slides

January 4, 2019

In [23]:
```
# Note: Before starting the slideshow, make sure you run this code as it
# provides helper functions that the other slides need...
#
# Other than that, you can ignore this content as it won't show up in the slideshow.
#
# It's a helper function that makes it easier to show OpenCV images directly
# in the notebook environment. When using OpenCV locally, you'll want to use
# `cv2.imshow(name, img)` instead.

import numpy as np
import cv2
import math

# Notebook setup + convenience functions
%matplotlib inline
import matplotlib.pyplot as plt

def force_bgr(img):
    '''Forces image to 3-channel representation if grayscale'''
    if len(img.shape) == 2 or img.shape[2] == 1:
        return cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
    return img

def imshow(*args):
    '''Helper function to show images, because matplotlib and OpenCV aren't a perfect m
    fig = plt.figure()
    for i, img in enumerate(args):
        fig.add_subplot(1,len(args),i+1)
        plt.imshow(cv2.cvtColor(force_bgr(img), cv2.COLOR_BGR2RGB))

def blur(src, radius):
    ksize = int(2 * round(radius) + 1)
    return cv2.blur(src, (ksize, ksize))
```

#
Image processing using
##
OpenCV + Python

Tim Winters
Created by Dustin Spicuzza (Team 2423/1418)
September 10, 2016
NE FIRST University Day

1 Agenda

- Why OpenCV + Python?
- Image filtering demo
- pynetworktables

2 Image processing

- FRC Teams do it a lot of ways
  - NIVision (LabVIEW)
  - GRIP (Uses OpenCV as engine)
  - OpenCV (various custom stuff)
- We're going to talk about OpenCV

3 Why OpenCV?

- Originally developed by Intel
- It has thousands of image processing related algorithms and functions available
- Highly optimized and reliable
- Has building blocks that fit together
- Lets you do complex image processing without needing to understand the math
- If you understand the math, it helps!

4 Why OpenCV?

- Bindings for multiple languages
  - C/C++
  - Java
  - Python
- Multiple platforms supported
  - Windows
  - Linux
  - OSX
  - Android
- Oh, and it's **FREE**!

5 What OpenCV Provides

- Image I/O:
  - Read/Write images from disk
  - Use native OS functionality to interface with cameras
- Image Segmentation
  - Edge finding
  - Contour detection
  - Thresholding

6 What OpenCV Provides

- Face detection
- Motion tracking
- Stereo vision support
- Support for GPU acceleration
- Machine learning operations
  - Classifiers
  - Neural networks

7 What OpenCV Provides

- Distributed with lots of useful samples that you can use to figure out how OpenCV works
  - Face detection
  - Edge finding
  - Histograms
  - Square finder

Lots and lots and lots of stuff...

8 Why Python + OpenCV?

- Python is really easy to learn and use
  - Simple syntax
  - Rapid prototyping
- Most of the compute intensive work is implemented in C/C++
  - Python is just glue, realtime operation **is** possible
- NumPy is awesome
  - Manipulating image data is trivial compared to other OpenCV bindings (Java, C++)

#
Time to CODE!
Go to http://goo.gl/nB0NCG

9 About this environment

http://goo.gl/nB0NCG

- It's a Jupyter Notebook (formerly IPython Notebook)
  - This slideshow uses Jupyter too!
- It allows you to mix text and executable code in a webpage
- You execute each cell using SHIFT-ENTER

10 Hello World!

- Click the cell with the following text, and press SHIFT-ENTER

```
In [24]: print("Hello class")
```

```
Hello class
```

11 Next Steps

- Execute the helper code

- The next cell tells you about the images available in your environment

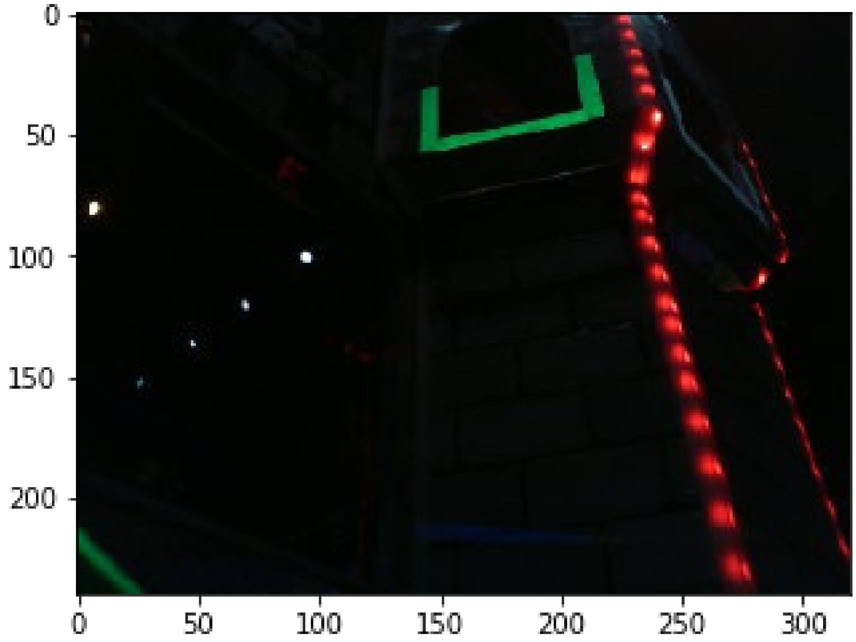
```
In [25]: %ls images
```

```
1ftH3ftD2Angle0Brightness.jpg* 2016-cmp-5.jpg
2013-f0.png                    2016-dcmp1.jpg
2013-p0.png                    2016-dcmp2.jpg
2013-p1.png                    2016-dcmp3.jpg
2014-f0.png                    2016-p0.jpg
2016-cmp-0.jpg                 2016-p1.jpg
2016-cmp-1.jpg                 2016-p2.jpg
2016-cmp-3.jpg                 2016-p3.jpg
2016-cmp-4.jpg
```

