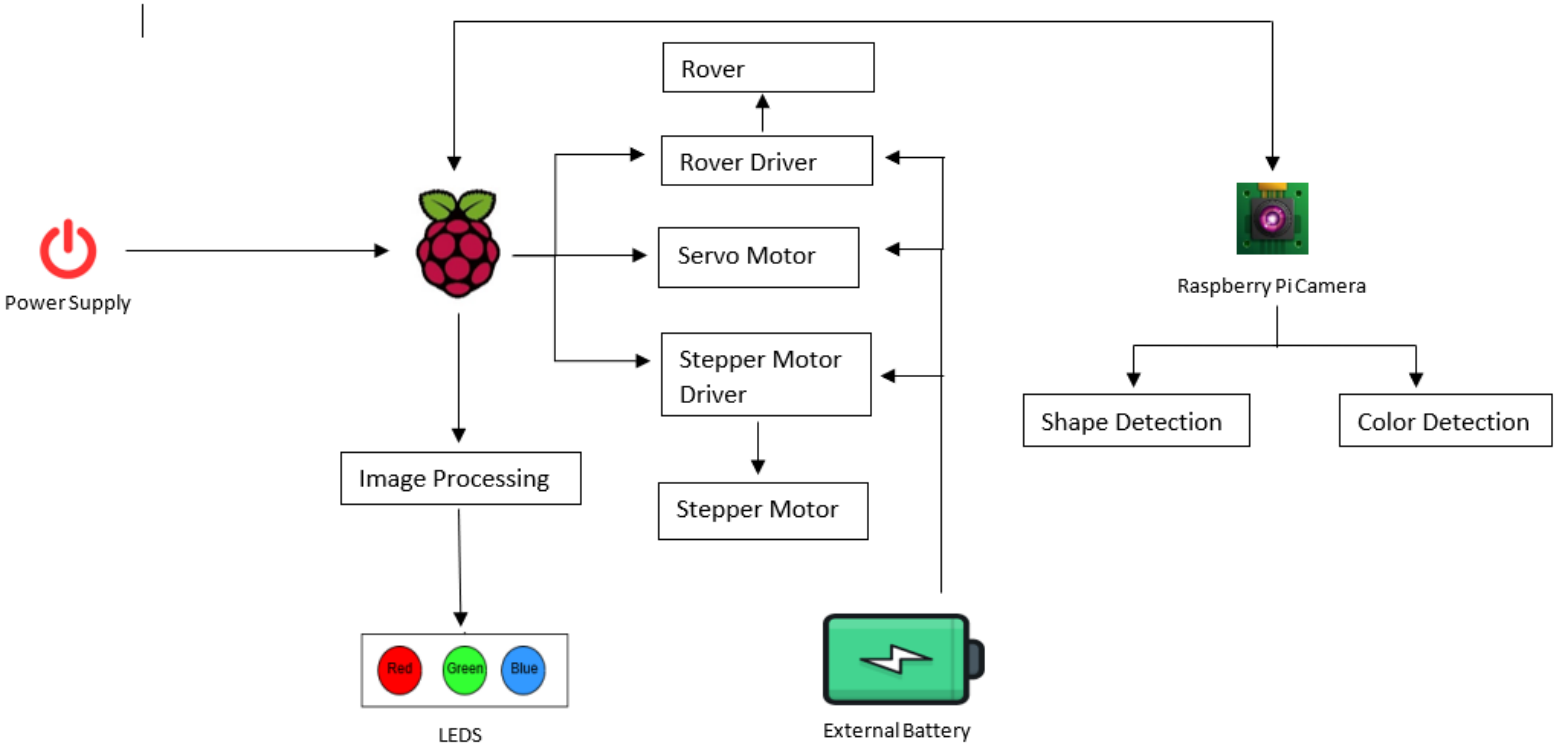
There are 2 possible scenarios:

1. If we are sorting based on color the conveyor belt resumes moving and drops the object into the rotating arm basket. The stepper motor rotates the arm basket to its specified location based on its color, and then the servo motor rotates the basket 180° dropping the object into the appropriate paper box. The servo then rotates the basket back to its normal position, and the stepper motors turns back the rotating arm basket to its initial position next to the conveyor belt.
2. If we are sorting based on shape and the object is a square or triangle, the same process as the first scenario happens (however position of the stepper is dependent on the shape). If the object is neither, this case is represented by a circle and the conveyor belt moves backward throwing the object into a paper box (circle labeled) and a red LED lights up.

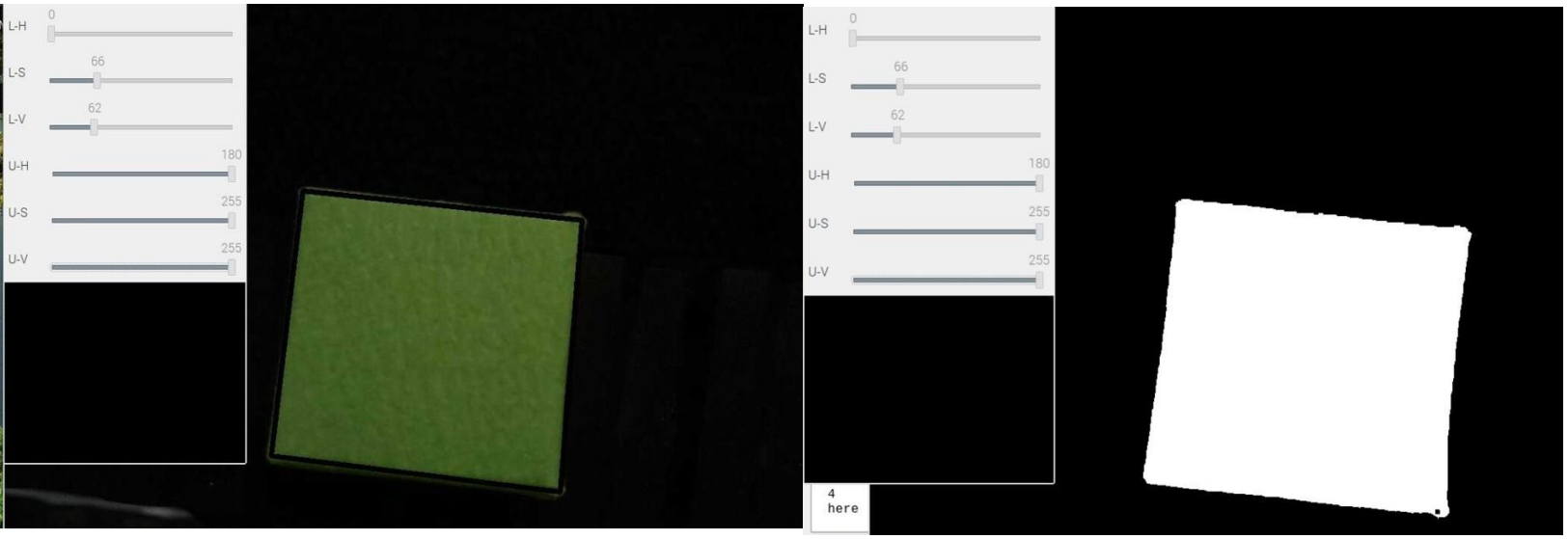
In both scenarios the color of the object is being detected and a LED of the same color lights up when detected.

The boxes are labeled on the inside to indicate the shape/color of the object they are supposed to hold.

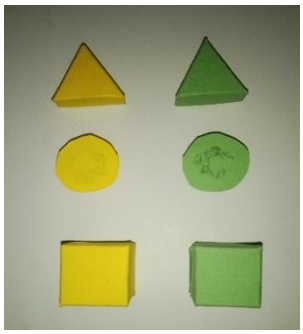
This process is repeated for all the objects.



