In [2]:
```
%matplotlib inline
from __future__ import print_function
# import ganymede
# ganymede.configure('uav.beaver.works')
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
from matplotlib.patches import Rectangle
```

In [3]:
```
#ganymede.name('YOUR NAME HERE')
def check(p): pass
check(0)
```

In [4]:
```
stopsigns = glob.glob("stopsigns/*.jpg")
stopsigns = [cv2.imread(s, cv2.IMREAD_COLOR) for s in stopsigns]
print( "{} stop signs in dataset".format(len(stopsigns)))
```

```
22 stop signs in dataset
```

Question

We have opened these images using OpenCV. What color format are these images in? BGR or RGB?

https://docs.opencv.org/3.4.1/d3/df2/tutorial_py_basic_ops.html (https://docs.opencv.org/3.4.1/d3/df2/tutorial_py_basic_ops.html)

In [5]:
```
plt.imshow(stopsigns[0]);
```

In [6]:
```
s = stopsigns[0]
b, g, r = cv2.split(s)

fig, ax = plt.subplots(ncols=3, nrows=2)
[a.axis('off') for a in ax.flatten()]
ax[0,0].imshow(b, cmap='Blues')
ax[0,1].imshow(g, cmap='Greens')
ax[0,2].imshow(r, cmap='Reds')
ax[1,0].imshow(b, cmap='gray')
ax[1,1].imshow(g, cmap='gray')
ax[1,2].imshow(r, cmap='gray')
fig.set_size_inches(15,7);
```

Exercise 1

Convert the first stop sign image from BGR to RGB (`cv2.cvtColor(YOUR_IMAGE, cv2.COLOR_BGR2RGB)`), to HSV, and to LAB.

For each colorspace, visualize the three channels (you probably want to use the gray colormap to compare between the three color spaces).

Which colorspace isolates the stop sign the best?

In [7]:
```
# TODO
RGBs=cv2.cvtColor(s,cv2.COLOR_BGR2RGB)
RedGreenBlue=cv2.split(RGBs)
HSVs=cv2.cvtColor(s,cv2.COLOR_BGR2HSV)
HueSatVal=cv2.split(HSVs)
LABs=cv2.cvtColor(s,cv2.COLOR_BGR2LAB)
LightAB=cv2.split(LABs)
splits=[RedGreenBlue,HueSatVal,LightAB]
fig, ax = plt.subplots(ncols=3, nrows=3)
[a.axis('off') for a in ax.flatten()]
for i in range(3):
    for j in range(3):
        ax[i,j].imshow(splits[i][j],cmap="gray")
#All of them seem to work quite well, although HSV seems to be the best
```

A dataset of stop sign images

In [8]:
```
fig, ax = plt.subplots(nrows=len(stopsigns))
for (a,s) in zip(ax,stopsigns):
    a.imshow(cv2.cvtColor(s, cv2.COLOR_BGR2RGB))
    a.axis('off')

fig.set_size_inches(10, 5 * len(stopsigns))
```