12 Hello image!

- Let's load an image and show it

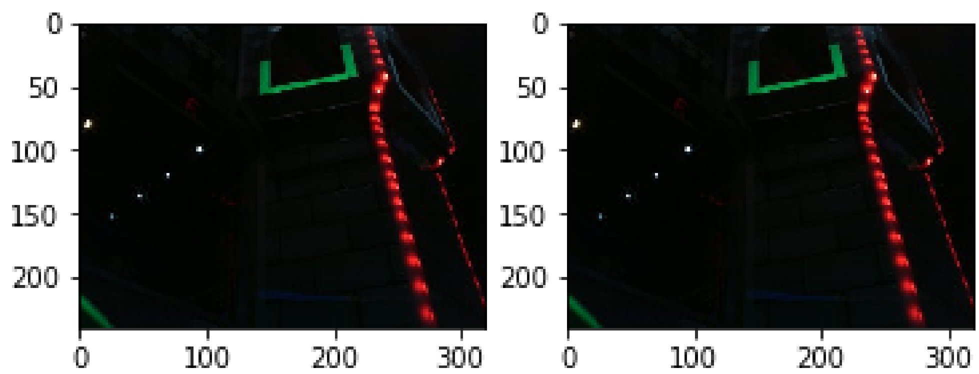
```
In [26]: # Change this to load different images
         img = cv2.imread('images/2016-cmp-5.jpg')
         imshow(img)
```



13 Hello image!

- You can show multiple images next to each other

```
In [5]: imshow(img, img)
```



14 OpenCV Image Basics

- Images are stored as multidimensional arrays

- Color images have 3 dimensions: height, width, channel

- Each pixel is a number stored in the array
- Numpy array notation allows you to do operations on individual pixels or ranges of pixels

```
In [6]: img[50, 150, :]          # Access a single pixel,
```

```
Out[6]: array([ 4, 21,  0], dtype=uint8)
```

```
In [7]: x = img[24:42, 42:100, :]    # Access a range of pixels
```

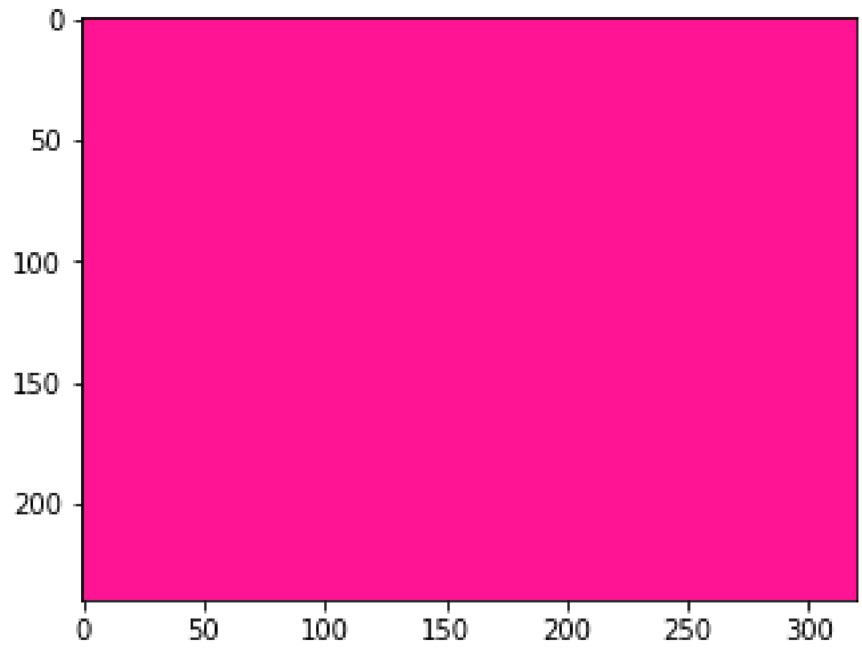
15 OpenCV Image Basics

- Color is represented by storing combinations of Red, Blue, and Green pixels in separate channels
  - OpenCV uses BGR representation, not RGB
- The amount of each individual color is represented in the individual channel
  - 'dark' is zero, 'bright' is 255
- Combine the channels to represent a color
  - Green = RGB(0, 255, 0)
  - Deep Pink = RGB(255, 20, 147)

16 OpenCV Image Basics

- Using numpy we can easily fill an image with a single color

```
In [8]: # define image with height=240, width=320, 3 channels
        shape = (240, 320, 3)
        pink_img = np.empty(shape, dtype=np.uint8)

        # Fill every pixel with a single color
        pink_img[:] = (147, 20, 255)

        imshow(pink_img)
```



17 Practical Example

- 2016 FIRST Stronghold: find targets that are surrounded by retroreflective tape, and shoot boulders into them
- 2017 FIRST Steamworks: use two targets to align to gear holder on the ship

18 Practical Example

- Finding gray tape at a distance isn't particularly easy
  - Key part of image processing is removing as much non-essential information from image
- We can do better!

19 Retroreflective Tape

- It has a useful property -- it reflects light directly back at the source
- What can we do with this property?
- Shine bright LEDs at the target and the tape reflects that color back to the camera
  - Many teams have found that green light works best
- Reduce exposure of camera so only bright light sources are seen

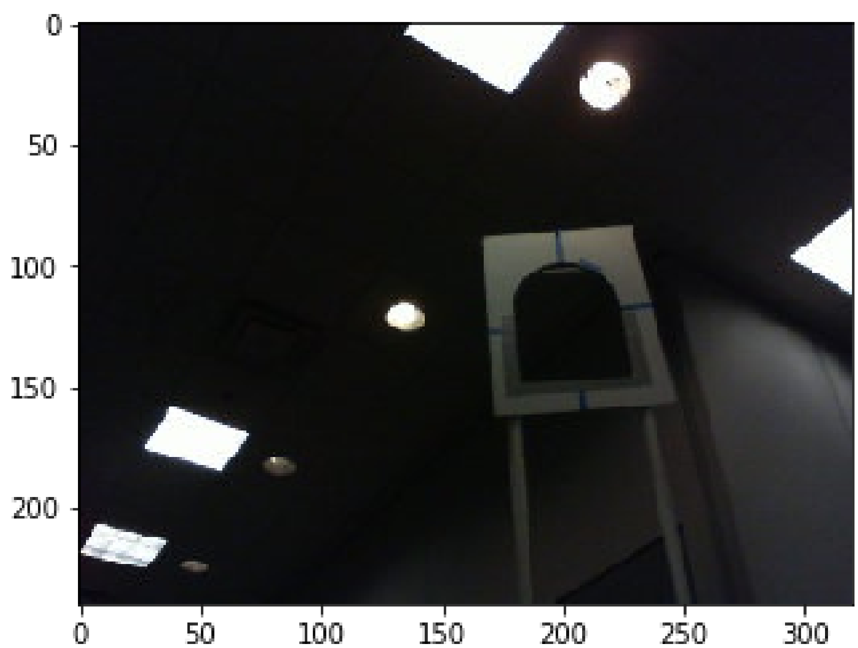
20 A note about exposure

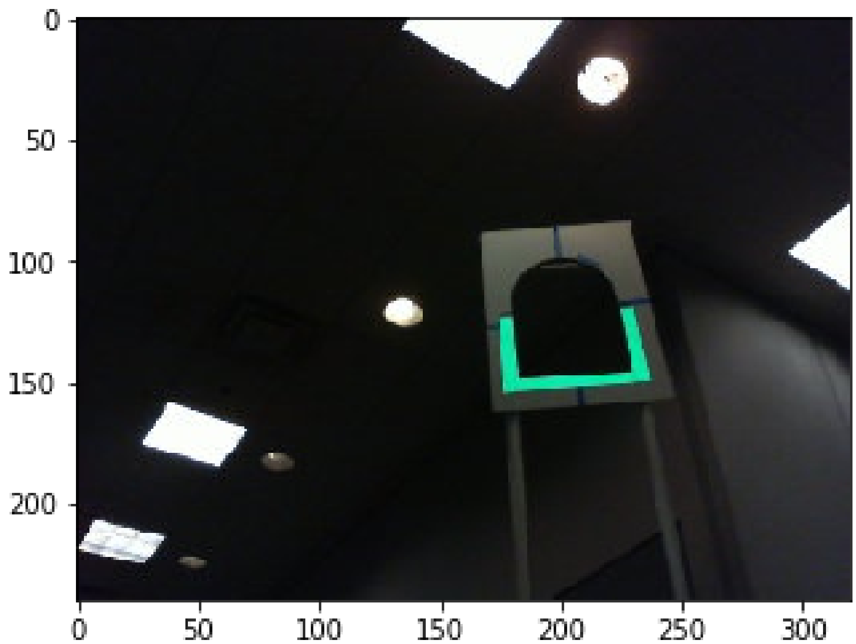
- Webcams support setting the exposure manually (yay)
- Some cameras only allow particular exposure settings
  - The lifecam is one of them
- OpenCV has bugs, it doesn't set the exposure properly
- Here's a workaround that works on linux:

```
v4l2-ctl -d /dev/video0 -c exposure_auto=1 -c exposure_absolute=10
```

