find_cone

January 23, 2023

[1]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

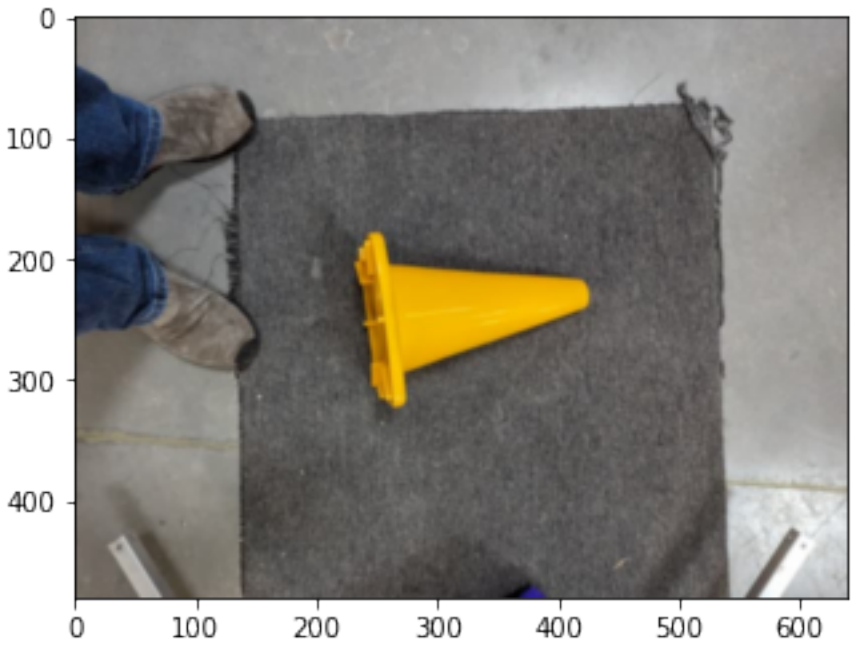
```
␣
↪---------------------------------------------------------------------------

ModuleNotFoundError                       Traceback (most recent call␣
↪last)

<ipython-input-1-223d3e2cdfe8> in <module>
----> 1 import cv2 as cv
      2 import numpy as np
      3 import matplotlib.pyplot as plt


ModuleNotFoundError: No module named 'cv2'
```

[564]:
```
def show(bgr):
    rgb = cv.cvtColor(bgr, cv.COLOR_BGR2RGB)
    plt.imshow(rgb)
```

1 Reading the image and blur

Below we load an image, we also blur the image. Often times you don't want to work with the full resolution image. Blurring removes high frequency content, which can help to remove noise artifacts in the processing.

[565]:
```
img = cv.imread('data/img01.jpg')
blur = cv.blur(img, (3,3))
show(blur)
```



