```python
import numpy as np
import cv2
from picamera.array import PiRGBArray
from picamera import PiCamera
import RPi.GPIO as GPIO
import time

statusPin = 18
infoPin = 23
field_of_view = 62
lower_test = np.array([20, 100, 100])
upper_test = np.array([30, 255, 255])
# actual angle = output angle - 31
# we add 31 to simplify the data transmission (avoid 2's complement on MCU's side)
# we add the 31 to simply the data transmission process
def calculateAngle(x_min, x_max):
    # field of view: 62
    center_object = (x_min + x_max)/2 # float normative to 1
    angle = int(field_of_view/2 + field_of_view*(center_object - 0.5)/1)
    return angle

def transmitAngle(angle):
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(statusPin, GPIO.OUT)
    GPIO.setup(infoPin, GPIO.OUT)
    # avoid first high bit issue
    GPIO.output(statusPin, GPIO.LOW)
    GPIO.output(infoPin, GPIO.LOW)
    # convert the angle to binary
    msg = '{0:06b}'.format(angle)
    # turn on status pin, start transmitting
    GPIO.output(statusPin, GPIO.HIGH)
    # time.sleep(0.3)
    # start transmitting signal
    print "Start Transmiiting Angle"
    for i in range(6):
        if test_msg[i] == '1':
            GPIO.output(infoPin, GPIO.HIGH)
        elif test_msg[i] == '0':
            GPIO.output(infoPin, GPIO.LOW)
        # time.sleep(3)
        time.sleep(0.3)
    GPIO.output(statusPin, GPIO.LOW)
    GPIO.output(infoPin, GPIO.LOW)
    # reset all the pins to input mode, protecting Pi
    GPIO.cleanup()
# ------------------------------------------------- #
# Initialize the camera with picamera and warm it up #
# ------------------------------------------------- #
camera = PiCamera() # initialize the camera
camera.rotation = 180 # upsidedown
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480)) #reference to the raw camera capture
time.sleep(0.1) # allow the camera to warmup


for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):

    img = frame.array
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # convert to HSV
    mask = cv2.inRange(hsv, lower_test, upper_test) # apply color filtering
    im2, contours, hierarchy = cv2.findContours(mask, 3, 2) # find contours
    # https://docs.opencv.org/3.4.0/d3/dc0/group__imgproc__shape.html#ga819779b9857cc2f8601e6526a3a5bc71
    # https://docs.opencv.org/3.4.0/d3/dc0/group__imgproc__shape.html#ga4303f45752694956374734a03c54d5ff

    # draw rectangles
    longest = 0
    index = 0
```

```
70      if len(contours) == 0:
71          print "nothing found"
72          continue
73      for i in range(len(contours)):
74          if len(contours[i]) > longest:
75              index = i
76
77      print index
78      cnt = contours[index]
79      x, y, w, h = cv2.boundingRect(cnt)
80      print x, y, w, h
81      cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
82
83      cv2.imshow('original', img)
84      # cv2.imshow('threshed', threshed)
85      cv2.imshow('masked', mask)
86
87
88      rawCapture.truncate(0)
89
90
91
92
93  # with detection_graph.as_default():
94  #   with tf.Session(graph=detection_graph) as sess:
95
96  #         # grab the raw NumPy array representing the image, then initialize the timestamp
97  #         # and occupied/unoccupied text
98  #         image = frame.array
99  #         # test tf code
100 #         # image_np = load_image_into_numpy_array(image)
101 #         image_np_expanded = np.expand_dims(image, axis=0)
102 #         image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
103 #         # Each box represents a part of the image where a particular object was detected.
104 #         boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
105 #         # Each score represent how level of confidence for each of the objects.
106 #         # Score is shown on the result image, together with the class label.
107 #         scores = detection_graph.get_tensor_by_name('detection_scores:0')
108 #         classes = detection_graph.get_tensor_by_name('detection_classes:0')
109 #         num_detections = detection_graph.get_tensor_by_name('num_detections:0')
110 #         # print "get here"
111 #         # Actual detection.
112 #         (boxes, scores, classes, num_detections) = sess.run(
113 #           [boxes, scores, classes, num_detections],
114 #           feed_dict={image_tensor: image_np_expanded})
115 #         # Visualization of the results of a detection.
116 #         # boxes will contain the box coordinates in terms of [0][i][0~3], [ymin, xmin, ymax, xmax]
117 #         # scores will contain the probability in terms of scores[i]
118 #         # the folloiwng four coordinates should be used to calculate angle
119 #         # print "y min: ", boxes[0][0][0]*480
120 #         # print "x min: ", boxes[0][0][1]*640
121 #         # print "y max: ", boxes[0][0][2]*480
122 #         # print "x max: ", boxes[0][0][3]*640
123 #         # print scores
124 #         # calculate angle
125 #         # print type(scores), len(scores), len(boxes) #numpy array, length 1
126 #         # print len(scores[0]), len(boxes[0]) # length 100
127
128 #         # Find the detection of likelihood over 70%.
129 #         qualified_res = boxes[np.where(scores > 0.7)]
130 #         if qualified_res.size == 0:
131 #             print "Nothing detected"
132 #         else:
133 #             print qualified_res[0][1], qualified_res[0][2]
134
135 #         # print boxes[np.where(scores > 0.7)][1], boxes[np.where(scores > 0.7)][2]
136
137
138
139
```

```
140 #         # Visualization of Detection Result
141 #         # print boxes
142 #         # image.setflags(write=1)
143 #         # vis_util.visualize_boxes_and_labels_on_image_array(
144 #         #   image,
145 #         #   np.squeeze(boxes),
146 #         #   np.squeeze(classes).astype(np.int32),
147 #         #   np.squeeze(scores),
148 #         #   category_index,
149 #         #   use_normalized_coordinates=True,
150 #         #   line_thickness=8)
151 #   #         # show the frame
152 #         # cv2.imshow("Frame", image)
153 #         # key = cv2.waitKey(1) & 0xFF
154 
155 #         # # clear the stream in preparation for the next frame
156 #         rawCapture.truncate(0)
157 
158 #         # # if the `q` key was pressed, break from the loop
159 #         if key == ord("q"):
160 #           break
```