21 Retroreflective Tape

```
In [9]: img1 = cv2.imread('images/2016-p0.jpg')
        img2 = cv2.imread('images/2016-p1.jpg')
        imshow(img1)
        imshow(img2)
```





22 Practical Example

Processing steps to find targets:

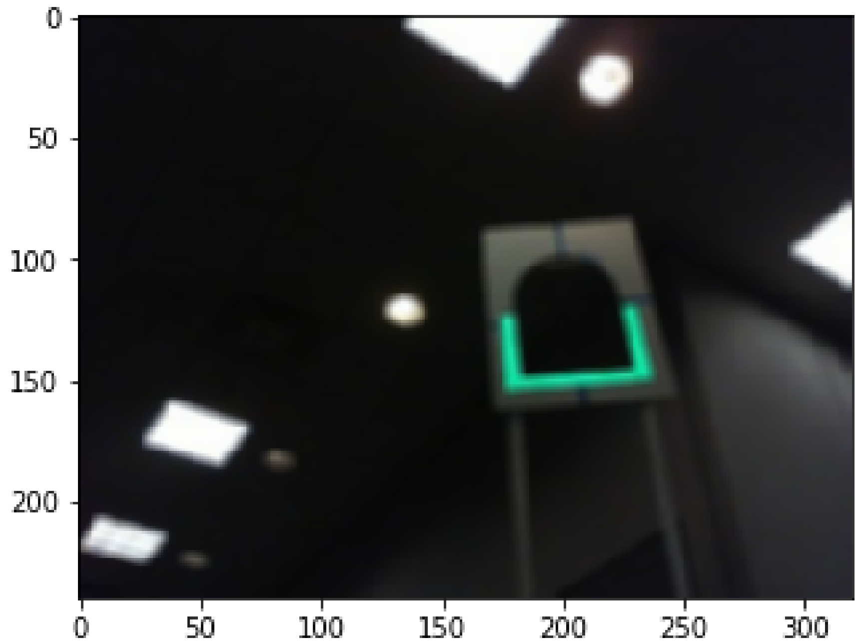
- Isolate the green portions of the image
- Analyze the green portions to determine targets

Note: There are a lot of ways to go about this, I'm just showing you one way

23 Blurring the Image

- Get rid of small artifacts (noise)
- Makes the target a little more complete
  - Easier for thresholding and contour finding

```
In [10]: blurred = blur(img2, 2)
         imshow(blurred)
```



