# Raspberry PI Image Processing Code

import cv2

import math

import numpy as np

import RPi.GPIO as GPIO

cap = cv2.VideoCapture(0) # start camera

GPIO.setmode(GPIO.BCM) # set GPIO addressing format

GPIO.setwarnings(False) # neglect warnings

GPIO.setup(18, GPIO.OUT) # set pins as output

GPIO.setup(23, GPIO.OUT)

GPIO.setup(24, GPIO.OUT)

low\_red = np.array([161, 155, 84]) # red color range lower bound

high\_red = np.array([179, 255, 255]) # red color range upper bound

# pixelScanDivider > 1 🡪 pixel array nested while loops to change step size (for faster scan but poor #accuracy)

pixelScanDivider = 32

center\_error = 50 # range of center detection [center - center\_error ,center + center\_error]

max\_d = 250 # max diameter of ball (arrived to ball) in pixels

check\_loop\_count = 2 # check if arrived (to filter noise)

count = 1 # count check initial value

while True:

\_,frame = cap.read() # read pixels into array 'frame'

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV) # convert from BGR to HSV

red\_mask = cv2.inRange(hsv, low\_red, high\_red) # find pixels in red range and blacken the others

red = cv2.GaussianBlur(red\_mask,(5,5),0) # filter out unwanted pixels

u = [-1,-1] #[x,y] <-- upper pixel

d = [-1,-1] #[x,y] <-- down pixel

r = [-1,-1] #[x,y] <-- right pixel

l = [-1,-1] #[x,y] <-- left pixel

# what comes next is explained in the report

k = 0

y = 0

while pixelScanDivider \* k < red.shape[0]:

y = pixelScanDivider \* k

j = 0

x = 0

while pixelScanDivider \* j < red.shape[1]:

x = pixelScanDivider \* j

i = red[y][x]

if i[\_] != 0:

if u == [-1,-1]:

u = [x,y]

x1 = red.shape[1]-x-1

y1 = red.shape[0]-y-1

i = red[y1][x1]

if i[\_] != 0:

if d == [-1,-1]:

d = [x1,y1]

x1 = int(round(red.shape[1]-(y \* ((red.shape[1]-1) / (red.shape[0]-1)))-1))

y1 = int(round(x \* ((red.shape[0]-1) / (red.shape[1]-1))))

i = red[y1][x1]

if i[\_] != 0:

if r == [-1,-1]:

r = [x1,y1]

x1 = int(round(y \* ((red.shape[1]-1) / (red.shape[0]-1))))

y1 = int(round(x \* ((red.shape[0]-1) / (red.shape[1]-1))))

i = red[y1][x1]

if i[\_] != 0:

if l == [-1,-1]:

l = [x1,y1]

j = j + 1

k = k + 1

if u[\_] == -1 or d[\_] == -1 or l[\_] == -1 or r[\_] == -1: # check to see if not detected (search)

GPIO.output(18,GPIO.LOW)

GPIO.output(23,GPIO.LOW)

GPIO.output(24,GPIO.LOW)

else:

GPIO.output(18,GPIO.HIGH)

cv2.rectangle(frame, (l[0], u[1]), (r[0], d[1]), (150,0,0), 2) # draw rectangle around ball

i = (l[0]+r[0])/2 # find center x coordinate

ctr = (red.shape[1]-1)/2 # find screen center

d = math.sqrt(pow(r[0]-l[0], 2)+pow(d[1]-u[1], 2)) # find ball size (proportional to distance)

d\_f = (d / max\_d) + 1 # coefficient to vary the center range detection (1 -> 2+)

rp = ctr+(center\_error\*d\_f) # calculating new center range upper bound

rm = ctr-(center\_error\*d\_f) # calculating new center range lower bound

# draw line to view center range lower bound

cv2.line(frame, (int(rm), 0), (int(rm), red.shape[0]-1), (0,0,255), 2)

# draw line to view center range upper bound

cv2.line(frame, (int(rp), 0), (int(rp), red.shape[0]-1), (0,0,255), 2)

if d < max\_d: # check if ball is not in grabbing range

count = 1

if rm <= i <= rp: # check if in center range (go straight)

GPIO.output(23,GPIO.HIGH)

GPIO.output(24,GPIO.HIGH)

if 0 <= i < rm: # check if on the left (go left)

GPIO.output(24,GPIO.HIGH)

GPIO.output(23,GPIO.LOW)

if rp < i <= red.shape[1]-1: # check if on the right (go right)

GPIO.output(23,GPIO.HIGH)

GPIO.output(24,GPIO.LOW)

else:

if count > check\_loop\_count: # check if arrived (move arm)

GPIO.output(23,GPIO.LOW)

GPIO.output(24,GPIO.LOW)

elif count <= check\_loop\_count: # counter to check if ball is really in grabbing or just noise

count = count + 1

cv2.imshow("1", frame) # show camera view in window

key = cv2.waitKey(1) # if 'ESC' is pressed exit infinite loop

if key == 27:

break

GPIO.output(18,GPIO.LOW) # reset LEDs

GPIO.output(23,GPIO.LOW)

GPIO.output(24,GPIO.LOW)

cap.release() # stop camera

cv2.destroyAllWindows() # close camera view window

# Raspberry PI Image Processing Explanations

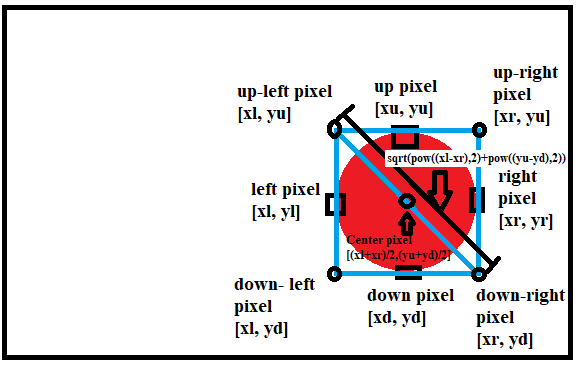
After processing the image to find the red ball, we still must find the coordinates of the center of the ball and its size (which is directly proportional to its distance from the camera). This can be done by looping on all pixels and mapping, as we go, the x and y coordinates relatively to the other three corners. We then let the nested loops close in on the object until the four extremities of the ball are found in which case, we can draw a rectangle over its area and calculate the values needed as follows:

Figure Ball position and size computation

The rest of the code is explained in the code as comments. Last importing thing is detection ranges along the width of the camera view. We need to assign a left area, a right area and a centered area. We also need to resize the centered range to proportionally match the size of the ball to increase accuracy (explained in the code). A couple examples of that would be:

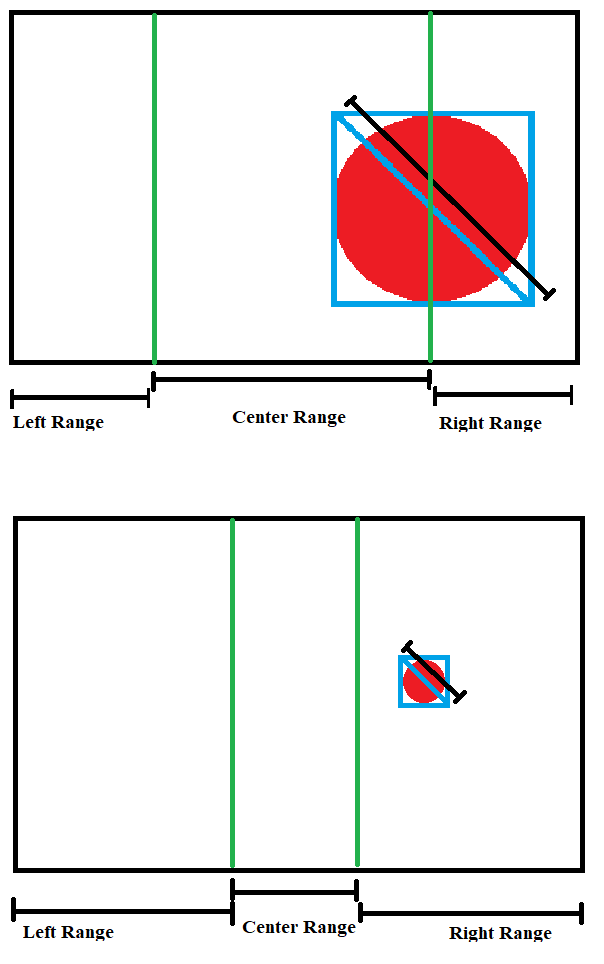


Figure Camera range divisions examples