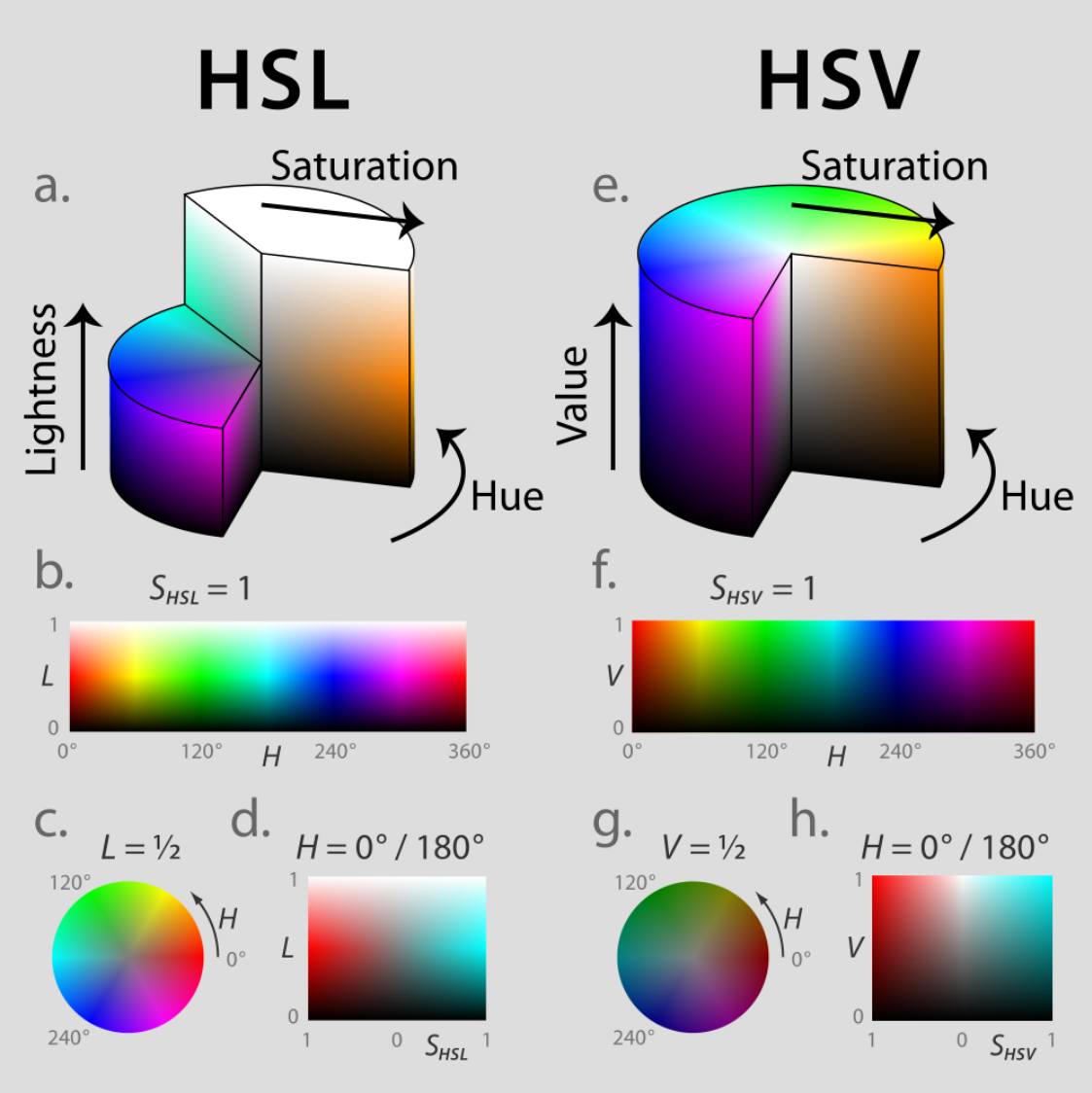
2 HSV Color Space

Normal images are captured by image sensors with three color channels: Red, Green, Blue.

Note: If you want to get really pedanic, image sensors capture a bayer pattern of RGGB, you then have to do demosaicing and all kinds of processing to get the RGB image we are used to.

Once we have the RGB image, we can select known cone pixels using Photoshop or the Open Source alternative GIMP. When we do this we see the RGB value is 95, 72, and 1 respectively. OpenCV, the worlds most awesomest computer vision library stores images in Blue, Green, Red format for historical reasons.

So we can convert that BGR image into the Hue, Saturation, and Value colorspace HSV



When we do this, we see the Hue and Saturation are 23 and 252. We can now plot a histogram of the entire image above.

[566]:
```
bgr_cone = np.zeros((1, 1, 3), 'uint8')
bgr_cone[0,0,0] = 1
bgr_cone[0,0,1] = 72
bgr_cone[0,0,2] = 95
hls_cone = cv.cvtColor(bgr_cone, cv.COLOR_BGR2HSV)
print(bgr_cone)
print(hls_cone)
print(hls_cone.dtype)
```

```
[[[ 1 72 95]]]
[[[ 23 252  95]]]
uint8
```

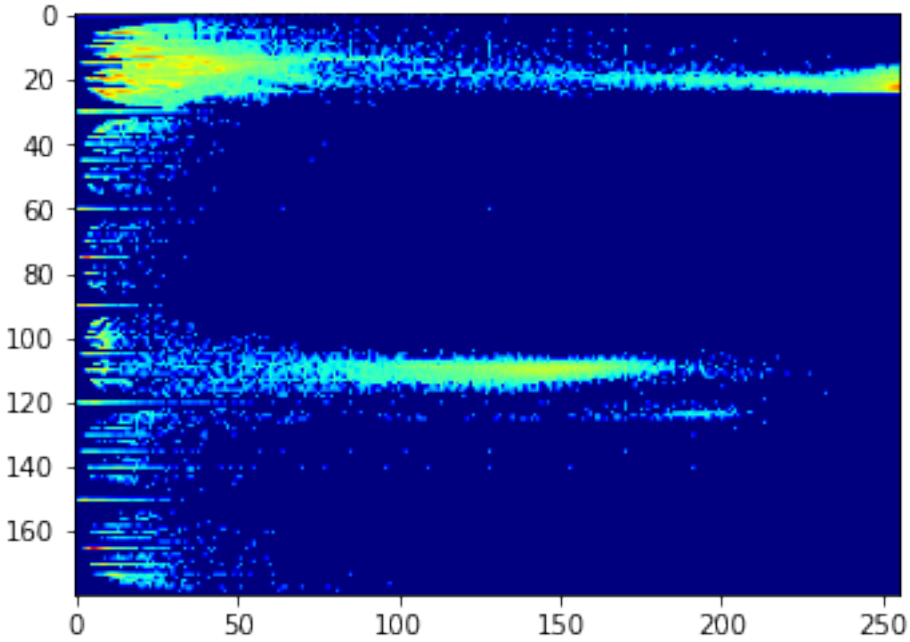
3 Histograms

Histograms are a way of counting how many times something occurs in a dataset. Below we are captureing a histogram of the hue and saturation of the entire image. This gives us another "image". On the X-axis is saturation and the Y axis is Hue. So we know the cone has a saturation of 252 and a hue of 23. We can look up in the histogram and find that has a high concentration. Meaning there are a lot of pixels that fit in that spot on our graph below.

Note: We plot it with log to bring out the data for us humans.

[567]:
```
hsv = cv.cvtColor(blur, cv.COLOR_BGR2HSV)
hist = cv.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])
cv.minMaxLoc(hsv[:,:,1])
plt.imshow(np.log(hist+0.01), cmap='jet')
```

[567]:
```
<matplotlib.image.AxesImage at 0x7f3a47676760>
```

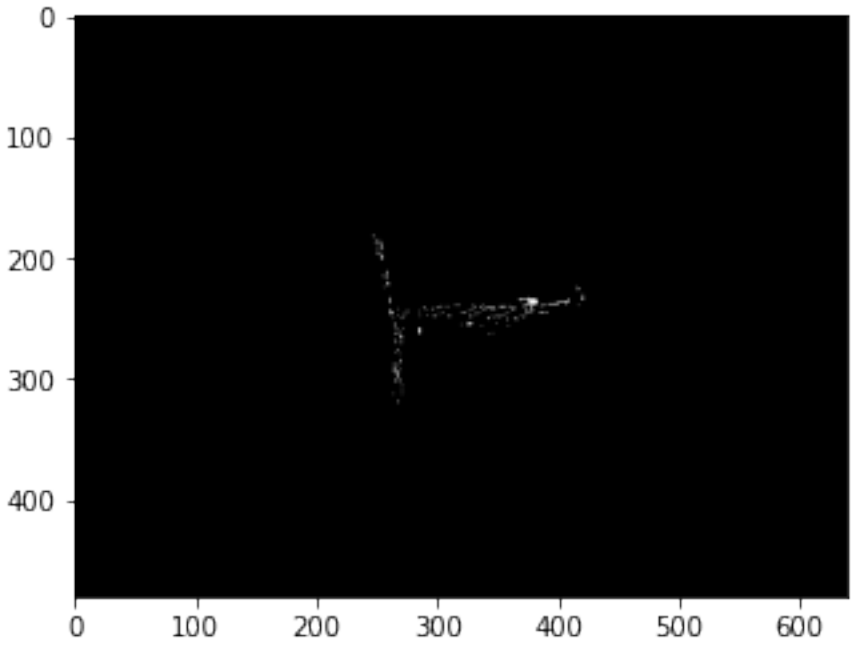


4 Backproject

We can take our selected color and back project it into the original image. This will select pixels in the image that have that exact color range. Lets try it with the selected color above.

[568]:
```
bphist = np.zeros((180, 256), 'float32')
bphist[23, 252] = 1
mask = cv.calcBackProject([hsv], [0, 1], bphist, [0, 180, 0, 256], 1)
plt.imshow(mask, cmap='gray')
```

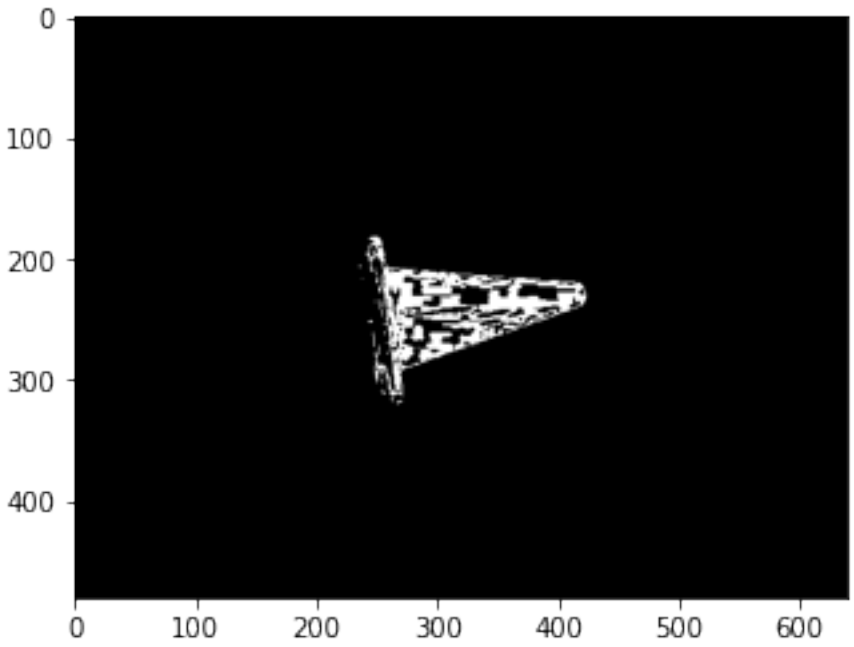
[568]: <matplotlib.image.AxesImage at 0x7f3a475e3700>



We can see that not very many pixels were detected. We can expand our selection which will pick up more pixels.