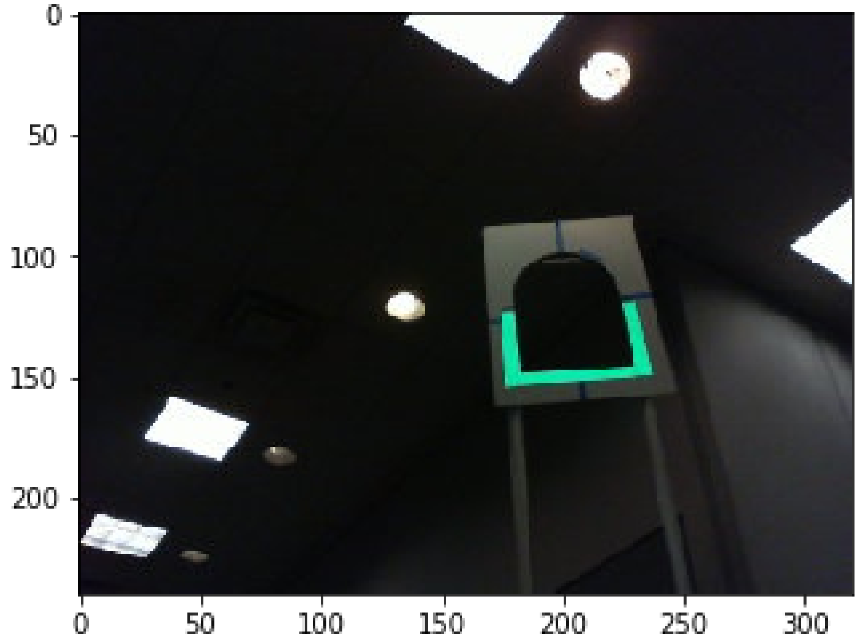
24 Identify the green

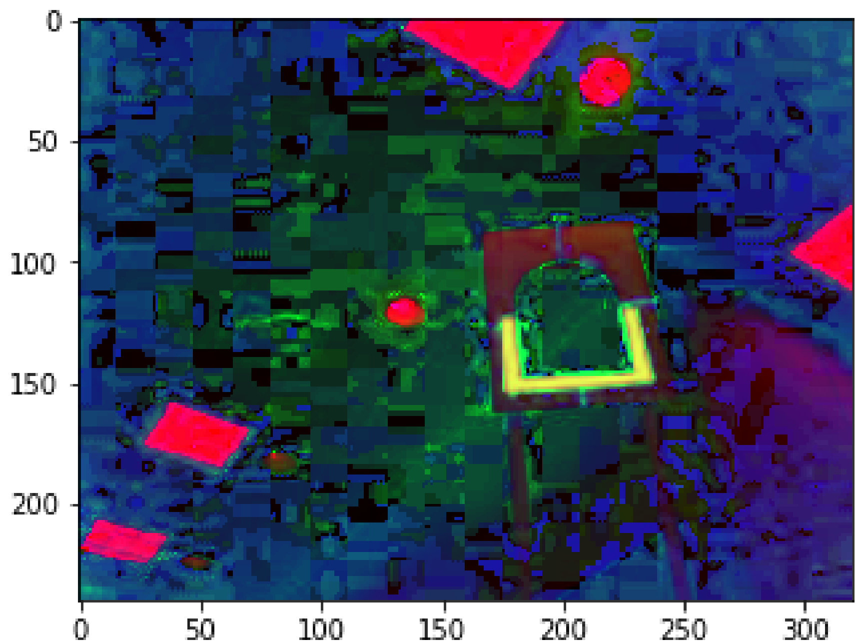
- What is "green" anyways?
    - This is green. This is also green.
- To a computer, green is really a range of colors
- An object's color changes depending on lighting conditions
- We can transform the image to identify colors independent of lighting conditions

25 Identify the green

- Convert the image from RGB to HSV
    - Hue: the color
    - Saturation: Colorfulness
    - Value: Brightness

```
In [11]: hsv = cv2.cvtColor(img2, cv2.COLOR_BGR2HSV)
         imshow(img2)
         imshow(hsv)
```

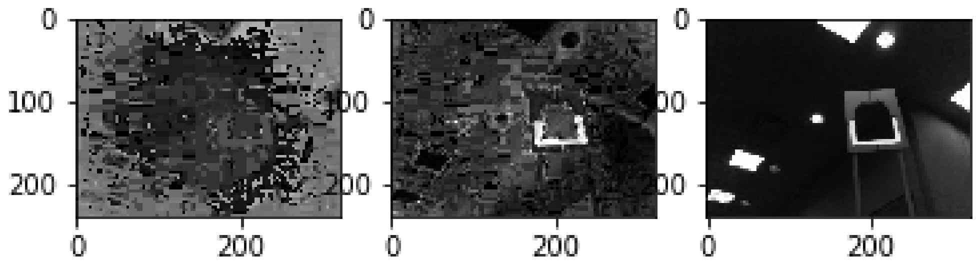




26 Identify the green

That doesn't show why HSV is useful. Let's look at the individual channels instead.

```
In [12]: h, s, v = cv2.split(hsv)
         imshow(h, s, v)
```



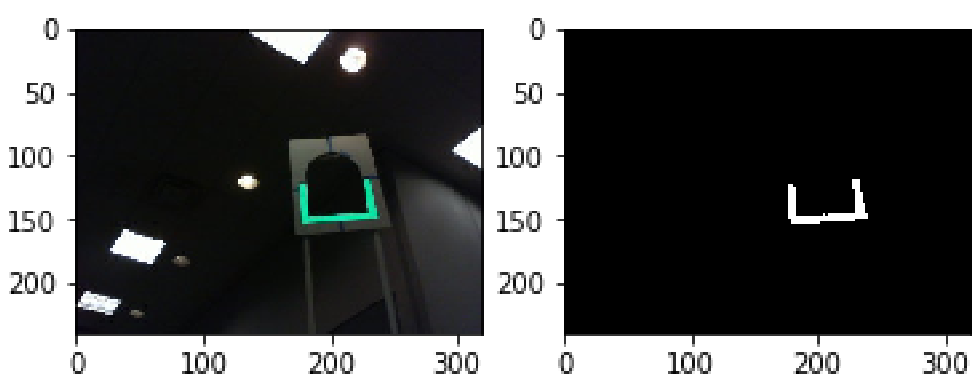
27 Identify the green

- Green is a range of values present in the image
- 'Threshold' the image to get rid of the colors that we don't care about
- Lots of ways to do this
  - Manually specify values
  - Automated methods