System Architecture



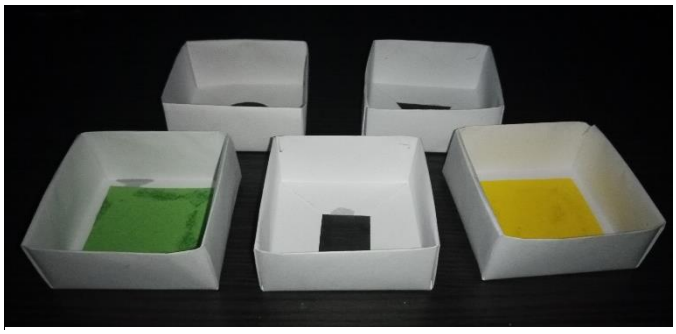
Contour Detection – Masked and Unmasked Frames



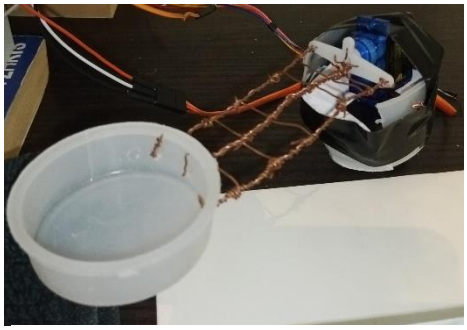
Objects - Top View

Objects - Side View

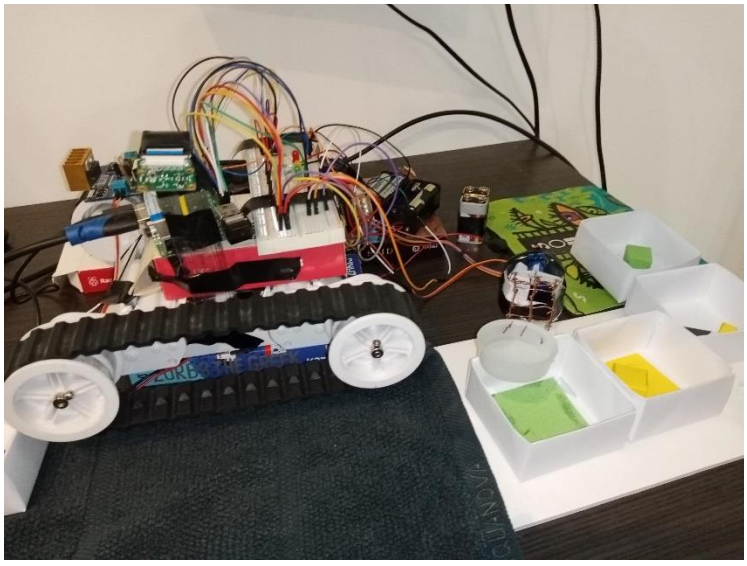
Labeled Paper Boxes - Top View

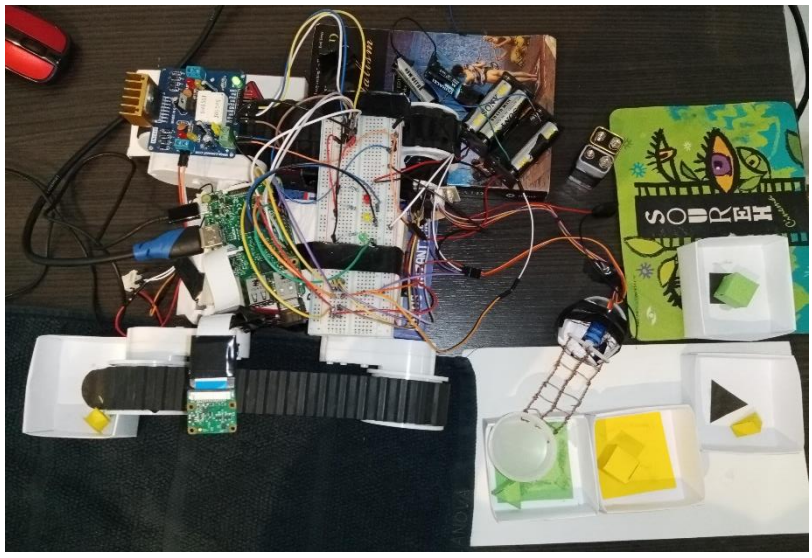


Labeled Paper Boxes – Front View



Rotating Basket Arm

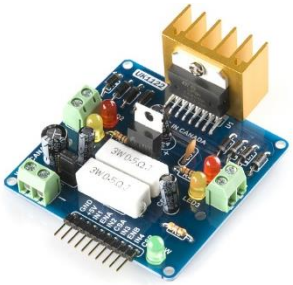




Project (2 Views)

Materials & Components

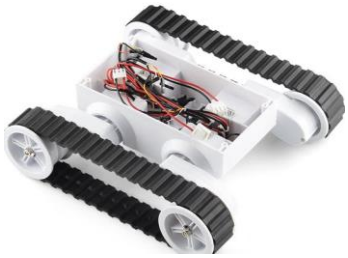
- Female-to-Male Wires
- Male-to-Male Wires
- Breadboard
- SG90 Servo Motor
- 28BYJ-48 Stepper Motor
- ULN2003 Stepper Motor Driver
- Copper Wires
- Raspberry Pi 3B
- Raspberry Pi V2 Camera
- Rover 5 – 4 motors 4 encoders
- Small Plastic Containers
- Electric Tape
- LED
- Cardboard
- UK1122 Rover Driver
- Altico Glue
- Resistors
- 32 GB SD Card
- 9V Batteries
- Paper / Cardboard
- Battery Chassis



UK1122 Rover Driver

SG90 Servo Motor

Stepper Motor & Driver



Rover 5 - Robot Chassis

Raspberry Pi 3B

Female-to-Male Wires

Raspberry Pi V2 Camera

Male-to-Male Wires

Datasheets

SG90 Servo Motor	UK1122 Rover Driver
ULN2003 Stepper Motor Driver	28BYJ-48 Stepper Motor

Code

Documentation

Time

- time.sleep ([duration]): Allows us to stop the code from running for the specified duration.

GPIO as IO Library

- IO.output ([pin], [value]): Assign each pin a digital value by providing the number of the pin and the value wanted.
- IO.PWM ([pin], [frequency]): Allows us to generate an analog output to the pin of the pi with the wanted frequency.
- IO.setwarnings([boolean]]: Disables/Enables unwanted warnings
- pwm.start([duty cycle]): Allows us to start the analog output with the wanted duty cycle.
- IO.setup([pin], [IO.IN, IO.OUT]): Setting the pin mode for either use as an input or an output.
- IO.setmode([system]): Allows us to specify which number-system is being used In our code.
- IO.cleanup(): Cleans up all the ports and resets them to inputs.

cv2

- cv2.videoCapture(index): Opens camera for video capturing.
- cv2.cvtColor(src, code [, dst [, dstCn]]: Converts an image from one color space to another where the src is the input image, code is the color space conversion, dst is the output image and dstCn is the number of channels in the destination image.
- cv2.inRange(src, lowerb, upperb): Checks if array elements lie between the elements of two other arrays where src is the first input array and lowerb and upperb are the boundaries.
- cv2.imshow('string', img): Opens a tab named string that shows the image img
- cv2.createTrackbar (trackbarname, winname , value , count, TrackbarCallback onChange=0 ,*userdata=0) : creates a trackbar (a slider or range control) with the specified name and range, assigns a variable value to be a position synchronized with the trackbar and specifies the callback function onChange to be called on the trackbar position change. The created trackbar is displayed in the specified window winname.
- cv2.namedWindow (window name, int flags=WINDOW_AUTOSIZE): creates a window that can be used as a placeholder for images and trackbars.
- cv2.destroyWindow(window name): destroys the window with the given name.
- cv2.getTrackbarPos (trackbarname, window name): returns the current position of the specified trackbar.
- cv2.erode(src, kernel [, dst [, anchor [, iterations [, borderType[, borderValue]]]]]): erodes the source image using the specified structuring element.
- cv2.contourArea([input]): Calculates a contour area.
- cv2.approxPolyDP([curve], epsilon, Boolean): Approximates a polygonal curve(s) with the specified precision (epsilon).
- cv2.drawContours(image, contours, contoursIdx, color, thickness, lineType, hierarchy, maxLevel, offset): Draws contours outlines or filled contours.
- cv2.findContours(image, mode, method [, contours[, hierarchy[, offset]]]): Finds contours in a binary image (using algorithm 234).