Exercise 2

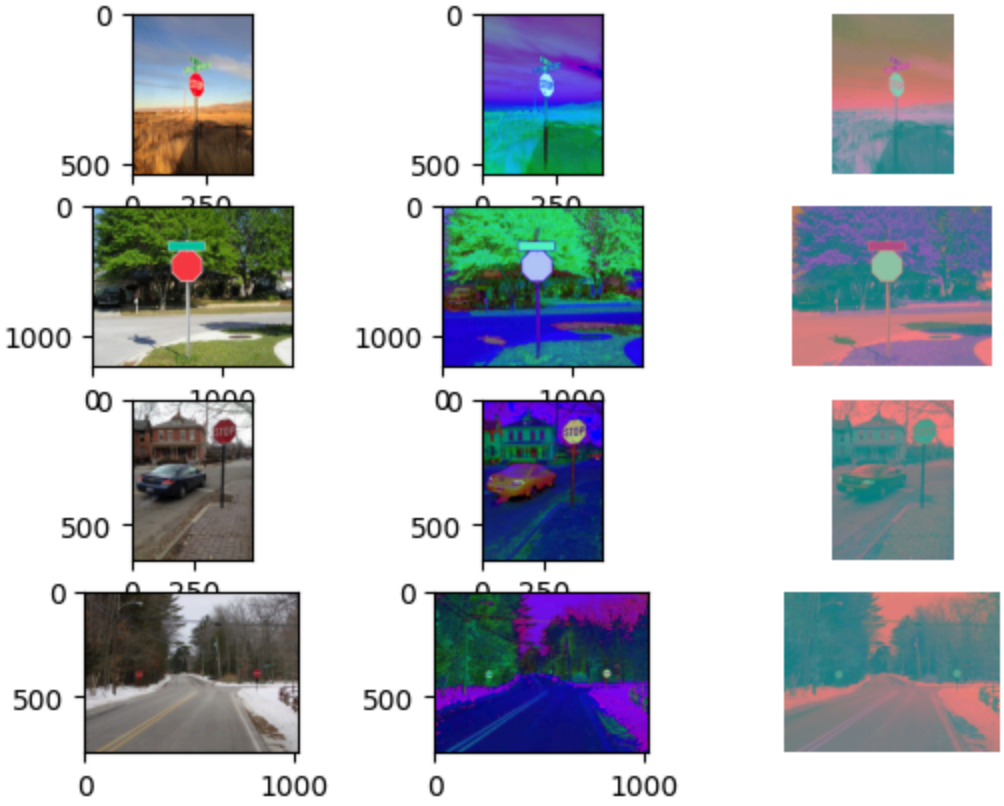
Choose 4 images from the above dataset and plot each separate color channel in RGB, HSV, and LAB

Coordinate with your teammates so that each team member chooses 4 different images. (e.g., person A chooses [0:3], person B chooses [4:7], or you can shuffle the order).

Together, your team will be processing 20 images from the above dataset. For the rest of this lab, you will be working independently on your 4 images, and you will compare your answers at the end of all the exercises.

Once you have selected your four images, revisit your answer in exercise 1. Is it the same for the other 4 images in your dataset?

In [59]:
```
## TODO
images=stopsigns[18:21]
images.append(stopsigns[15])
fig, ax = plt.subplots(ncols=3, nrows=4)
for i in range(4):
    img=images[i]
    a=ax[i, 0]
    a.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    a=ax[i, 1]
    a.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2HSV))
    a=ax[i, 2]
    a.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2LAB))
    a.axis('off')
```



Exercise 3

Filter the H (*Hue*) channel of the first image. Find the range of blue values that include as much of the sky as possible.

Use http://colorizer.org/ (http://colorizer.org/) to find the values in Hue that correspond to blue.

However, note that OpenCV does not allow Hue to extend all the way up to 360 degrees. Why?

See this answer: https://stackoverflow.com/questions/16685707/why-is-the-range-of-hue-0-180-in-opencv (https://stackoverflow.com/questions/16685707/why-is-the-range-of-hue-0-180-in-opencv)

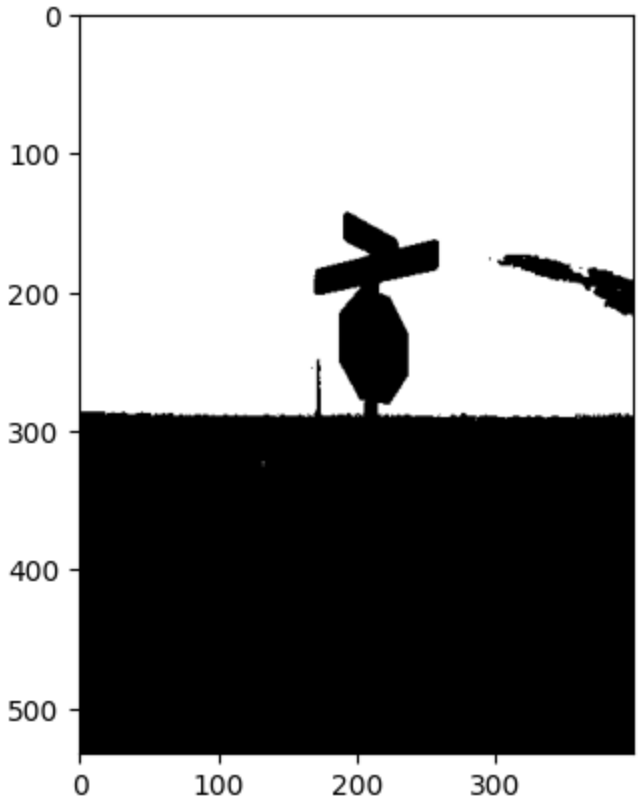
An 8-bit value can only hold values from 0 to 255.

Therefore, in OpenCV, Hue values are between 0 and 180 (the values from colorizer are divided by 2)

To filter, use the `cv2.inRange` function. See the following tutorial:

https://docs.opencv.org/3.4.1/df/d9d/tutorial_py_colorspaces.html (https://docs.opencv.org/3.4.1/df/d9d/tutorial_py_colorspaces.html)

In [54]:
```
## TODO
def get_mask(image, hsv_lower, hsv_upper):
    """
    Returns a mask containing all of the areas of image which were betwe

    Args:
        image: The image (stored in BGR) from which to create a mask.
        hsv_lower: The lower bound of HSV values to include in the mask.
        hsv_upper: The upper bound of HSV values to include in the mask.
    """
    # Convert hsv_lower and hsv_upper to numpy arrays so they can be use
    hsv_lower = np.array(hsv_lower)
    hsv_upper = np.array(hsv_upper)

    # TODO: Use the cv2.cvtColor function to switch our RGB colors to HS
    image_HSV=cv2.cvtColor(image,cv2.COLOR_RGB2HSV)
    mask=cv2.inRange(image_HSV, hsv_lower, hsv_upper)
    # TODO: Use the cv2.inRange function to highlight areas in the corre


    return mask

rgb_img=cv2.cvtColor(images[0], cv2.COLOR_BGR2RGB)
hsv_lower_sky = (80, 0, 50) #sky
hsv_upper_sky = (140, 255, 255) #sky
#stop sign: hsv_lower=(150,100,100), hsv_upper=(180,255,255)

mask_sky = get_mask(rgb_img, hsv_lower_sky, hsv_upper_sky)

plt.imshow(mask_sky,cmap="gray");
```



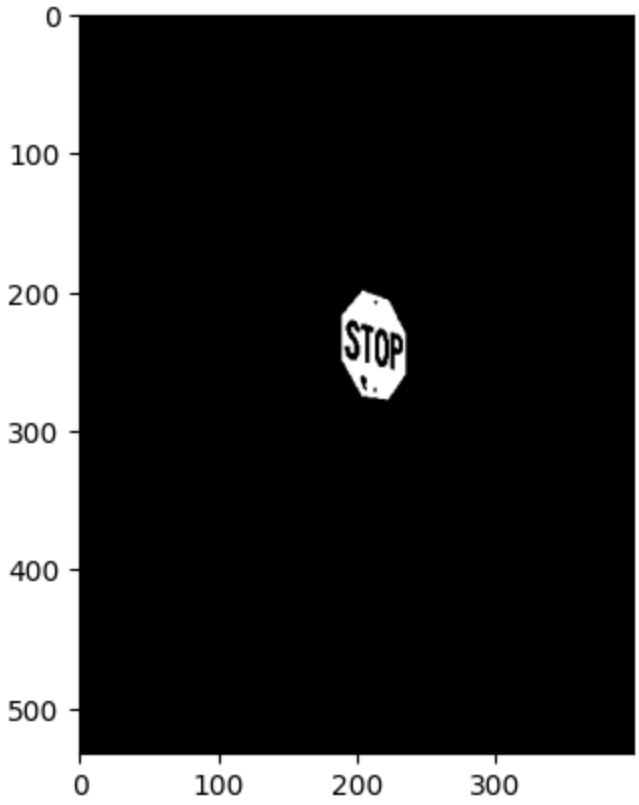
Exercise 4

Can you find a good range of hue values that mask out the stop sign in the four images you chose in Exercise 2?

Two hints:

1. Consider how polar coordinates work. Because hues are measured in angles, hue values from 0 to 5 degrees are very close to the values 355-360 (which are then divided by 2, if you are using OpenCV)
2. You might find that saturation and value also impact the hue for stop signs. Why? One example may be shadows. Can you name others?

In [55]:
```
## TODO
hsv_lower_red = (150, 100, 100) #red
hsv_upper_red = (180, 255, 255) #red
mask_red = get_mask(rgb_img, hsv_lower_red, hsv_upper_red)
plt.imshow(mask_red,cmap="gray");
```



Bounding boxes

Adding bounding box rectangles to images:

https://stackoverflow.com/a/37437395 (https://stackoverflow.com/a/37437395)

In [56]:
```
# plot an image on both the left and right
fig,ax = plt.subplots(ncols=2)
fig.set_size_inches(15,10)
ax[0].imshow(cv2.cvtColor(stopsigns[0], cv2.COLOR_BGR2RGB));
ax[1].imshow(cv2.cvtColor(stopsigns[0], cv2.COLOR_BGR2RGB));

# Define a bounding box:
bbox = Rectangle(xy=(300,90), width=200, height=180, linewidth=4, edgeco

# add the bounding box to the image on the right
ax[1].add_patch(bbox);
```

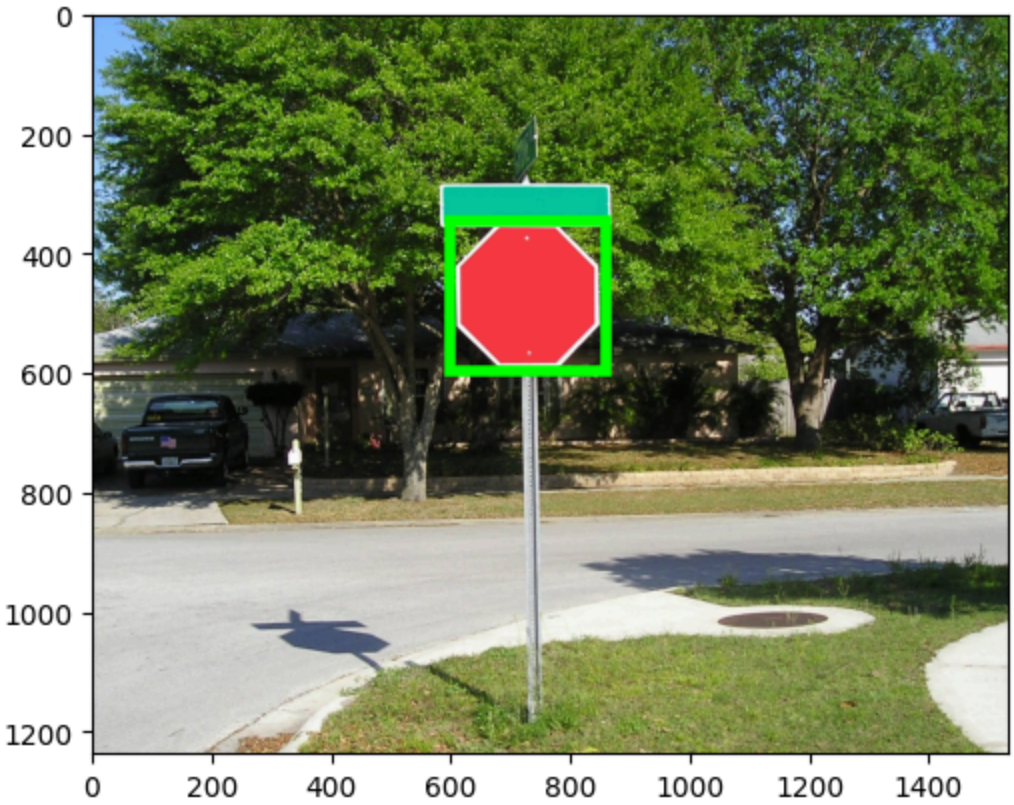
Exercise 5

Choose a different image from your dataset. Manually define a bounding box around the stop sign in that image, using `Rectangle` .

What direction is positive X in? What direction is positive y in? Hint: think back to the Linear Regression exercise.

In [57]:
```
## TODO
fig,ax=plt.subplots()
rgbimg2=cv2.cvtColor(images[1],cv2.COLOR_BGR2RGB)
ax.imshow(rgbimg2)
bbox = Rectangle(xy=(600,345), width=260, height=250, linewidth=4, edgec
ax.add_patch(bbox)
```

Out[57]:
```
<matplotlib.patches.Rectangle at 0x12bac3a60>
```



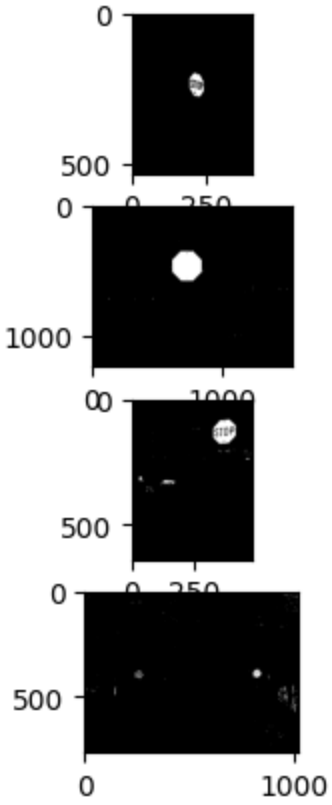
Exercise 6: Combining masking and morphology

For the four images you selected in Exercise 2, choose a colorspace and channel that you believe will work best for isolating stop signs.

1. Mask the five images using a narrow range of values in that channel.
2. Run erosion and dilation on the mask to reduce the amount of noise. What kernel sizes work best? If you want to run erosion immediately followed by dilation for the same kernel size, you can use `opening` :

https://docs.opencv.org/3.4.1/d9/d61/tutorial_py_morphological_ops.html (https://docs.opencv.org/3.4.1/d9/d61/tutorial_py_morphological_ops.html)

In [60]:
```
## TODO
hsv_lower_red = (150, 50, 50) #red
hsv_upper_red = (200, 255, 255) #red
fig, ax = plt.subplots(ncols=1, nrows=4)
for a,i in zip(ax,images):
    a.imshow(get_mask(cv2.cvtColor(i,cv2.COLOR_BGR2RGB),hsv_lower_red,hs
```



Exercise 7 - Compare with your team mates

Some images will have a lot of "noise" - in other words, for any choice of color channel, and any two values for `inRange` which mask stop sign pixels, a lot of pixels which don't come from the stop sign will also be included by the filter. What causes this? Which images are most prone to noise? What's significantly different about those images compared to the others?

In [ ]:
```
## TODO
# team solution
```

When you are done:

1. Double-check that you filled in your name at the top of the notebook!
2. Click `File -> Export Notebook As -> PDF`
3. Email the PDF to `YOURTEAMNAME@beaver.works`

Stretch Goal: Combining Linear Regression and Color Segmentation

A lot of the above images have solid white lines or solid yellow lines, due to traffic lane markings. A lot of the images also have solid black lines (telephone poles), and solid green lines (the pole of the stop sign).

1. Choose some images that contain lines (at least 5).
2. For each image, find a good color threshold which can mask the line or lines out.
3. Add in erosion, dilation, opening, or other morphological operations to improve the quality of the mask.
4. For each image, use OpenCV's linear regression function `cv2.fitLine` to plot the line of best fit from your results in the previous step.

See the following tutorial for `cv2.fitLine` :
https://docs.opencv.org/3.4.1/dd/d49/tutorial_py_contour_features.html (https://docs.opencv.org/3.4.1/dd/d49/tutorial_py_contour_features.html)