[569]:
```
bphist = np.zeros((180, 256), 'float32')
bphist[20:26, 249:255] = 1
mask = cv.calcBackProject([hsv], [0, 1], bphist, [0, 180, 0, 256], 1)
plt.imshow(mask, cmap='gray')
```

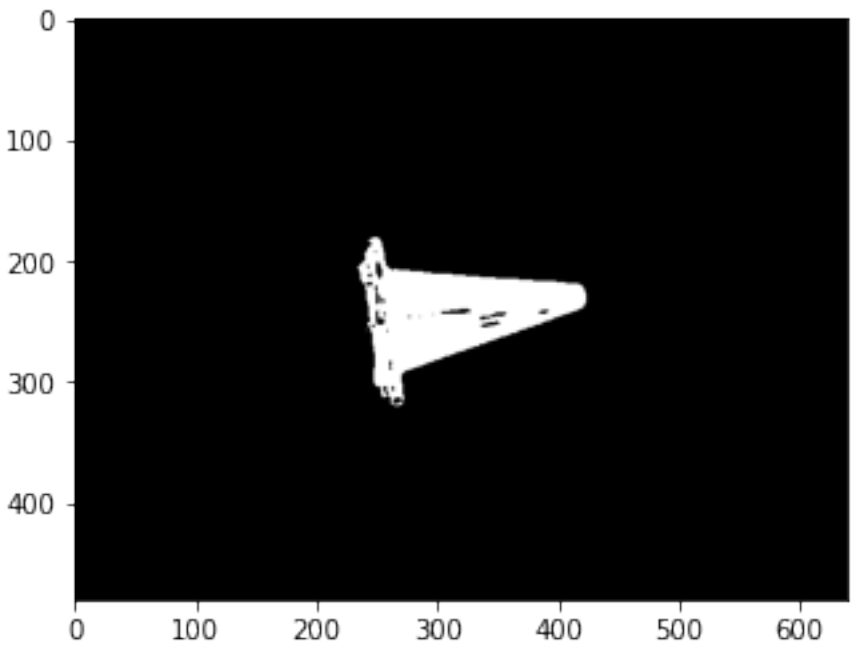
[569]: <matplotlib.image.AxesImage at 0x7f3a47544760>



We are selecting more pixels, lets increase the range further

[570]:
```
bphist = np.zeros((180, 256), 'float32')
bphist[13:33, 242:262] = 1
mask = cv.calcBackProject([hsv], [0, 1], bphist, [0, 180, 0, 256], 1)
plt.imshow(mask, cmap='gray')
```

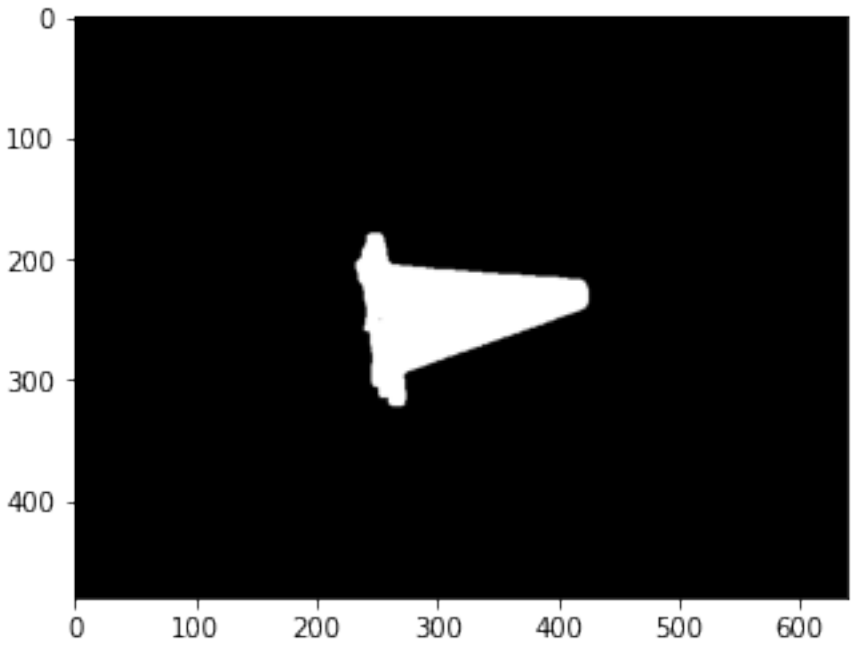
[570]:
```
<matplotlib.image.AxesImage at 0x7f3a47524790>
```



This is a pretty good attempt. We can fill in the blanks with something called morphological operations. We will try the **dilate** morph op

[571]:
```
kernel = cv.getStructuringElement(cv.MORPH_RECT, (5,5))
dilate = cv.dilate(mask, kernel)
plt.imshow(dilate, cmap='gray')
```

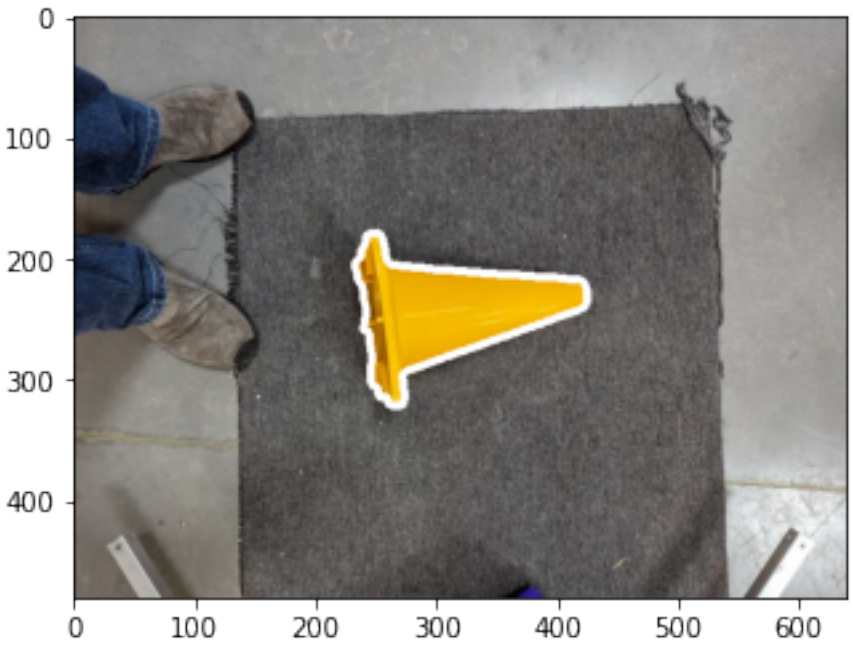
[571]: `<matplotlib.image.AxesImage at 0x7f3a4748a430>`



We now have a mask for the cone. The next thing we want to do is try and determine the orientation. We can get a bounding rectangle of this object and find the rotation of the rectangle.

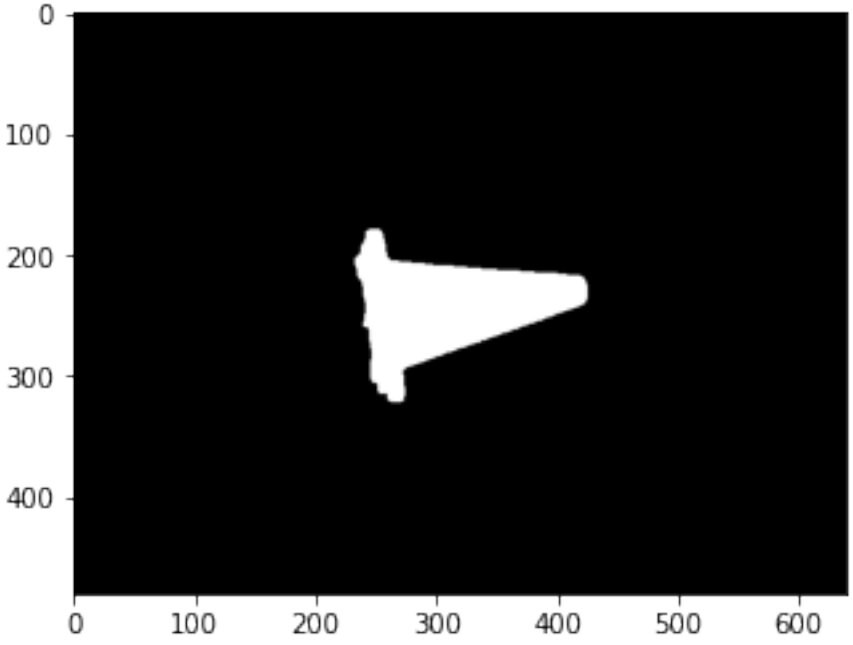
[572]:
```
contours, hier = cv.findContours(dilate, cv.RETR_EXTERNAL, cv.
↪CHAIN_APPROX_SIMPLE)
for i, c in enumerate(contours):
    area = cv.contourArea(c)
    print(f'Area: {area}')
    if area > 1000:
        simple_contour = cv.approxPolyDP(c, 1, True)
        print(len(simple_contour))
        draw = img.copy()
        draw = cv.drawContours(draw, contours, i, (255, 255, 255), 5)
        mask = np.zeros((img.shape[0], img.shape[1]), 'uint8')
        cv.drawContours(mask, contours, i, 255, -1)
        show(draw)
```

```
Area: 12129.0
41
```