Numpy Library

- np.array([array]): Allows us to store items of same type and size in a multidimensional container.
- np.sum([array]): Sums elements of array.

Full Code

```
from numpy.core.fromnumeric import shape
import cv2
import time
import RPi.GPIO as IO
import numpy as np
from tkinter import *
# start capturing video using using the camera
# setting the resolution and frame rate of the
video
video = cv2.VideoCapture(0)
video.set(3, 1280)
video.set(4, 1024)
video.set(5, 30)

# code for stepper motor control
P_A1 = 7  # adapt to your wiring
P_A2 = 11  # ditto
P_B1 = 16  # ditto
P_B2 = 18  # ditto
delay = 0.005  # time to settle
servopwm = 13


# Window method for user interface
def openWindow():
    window = Tk()

    # set background colors,fonts, size of elements
of gui
    window.configure(bg="black")
    window.geometry("500x400+400+200")
    shpbtn = Button(window, text='Shape sort')
    clrbtn = Button(window, text='Color sort')
    shpbtn.place(x=325, y=200)
    clrbtn.place(x=325, y=500)
    label = Label(window, text="Choose the mode of
sorting: ")
    label.config(bg='#000000')
    label.config(fg="#ffffff")
    shpbtn.config(command=lambda:
[window.destroy(), Shape()])  # performs a shape
sort when clicked and closes pop window
    clrbtn.config(command=lambda:
[window.destroy(), Color()])  # performs a color
sort when clicked and closes pop window
    shpbtn.config(font=('Arial', 25, 'bold'))
    shpbtn.config(bg='#368f46')
    shpbtn.config(fg='#000000')
    label.config(font=('Arial', 15))
    clrbtn.config(font=('Arial', 25, 'bold'))
    clrbtn.config(bg='#368f46')

    clrbtn.config(fg='#000000')

    # pack the elements on the window with a
certain padding
    label.pack(pady=50, padx=25)
    shpbtn.pack(pady=5)
    clrbtn.pack(pady=5)

    window.mainloop()


# defining the sequence for the stepper motor
forward rotation
def forwardStep():
    setStepper(1, 0, 0, 0)
    setStepper(1, 1, 0, 0)
    setStepper(0, 1, 0, 0)
    setStepper(0, 1, 1, 0)
    setStepper(0, 0, 1, 0)
    setStepper(0, 0, 1, 1)
    setStepper(0, 0, 0, 1)
    setStepper(1, 0, 0, 1)


# defining the sequence for the stepper motor
backward rotation
def backwardStep():
    setStepper(1, 0, 0, 1)
    setStepper(0, 0, 0, 1)
    setStepper(0, 0, 1, 1)
    setStepper(0, 0, 1, 0)
    setStepper(0, 1, 1, 0)
    setStepper(0, 1, 0, 0)
    setStepper(1, 1, 0, 0)
    setStepper(1, 0, 0, 0)


# setting the pins used to connect the stepper
motor
def setStepper(in1, in2, in3, in4):
    IO.setmode(IO.BOARD)
    IO.output(P_A1, in1)
    IO.output(P_A2, in2)
    IO.output(P_B1, in3)
    IO.output(P_B2, in4)
    time.sleep(delay) #delay until the object is
dropped by the servo
```

```
# function to stimulate direction and degree of rotation of stepper motor
# 512 step for 360 degrees
def stepper(direction, steps):
    IO.setmode(IO.BOARD)
    IO.setwarnings(False)
    IO.setup(7, IO.OUT)  # stepper PA1
    IO.setup(11, IO.OUT)  # stepper PA2
    IO.setup(16, IO.OUT)  # stepper PB1
    IO.setup(18, IO.OUT)  # stepper PB2

    # conditional for loop that runs the code for the number of steps given
    # in the function arguments
    # the direction given as an argument is checked to specify direction mode of the motor
    for i in range(steps):
        if direction == "forward":
            forwardStep()
        else:
            backwardStep()


# function to rotate the head of the servo to drop object in desired location
# first we rotate the motor one half rotation(180 degrees) then return it back(0)
def drop():
    servo_motor(180)
    IO.setmode(IO.BOARD)
    IO.setup(servopwm, IO.OUT)
    servo_motor(0)


# code for servo motor functionality
# setting up all output and pwm pins needed
def servo_motor(angle):
    # Then, setup the GPIO mode as BOARD so that you can reference the PINs and not the BCM pins.
    IO.setup(servopwm, IO.OUT)
    # Next, create a variable for the servo. I called mine “PWM.” Send a 50 Hz PWM signal on that GPIO pin
    # using the GPIO.PWM() function.
    # #Start the signal at 0.
    pwm = IO.PWM(servopwm, 50)
    pwm.start(0)

    # calculate duty cycle using the angle given
    duty = angle / 18 + 3
    IO.output(servopwm, True)
    pwm.ChangeDutyCycle(duty)
    time.sleep(1) #small delay for the object ot be dropped smoothly
    IO.output(servopwm, False)
    pwm.ChangeDutyCycle(duty)
    # Lastly, clean up the code by stopping the PWM signal and running the cleanup function on the GPIO pins.
    pwm.stop()
    IO.cleanup()


# conveyor function to run the conveyor
def conveyor(duty_cycle1, front, back, duration=0):

    #setting the pins used for running the conveyor via the motor driver
    inA = 36
    inB = 37
    # pin 36 connected to IN1 of the motor drive
    IO.output(inA, back)
    # pin 37 connected to IN2 of the motor drive
    IO.output(inB, front)
    # pin 33 connected to EN1 of the motor drive
    pwm1 = IO.PWM(33, 100)
    pwm1.start(duty_cycle1)

    time.sleep(duration)
    IO.output(inA, 1)
    IO.output(inB, 1)


# function for checking the color of the object
def checkColor():
    color = "none"

    for i in range(20):

        # take a frame of the running video
        # convert colors from BGT to HSV in order to make use of openCV methods
        _, frame = video.read()
        hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)


        # setting code for green color detection and ranges of light and dark green
        low_green = np.array([40, 150, 20])
        high_green = np.array([70, 255, 255])
        green_mask = cv2.inRange(hsv_frame, low_green, high_green)  #mask all pixels that are not green range
```

```
        green_sum = np.sum(green_mask) #sum of
pixels detected to use as a condition below
        # green = cv2.bitwise_and(frame, frame,
mask=green_mask)



        # setting code for yellow color detection
and ranges of light and dark yellow
        low = np.array([20, 100, 100])
        high = np.array([30, 255, 255])
        yellow_mask = cv2.inRange(hsv_frame, low,
high)
        sum_yellow = np.sum(yellow_mask)
        # yellow = cv2.bitwise_and(frame, frame,
mask=yellow_mask)

        # for verification of the color:
        # first make sure either a yellow or green
color is detected
        if green_sum > 1500 or sum_yellow > 1500:
            if green_sum >
sum_yellow:              #the larger pixel sum
indicates the color
                print("green")
                color = "green"
                return color
            else:
                print("yellow")
                color = "yellow"
                return color

    #     cv2.imshow("Green", green)
    #     cv2.imshow("Yellow", yellow)

    return color


#def nothing(x):
    # any operation
#    pass

# check shape function
def checkShape():
    shape = "none"

    #trackbars to tune the readings

    #     cv2.namedWindow("Trackbars")
    #     cv2.createTrackbar("L-H", "Trackbars", 0,
180, nothing)
    #     cv2.createTrackbar("L-S", "Trackbars",
66, 255, nothing)
    #     cv2.createTrackbar("L-V", "Trackbars",
134, 255, nothing)
    #     cv2.createTrackbar("U-H", "Trackbars",
180, 180, nothing)
    #     cv2.createTrackbar("U-S", "Trackbars",
255, 255, nothing)
    #     cv2.createTrackbar("U-V", "Trackbars",
243, 255, nothing)

    # take a frame of the running video
    # set a font for the contours to be drawn
around object
    #font = cv2.FONT_HERSHEY_COMPLEX
    _, frame = video.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)  #
convert colors from BGT to HSV in order to make use
of openCV methods

    #     l_h = cv2.getTrackbarPos("L-H",
"Trackbars")
    #     l_s = cv2.getTrackbarPos("L-S",
"Trackbars")
    #     l_v = cv2.getTrackbarPos("L-V",
"Trackbars")
    #     u_h = cv2.getTrackbarPos("U-H",
"Trackbars")
    #     u_s = cv2.getTrackbarPos("U-S",
"Trackbars")
    #     u_v = cv2.getTrackbarPos("U-V",
"Trackbars")


    # set the ranges tuned from the trackbars as
default
    lower_red = np.array([0, 66, 62])
    upper_red = np.array([180, 255, 255])
    mask = cv2.inRange(hsv, lower_red, upper_red)
#mask all not in light range

    # we define a 5x5 kernel of 1s to multiply the
pixels by 1
    # hence the pixels needed are left the same
    # this is to erode the region of the object
detected to draw contours properly
    kernel = np.ones((5, 5), np.uint8)
    mask = cv2.erode(mask, kernel)

    # Contours detection
    if int(cv2.__version__[0]) > 3:
        # Opencv 4.x.x
```

```
        contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  #find the contours on eroded shape of object
    else:
        # Opencv 3.x.x
        _, contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)


    #as long as there are contours run the loop
    for cnt in contours:
        area = cv2.contourArea(cnt) #find area of object surface
        approx = cv2.approxPolyDP(cnt, 0.02 * cv2.arcLength(cnt, True), True) #approximate the contour with an accuracy factor of 0.02

        #if the area detected is large enough, hence an object is detected, draw the contours
        if area > 400:
            cv2.drawContours(frame, [approx], 0, (0, 0, 0), 5)
            if len(approx) == 3: # if 3 lines made the contour hence triangle
                print("Triangle")
                return "Triangle"
            elif 4 <= len(approx) < 6: # if 4 or 5 lines made the contour hence rectangle
                print("Rectangle")
                return "Square"
            else: #if neither a rectangle nor a triangle was detected then the object shape is rejected
                #rejected items variable is called circle to match shapes used with prototype
                print("Circle")
                return "Circle"
    #
    #            cv2.imshow("Frame", frame)
    #            cv2.imshow("Mask", mask)

    cv2.destroyAllWindows()


# Calling code portion for color detection
def Color():

    # call checkColor to check color of the object
    color = checkColor()

    # Based on the returned color the stepper is turned on
    # at a backward direction with a number of steps leading to desired sorting destination
    # after dropping the object using the servo, return the arm to initial position
    # using stepper function at a forward direction
    # Turns on the conveyor belt at a certain conveyor duty for a certain duration

    if color == 'green':
        # We turn on the green LED
        greenlight(1)
        conveyor(conveyor_duty, 1, 0, conveyor_duration)
        stepper("backward", 56)
        drop()
        stepper('forward', 56)
        # We turn off the green LED
        greenlight(0)

    # If the color detected is yellow
    elif color == 'yellow':
        yellowlight(1)
        conveyor(conveyor_duty, 1, 0, conveyor_duration)
        stepper("backward", 112)
        drop()
        stepper('forward', 112)
        # We turn off the yellow LED
        yellowlight(0)
    else:
        # We turn on the red LED
        redlight(1)
        conveyor(conveyor_duty, 0, 1, conveyor_duration)
        # We turn off the red LED
        redlight(0)

# Calling code portion for shape detection
def Shape():
    shape = "none"

    #call check shape function
    shape = checkShape()
    color = "none"
    #call checkcolor to turn colors accordingly
    color = checkColor()

    # based on the returned shape, we turn the proper LED
    # move the stepper for a proper steps
    if shape == 'Triangle':
        conveyor(50, 1, 0, 0.7)
```

```
        if color == "green":
            greenlight(1)
        elif color == "yellow":
            yellowlight(1)
        else:
            redlight(1)
        conveyor(conveyor_duty, 1, 0,
conveyor_duration)
        stepper("backward", 168)
        drop()
        stepper('forward', 168)
        if color == "green":
            greenlight(0)
        elif color == "yellow":
            yellowlight(0)
        else:
            redlight(0)


    elif shape == 'Square':
        conveyor(conveyor_duty, 1, 0,
conveyor_duration)
        if color == "green":
            greenlight(1)
        elif color == "yellow":
            yellowlight(1)
        else:
            redlight(1)
        conveyor(conveyor_duty, 1, 0,
conveyor_duration)
        stepper("backward", 224)
        drop()
        stepper('forward', 224)
        if color == "green":
            greenlight(0)
        elif color == "yellow":
            yellowlight(0)
        else:
            redlight(0)


    else:
        if color == "green":
            greenlight(1)
        elif color == "yellow":
            yellowlight(1)
        redlight(1)
        conveyor(conveyor_duty, 0, 1,
conveyor_duration)
        if color == "green":
            greenlight(0)
        elif color == "yellow":
            yellowlight(0)
        redlight(0)


#functions for turning  the LEDs on/off occurding
to function argument
def greenlight(i):
    IO.setup(38, IO.OUT)  # green LED
    IO.output(38, i)


def yellowlight(i):
    IO.setup(29, IO.OUT)  # yellow LED
    IO.output(29, i)


def redlight(i):
    IO.setup(31, IO.OUT)  # red LED
    IO.output(31, i)


# Main method
# setting proper mode to BOARD and setting up
conveyor inputs
# run the conveyor to place object  under camera
# setting conveyor duration and duty cycle
IO.setmode(IO.BOARD)
IO.setwarnings(False)
IO.setup(13, IO.OUT)  # servopwm
IO.setup(36, IO.OUT)  # Conveyor A
IO.setup(37, IO.OUT)  # Conveyor B
IO.setup(33, IO.OUT)  # conveyor pwmO


conveyor_duration = 0.9
conveyor_duty = 100

conveyor(conveyor_duty, 1, 0, 0.8)
openWindow()
```

Difficulties Faced

- We spent a lot of time installing and uninstalling Operating Systems because we believed that was the cause for the raspberry pi not booting, but later discovered the raspberry pi was inoperable. This caused further difficulties concerning finding a new raspberry pi at a convenient price; we did (obviously).
- The frequent electricity cuts when the pi was on corrupted the operating systems and files multiple times and we had to reinstall the installed libraries all over again.
- The raspberry pi we borrowed as we waited for the one we bought to arrive could not connect to WLAN even though we spent a significant amount of time trying to fix it; we later discovered that it was damaged and WLAN is not available.
- Since the WLAN was broken, and since the routers in our dorm are hung on the ceiling, we kept going back and forth between our dorm and a friend's dorm so we could connect the Raspberry Pi via an ethernet cable.
- For our python code to run, we needed to download OpenCv (CV2), however installations kept failing and we reinstalled the OS many times. Even when the dependencies were installing the version of python needed updating and wouldn't update. Not to mention, CV2 wasn't installing and we tried a significant number of different tutorials on how to install it which ended in failure, until by chance we decided to install numpy which magically solved our issue and all the libraries installed successfully (even though we had numpy installed before), which only worked for a certain OS we installed.
- Being new to python it took some time for us to get accustomed to the syntax and to find libraries and methods that can be used in our implementation and to make them all function properly and in sync; especially since our first contact with python was to implement relatively complex functions.
- For the color and shape detection, the amount of light and angle of the light turned out to be a very sensitive factor affecting the readings; unaware of that initially we kept debugging our code thinking it was the one malfunctioning or incorrect. Not to mention, the camera was taking small details that sometimes can't be seen with the human eye and considering them as contours resulting in wrong readings. In conclusion, tuning the camera proved difficult with the different factors affecting it.