28 Identify the green

`cv2.inRange` can threshold an image given two ranges of pixels. * Wanted values are converted to 255 * Unwanted values are now 0

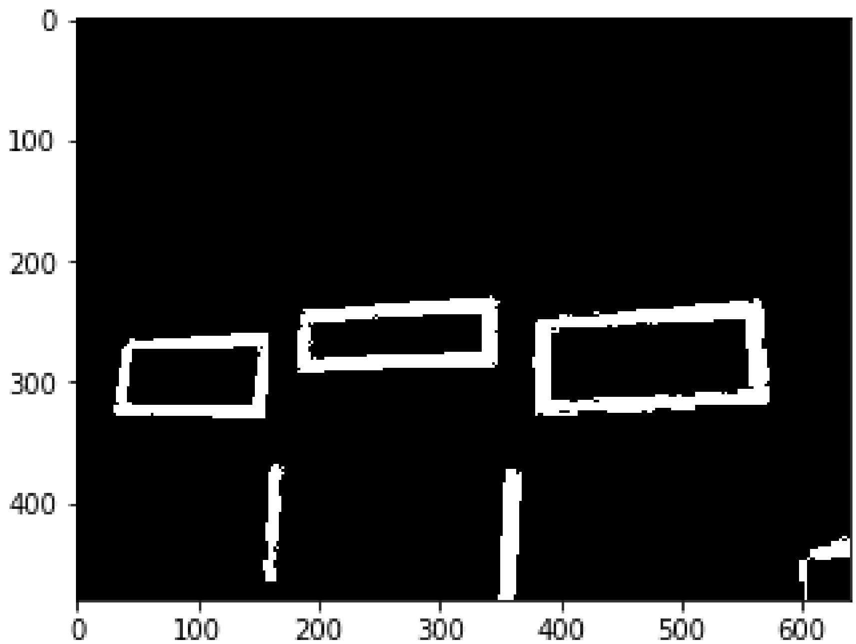
```
In [13]: lower = np.array([0, 145, 80])
         upper = np.array([255, 255, 255])

         filtered = cv2.inRange(hsv, lower, upper)
         imshow(img2, filtered)
```



29 Identify the green

Sometimes, you end up with holes in your output

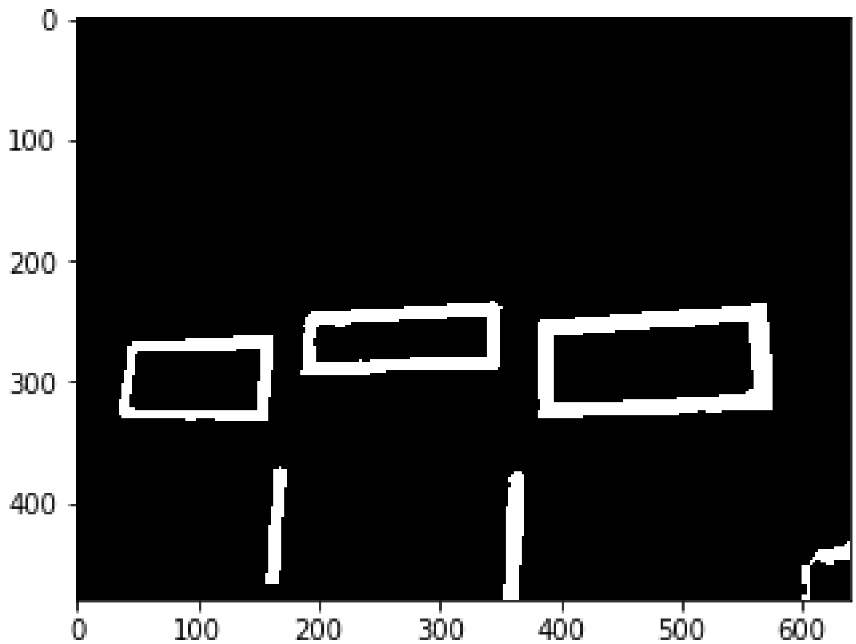
In [14]:
```
img3 = cv2.imread('images/2013-f0.png')
hsv3 = cv2.cvtColor(img3, cv2.COLOR_BGR2HSV)

# Thresholds are different because different camera/lighting
lower3 = np.array([30, 188, 16])
upper3 = np.array([75, 255, 255])
filtered3 = cv2.inRange(hsv3, lower3, upper3)
imshow(filtered3)
```



30 Identify the green

- We can use a morphological operation to fill in the holes
  - Various types of morphology operations available
- They modify a pixel based on the values of its neighboring pixels
  - The one we use to fill in holes is a "closing" operation

In [15]:
```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2,2), anchor=(1,1))
output = cv2.morphologyEx(filtered3, cv2.MORPH_CLOSE, kernel,
                          iterations=3)
imshow(output)
```



31 Identifying Targets

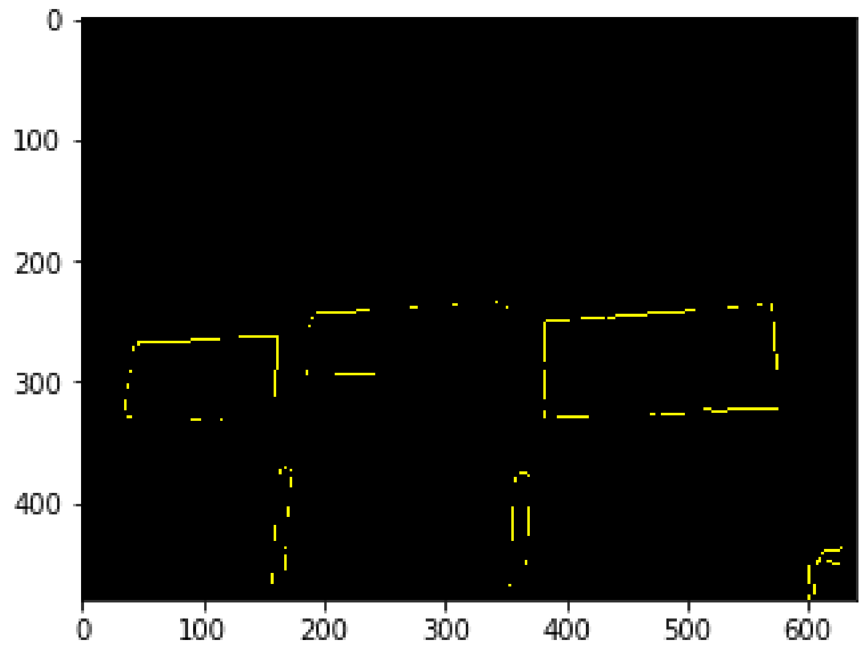
Use `findContours()` to find regions of interest * Returns a list of points bounding each separate blob in the image (called a contour) * Also returns a hierarchy so you can determine whether a contour is entirely inside another contour

```
In [16]: image, contours, hierarchy = cv2.findContours(output,
                                                        cv2.RETR_EXTERNAL,
                                                        cv2.CHAIN_APPROX_SIMPLE)
```

32 Identifying Targets

If you want to see what it found, you can draw the found contours.

```
In [17]: dst = np.zeros(shape=img3.shape, dtype=img3.dtype)
         cv2.drawContours(dst, contours, -1, (0, 255, 255), 1)
         imshow(dst)
         #print(contours[0])
```



33 Identifying Targets

- As you can see, contours aren't the whole story

34 Identifying Targets

- Contour analysis
  - Discard non-convex contours
  - Convert to polygon approximation (approxPolyDP)
  - Discard polygons that aren't rectangles
  - Discard polygons that aren't the right size

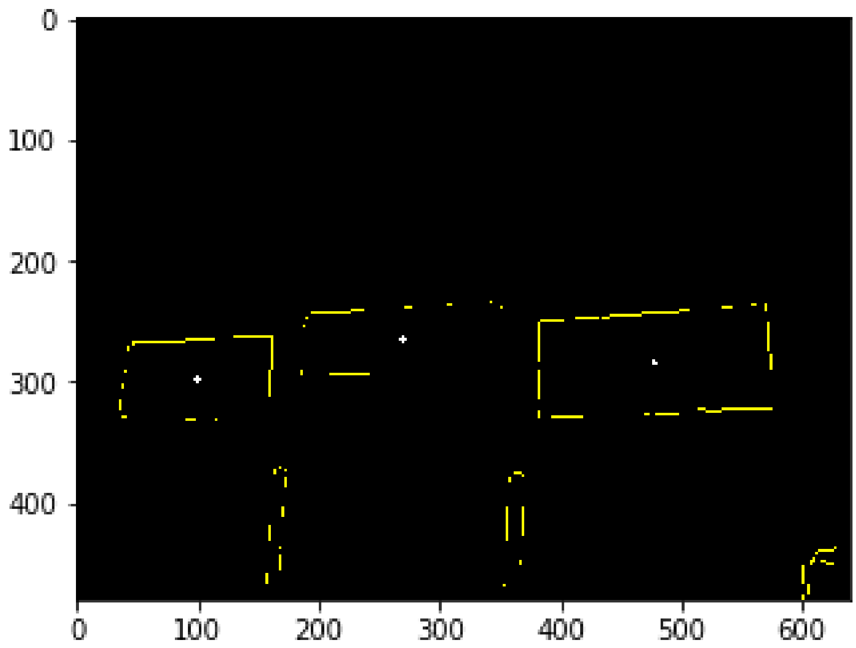
35 Magic?

```
In [18]: min_width = 20  # in pixels
         results = []
         centers = []

         # Iterate over each contour
         for c in contours:

             # Contours are jagged lines -- smooth it out using an approximation
             a1 = cv2.approxPolyDP(c, 0.01 * cv2.arcLength(c, True), True)
```

```
        # This fills in the contour so that it's a rectangle
        hull = cv2.convexHull(c)

        # Approximate the points again, smoothing out the hull
        a2 = cv2.approxPolyDP(hull, 0.01 * cv2.arcLength(hull, True), True)

        # We only care about objects that are wider than they are tall, and things wider
        # than a particular width. Only keep things that meet that criteria.
        _, _, w, h = cv2.boundingRect(a2)
        if w < h or w < min_width or len(a2) not in (4, 5):
            continue

        results.append(a2)
        M = cv2.moments(c)
        if M["m00"] == 0:
            continue
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])

        # draw the contour and center of the shape on the image
        centers.append((cX, cY))
        #cv2.putText(image, "center", (cX - 20, cY - 20),
        #            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
```

36 Magic?

```
In [19]: # Finally, draw out our results
         for cnt in centers:
             cv2.circle(dst, (cnt[0], cnt[1]), 1, (255, 255, 255), 2)
         imshow(dst)
         #print(results[0])
```



37 Identifying Targets

- Sometimes you need to do more work
  - Use ratios to determine which target you're looking at
  - Remove duplicates (inner rectangles)
  - Other types of validation

38 Now what?

We have targets... probably should do something with them?

39 Calculate angle/distance to target

- I'm not a math guy, but this sorta works
  - Angle works, distance is a bit iffy
- Get the minimum bounding rectangle
- Figure out the horizontal and vertical field of view for your camera
  - Look it up online
- Do math to it

```
In [20]: # Just do the first one for now
         result = results[0]

         # Get the height/width
         h = float(img.shape[0])
         w = float(img.shape[1])

         # Define HFOV and VFOV
         VFOV = 45.6 # degrees
         HFOV = 61.0 # degrees
```

```
In [21]: ((cx, cy), (rw, rh), rotation) = cv2.minAreaRect(result)

         # These work fairly well
         angle = VFOV * cy / h - (VFOV/2.0)
         height = HFOV * cx / w - (HFOV/2.0)

         print(angle, height)

(33.24999420166016, -11.7234375)
```

```
In [22]: # This is magic, but it doesn't really work
         target_height = 7.66 # 7' 8"
         camera_height = 1.08 # 13"
         camera_pitch = 40.0  # What angle is the camera at?
         t = (target_height - camera_height)
         distance = t/math.tan(math.radians(-angle + camera_pitch))

         print(distance)

55.5940910952
```

40 Now What?

- Send data via NetworkTables
- ... I forgot to write this slide. It's easy, I promise.

41 Where to run the image processing

- RoboRIO
  - RoboRIO is relatively slow, OpenCV eats a lot of CPU
    - Hint: Make the images small (320x240)
  - Less hardware to deal with
  - FIRST intends to install OpenCV by default in 2017

42 Where to run the image processing

- Driver Station
  - Streaming images to OpenCV is possible
    - Various latency bugs
  - Latency is an issue here
  - mDNS problems (hopefully will be resolved in 2017)

43 Where to run the image processing

- Coprocessor (Jetson, Raspberry PI, Nexus 5)
  - Lots of teams do this
  - More hardware to deal with
  - Potentially higher fidelity processing

44 Want code?

- Working OpenCV code integrated with mjpg-streamer
  - https://github.com/frc2423/2016/tree/master/OpenCV
  - Includes code for storing images onto USB drive during matches
  - Don't let our robot's performance fool you... :(
- The stuff we did here will be available sometime tonight
  - https://github.com/virtuald/frc-imageprocessing-workshop-2016

45 If you want more

- Team 254 gave an excellent presentation at CMP in 2016
  - https://goo.gl/mppi4E
  - Video/audio: http://www.chiefdelphi.com/forums/showthread.php?t=147568&page=3
  - Latency compensation is an excellent technique presented here

46 Resources

- Python 3.5.x
  - https://www.python.org/downloads/
- Learn Python
  - http://www.codecademy.com/tracks/python
- OpenCV 3.1.0
  - http://opencv.org

- NumPy
  - Official site: http://www.numpy.org

47 Resources

- roborio-packages
  - https://github.com/robotpy/roborio-packages
- OpenCV for RoboRIO
  - https://github.com/robotpy/roborio-opencv
- mjpg-streamer for RoboRIO
  - https://github.com/robotpy/mjpg-streamer

48 Resources

- pynetworktables
  - source code + examples @ https://github.com/robotpy/pynetworktables
- Edit & debug python code using Eclipse
  - Pydev: http://pydev.org/

49 One more thing...

FIRSTwiki: https://firstwiki.github.io

- Publicly editable repository of information related to FIRST Robotics
  - Technical topics
  - Non-technical
  - Team pages
- Add content to your team's page!

#
Questions?