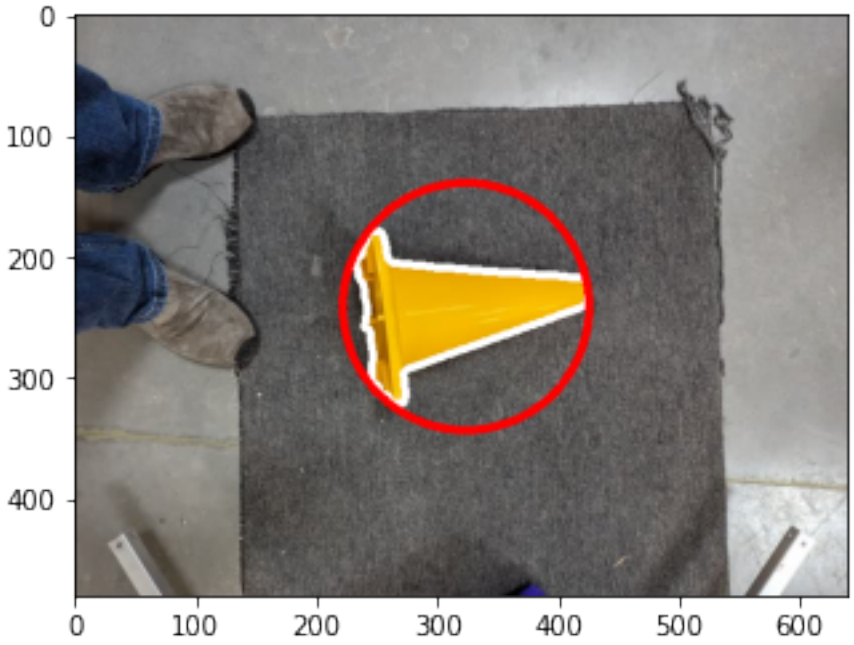
[573]:
```
plt.imshow(mask, cmap='gray')
```

[573]: <matplotlib.image.AxesImage at 0x7f3a473c93d0>



[574]:
```
center, radius = cv.minEnclosingCircle(simple_contour)
center = np.round(center, 0).astype('int')
radius = np.round(radius, 0).astype('int')
print(center)
print(radius)
draw = cv.circle(draw, center, radius, (0, 0, 255), 5)
show(draw)
```

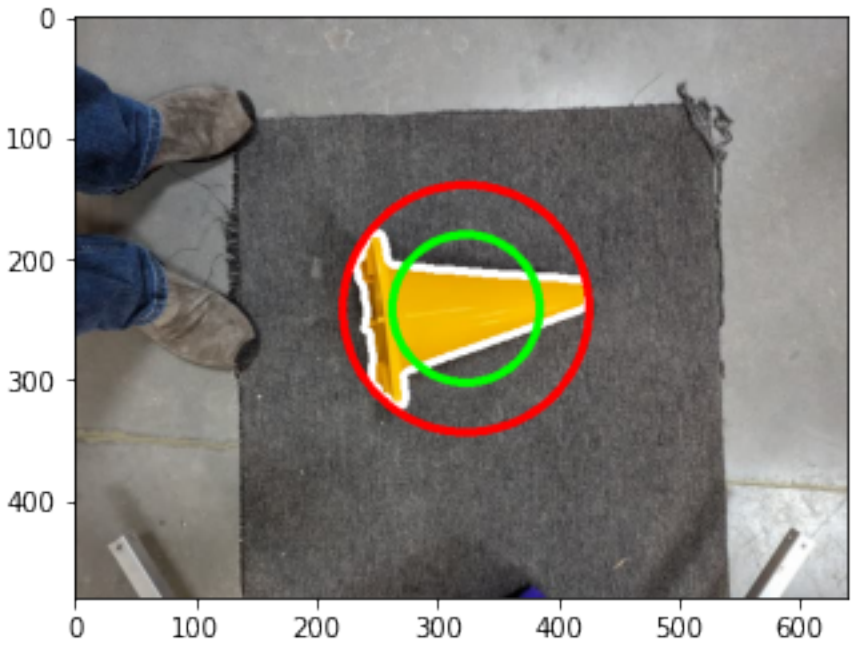
```
[324 241]
102
```



We now have a circle that surrounds the cone. We can now reduce the circle radius by 10% and plot the pixels that are on the circle, starting with the top pixels

[575]:
```
radius = int(round(radius * 0.6, 0))
print(center)
print(radius)
draw = cv.circle(draw, center, radius, (0, 255, 0), 5)
show(draw)
```

```
[324 241]
61
```



5 Sin and Cos goodness

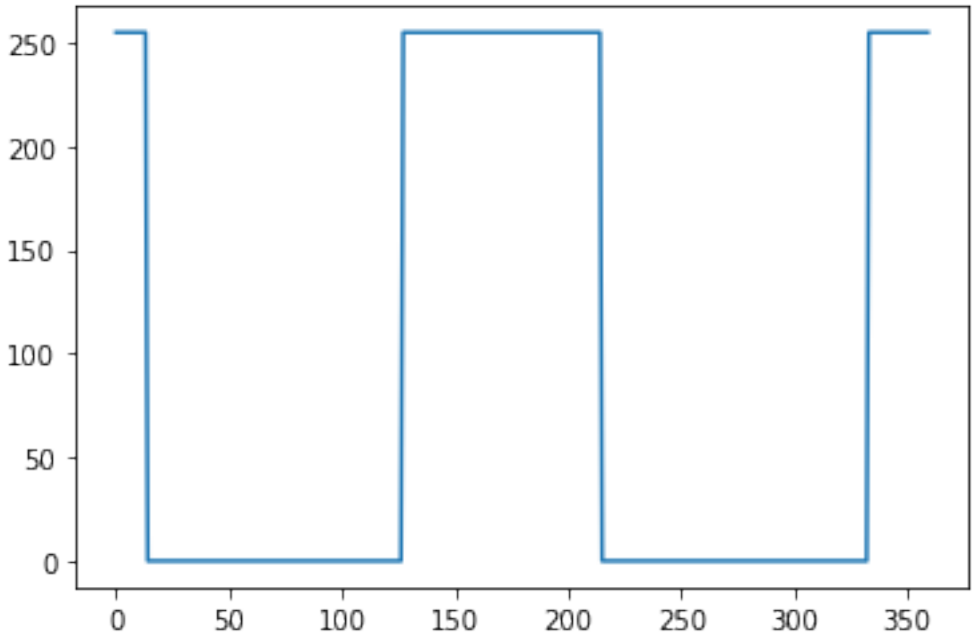
I really like Eulers identity: $e^{i\pi} + 1 = 0$

With this simple equation you can derive all of the sin, cos, tan, etc identities.

So what we want to do is grab all of the pixels on the green line above. So we will generate an array of radians (the real way to calculate degrees). OpenCV has a helper function to go from magnitude and radians to the cartesian coordinates. We then need to add in the center of the circle. Now we can grab those X/Y offsets into a line and plot it. We will use the dilated image.

[576]:
```
radians = np.linspace(0, 2*np.pi, 360)
radius_array = np.ones(360) * radius
x, y = cv.polarToCart(radius_array, radians)
x = np.round(x, 0).astype('int') + center[0]
y = np.round(y, 0).astype('int') + center[1]
line = mask[y, x]
line = line[:,0]
plt.plot(line)
#plt.plot(x)
#plt.plot(y)
```

[576]:
```
[<matplotlib.lines.Line2D at 0x7f3a472f90d0>]
```



We now have a binary value that is high when we are on the cone and low when we are not. But in math, 0 degrees is on the right middle of the cicle. 90 degrees if you are looking at a compass (or three o-clock if you are looking at a clock).

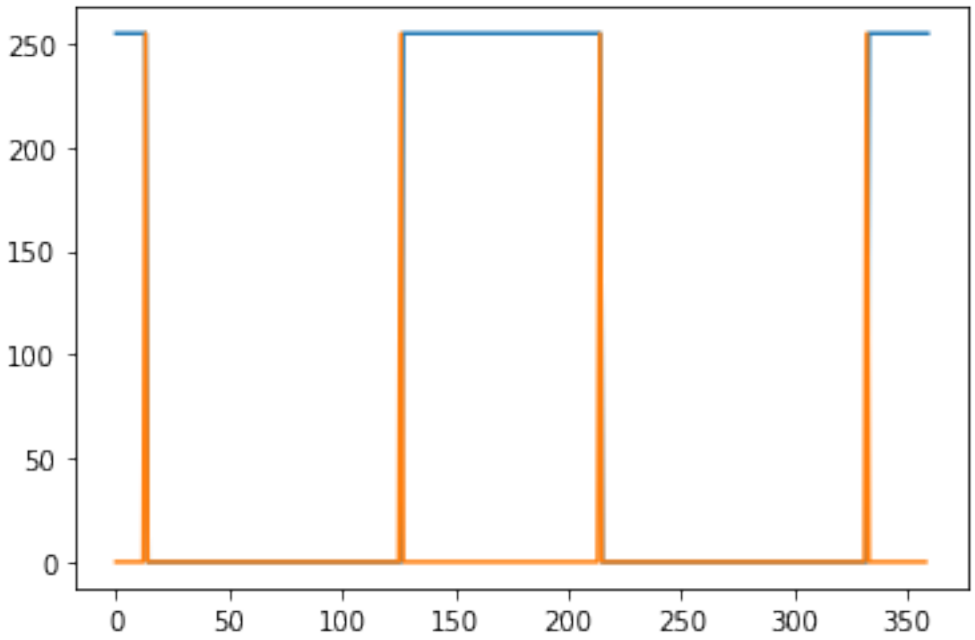
Looking at the graph above, we start at the top of the cone, then as we rotate clockwise we get back to the carpet, then the base, then carpet, then the tip.

Now lets start at 0 and calculate how many degrees the signal is high.

[577]:
```
fline = line.astype('float32')
diff = np.diff(fline)
diff = abs(diff)
plt.plot(line)
plt.plot(diff)
nonzero = np.nonzero(diff > 0)[0]
print(nonzero)
if line[0] > 0:
    print('Line starts high')
    # The cone is detected
    det1 = nonzero[2] - nonzero[1]
    det2 = (360 - nonzero[3]) + nonzero[0]
    angle1 = nonzero[1] + (det1 / 2)
    angle2 = nonzero[3] + (det2 / 2)
    if angle2 >= 360:
        angle2 -= 360
else:
    det1 = nonzero[1] - nonzero[0]
```

```
    det2 = nonzero[3] - nonzero[2]
    angle1 = nonzero[0] + (det1 / 2)
    angle2 = nonzero[2] + (det2 / 2)

print(f'Det1: {det1} Angle1: {angle1} Det2: {det2} Angle2: {angle2}')
if det1 < det2:
    print('Narrow part is det1')
    mangle = angle1
else:
    print('Narrow part is det 2')
    mangle = angle2
```

```
[ 13 126 214 332]
Line starts high
Det1: 88 Angle1: 170.0 Det2: 41 Angle2: 352.5
Narrow part is det 2
```

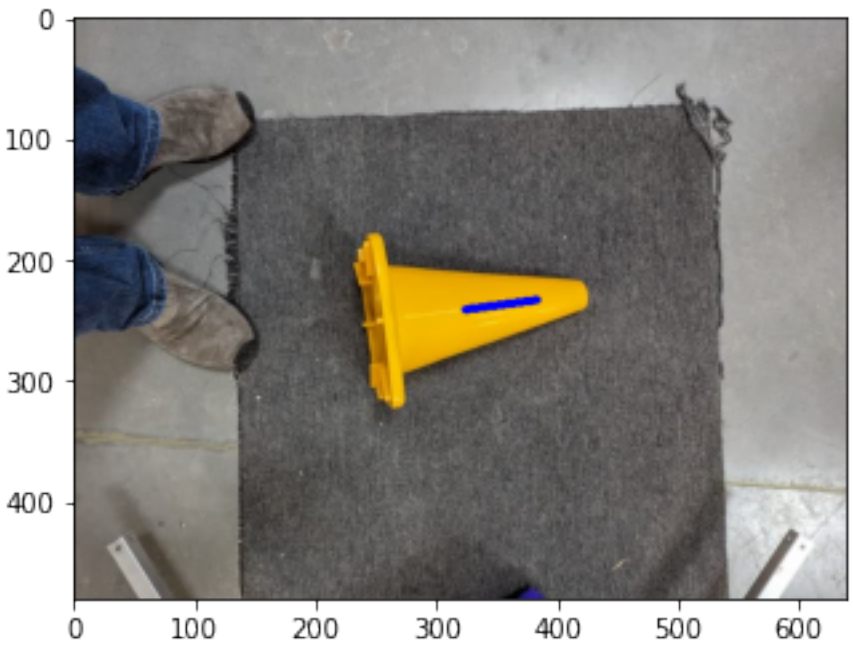


Once again, 0 degrees is the 3 o'clock position. So we need to subtract 90 degrees from our calculation

[578]:
```
angle = (mangle + 90)
if angle > 360:
    angle -= angle
draw = img.copy()
x, y = cv.polarToCart(radius, mangle / 180 * np.pi)
print(x)
```

```
print(y)
pt1 = x[0] + center[0]
pt2 = y[0] + center[1]
pt = np.array((pt1[0], pt2[0])).astype('int')
print(pt)
cv.line(draw, center, pt, (255, 0, 0), 5)
print(f'Cone angle is {angle}')
show(draw)
```

```
[[60.47813654]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]]
[[-7.96209773]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]]
[384 233]
Cone angle is 0.0
```



[ ]: