In [3]:
```
pip install opencv-python numpy
```

```
   --- ------------------------------------ 3.7/38.8 MB 3.2 MB/s eta
0:00:11
   --- ------------------------------------ 3.9/38.8 MB 3.2 MB/s eta
0:00:11
   ---- ----------------------------------- 4.0/38.8 MB 3.2 MB/s eta
0:00:11
   ---- ----------------------------------- 4.1/38.8 MB 3.2 MB/s eta
0:00:11
   ---- ----------------------------------- 4.3/38.8 MB 3.2 MB/s eta
0:00:11
   ---- ----------------------------------- 4.5/38.8 MB 3.2 MB/s eta
0:00:11
   ---- ----------------------------------- 4.6/38.8 MB 3.2 MB/s eta
0:00:11
   ---- ----------------------------------- 4.7/38.8 MB 3.2 MB/s eta
0:00:11
   ----- ---------------------------------- 4.9/38.8 MB 3.2 MB/s eta
0:00:11
   ----- ---------------------------------- 5.0/38.8 MB 3.2 MB/s eta
```

In [ ]:

```
from __future__ import division
import io
import os
import random
import cv2
import numpy as np
import time
from copy import deepcopy

kernelOpen = np.ones((5, 5))
kernelClose = np.ones((20, 20))

# The name of the image file to annotate
i = time.strftime("%d-%m-%y_%H-%M-%S")

# Capture image
camera = cv2.VideoCapture(0)
return_value, image = camera.read()
cv2.imwrite(i + '.jpeg', image)
del(camera)
frame = image
edge_img = deepcopy(image)

# Finds edges in the input image and marks them in the output map edges
edged = cv2.Canny(edge_img, 50, 100)
edged = cv2.dilate(edged, None, iterations=1)
edged = cv2.erode(edged, None, iterations=1)

# Find contours in the edge map
cnts, h = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPRO

max_cont = max(cnts, key=cv2.contourArea)
x, y, w, h = cv2.boundingRect(max_cont)
cv2.rectangle(edge_img, (x, y), (x + w, y + h), (0, 0, 255), 2)
croppedk = frame[y:y + h, x:x + w]

# Display the edges
cv2.imshow('Edges', edge_img)

frame = edge_img

# Converting BGR to HSV
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Define range of red color in HSV
lower_red1 = np.array([0, 50, 50])
upper_red1 = np.array([10, 255, 255])
lower_red2 = np.array([170, 50, 50])
upper_red2 = np.array([180, 255, 255])

# Create a red HSV colour boundary and threshold HSV image
redmask1 = cv2.inRange(hsv, lower_red1, upper_red1)
redmask2 = cv2.inRange(hsv, lower_red2, upper_red2)
redmask = redmask1 + redmask2
maskOpen = cv2.morphologyEx(redmask, cv2.MORPH_OPEN, kernelOpen)
maskClose = cv2.morphologyEx(maskOpen, cv2.MORPH_CLOSE, kernelClose)

maskFinal = maskClose
cv2.imshow('Red Mask', maskFinal)

# Calculate redness using contours
```

```
cnt_r, _ = cv2.findContours(maskFinal, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
cnt_r_area = sum(cv2.contourArea(c) for c in cnt_r)
print("Redness:", cnt_r_area)

lower_green = np.array([50, 50, 50])
upper_green = np.array([70, 255, 255])
greenmask = cv2.inRange(hsv, lower_green, upper_green)
cv2.imshow('Green Mask', greenmask)
cnt_g = cv2.countNonZero(greenmask)
print("Greenness:", cnt_g)

lower_yellow = np.array([20, 50, 50])
upper_yellow = np.array([50, 255, 255])
yellowmask = cv2.inRange(hsv, lower_yellow, upper_yellow)
cv2.imshow('Yellow Mask', yellowmask)
cnt_y = cv2.countNonZero(yellowmask)
print("Yellowness:", cnt_y)

# Calculate ripeness
tot_area = cnt_r_area + cnt_y + cnt_g
rperc = cnt_r_area / tot_area
yperc = cnt_y / tot_area
gperc = cnt_g / tot_area

# Adjust the limits for your fruit
glimit = 0.1
ylimit_low = 0.3
ylimit_high = 0.5
if yperc * 100 < 10 or gperc * 100 < 10:
    print("Fruit not detected")
else:
    if gperc > glimit:
        print(f"Unripe ({gperc * 100:.2f}% Unripe)")
    else:
        print(f"Ripe ({yperc * 100:.2f}% Ripe)")

# Wait for any key to close
while True:
    k = cv2.waitKey(5) & 0xFF
    if k != 255:  # If any key is pressed
        break

# De-allocate any associated memory usage
cv2.destroyAllWindows()
```

```
Redness: 12.0
Greenness: 49
Yellowness: 108
Unripe (28.99% Unripe)
```

In [ ]:

```
from __future__ import division
import io
import os
import random
import cv2
import numpy as np
import time
from copy import deepcopy

kernelOpen = np.ones((5, 5))
kernelClose = np.ones((20, 20))

# The name of the image file to annotate
i = time.strftime("%d-%m-%y_%H-%M-%S")

# Capture image
camera = cv2.VideoCapture(0)
return_value, image = camera.read()
cv2.imwrite(i + '.jpeg', image)
del(camera)
frame = image
edge_img = deepcopy(image)

# Finds edges in the input image and marks them in the output map edges
edged = cv2.Canny(edge_img, 50, 100)
edged = cv2.dilate(edged, None, iterations=1)
edged = cv2.erode(edged, None, iterations=1)

# Find contours in the edge map
cnts, h = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPRO

max_cont = max(cnts, key=cv2.contourArea)
x, y, w, h = cv2.boundingRect(max_cont)
cv2.rectangle(edge_img, (x, y), (x + w, y + h), (0, 0, 255), 2)
croppedk = frame[y:y + h, x:x + w]

# Display the edges
cv2.imshow('Edges', edge_img)

frame = edge_img

# Converting BGR to HSV
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Define range of red color in HSV
lower_red1 = np.array([0, 50, 50])
upper_red1 = np.array([10, 255, 255])
lower_red2 = np.array([170, 50, 50])
upper_red2 = np.array([180, 255, 255])

# Create a red HSV colour boundary and threshold HSV image
redmask1 = cv2.inRange(hsv, lower_red1, upper_red1)
redmask2 = cv2.inRange(hsv, lower_red2, upper_red2)
redmask = redmask1 + redmask2
maskOpen = cv2.morphologyEx(redmask, cv2.MORPH_OPEN, kernelOpen)
maskClose = cv2.morphologyEx(maskOpen, cv2.MORPH_CLOSE, kernelClose)

maskFinal = maskClose
cv2.imshow('Red Mask', maskFinal)

# Calculate redness using contours
```

```
cnt_r, _ = cv2.findContours(maskFinal, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
cnt_r_area = sum(cv2.contourArea(c) for c in cnt_r)
print("Redness:", cnt_r_area)

lower_green = np.array([50, 50, 50])
upper_green = np.array([70, 255, 255])
greenmask = cv2.inRange(hsv, lower_green, upper_green)
cv2.imshow('Green Mask', greenmask)
cnt_g = cv2.countNonZero(greenmask)
print("Greenness:", cnt_g)

lower_yellow = np.array([20, 50, 50])
upper_yellow = np.array([50, 255, 255])
yellowmask = cv2.inRange(hsv, lower_yellow, upper_yellow)
cv2.imshow('Yellow Mask', yellowmask)
cnt_y = cv2.countNonZero(yellowmask)
print("Yellowness:", cnt_y)

# Calculate ripeness
tot_area = cnt_r_area + cnt_y + cnt_g
rperc = cnt_r_area / tot_area
yperc = cnt_y / tot_area
gperc = cnt_g / tot_area

# Adjust the limits for your fruit
glimit = 0.1
ylimit_low = 0.3
ylimit_high = 0.5

if yperc * 100 < 10 or gperc * 100 < 10:
    print("Fruit not detected")
else:
    print("Fruit detected")
    if gperc > glimit:
        print(f"Unripe ({gperc * 100:.2f}% Unripe)")
    else:
        print(f"Ripe ({yperc * 100:.2f}% Ripe)")

# Wait for any key to close
while True:
    k = cv2.waitKey(5) & 0xFF
    if k != 255:  # If any key is pressed
        break

# De-allocate any associated memory usage
cv2.destroyAllWindows()
```

```
Redness: 6146.5
Greenness: 944
Yellowness: 52023
Fruit not detected
```

In [ ]:

In [ ]: