In [1]:
```
from __future__ import print_function
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
from matplotlib.patches import Rectangle
from scipy.stats import linregress
import matplotlib.patches as patches
from matplotlib.lines import Line2D
```

# Add your name below

In [2]:
```
# Fill in your name here
# Charlie Lai
```

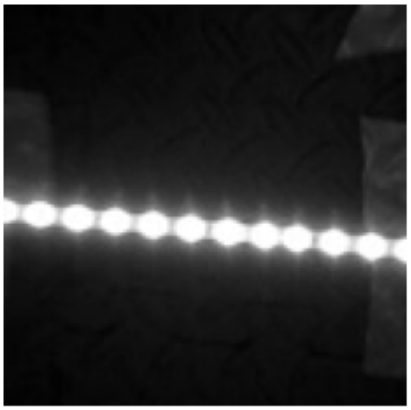
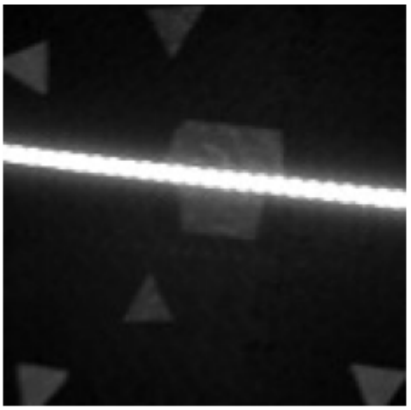
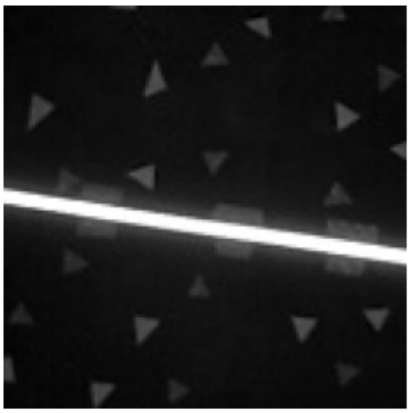
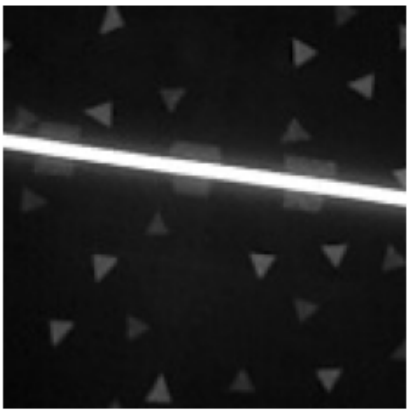
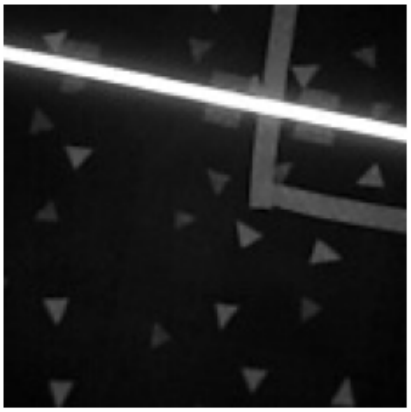
# Downward camera examples

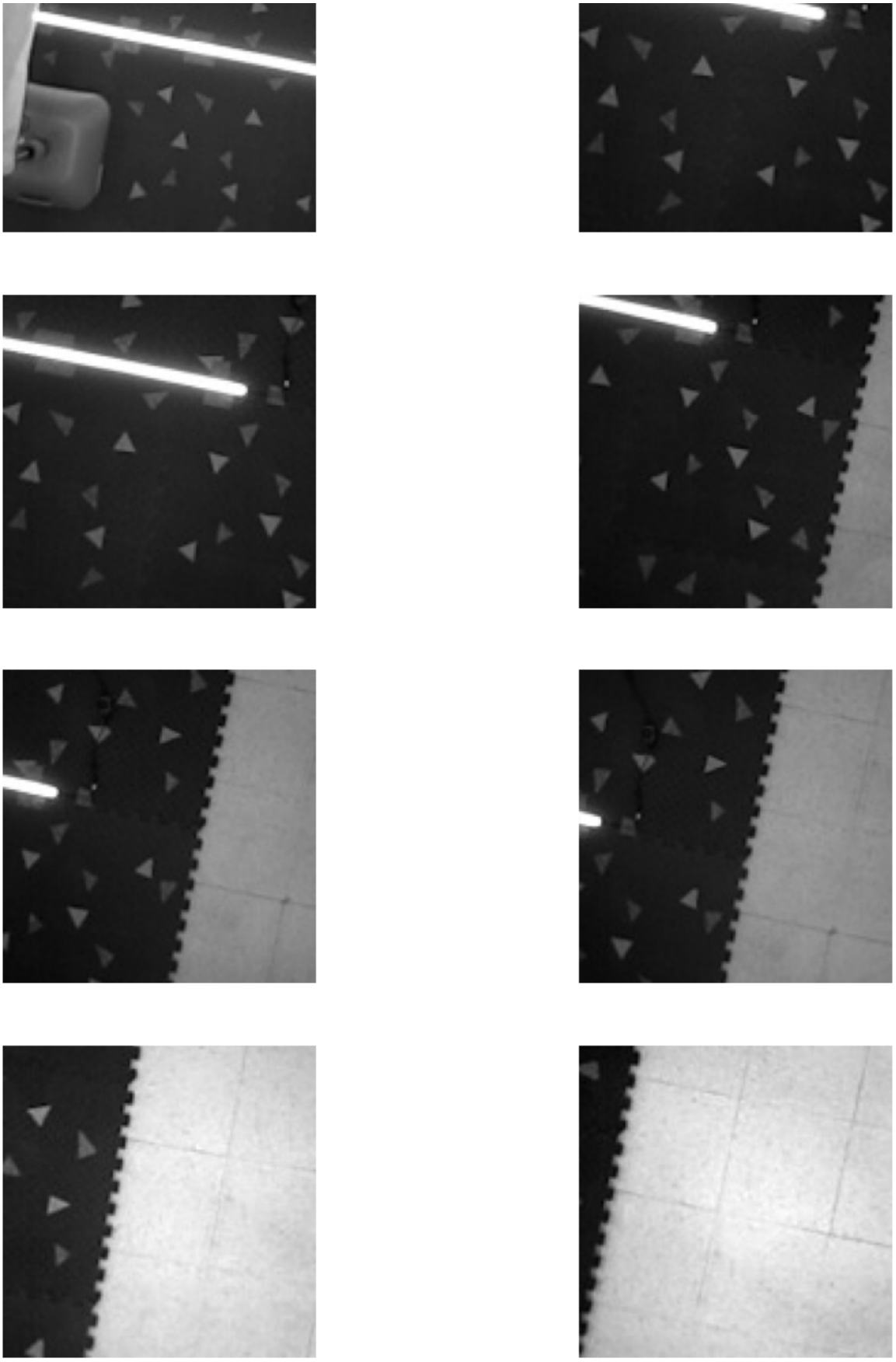
This notebook contains several images taken from the downward camera on the drone, some of which observe the LED rope, others which do not, and even a couple of examples with one of the obstacles partially blocking the view of the LED rope.

In [3]:
```
downward = glob.glob("downward/*.jpg")
downward.sort()
downward = [cv2.imread(s, cv2.IMREAD_GRAYSCALE) for s in downward]
print( "{} downward images in dataset".format(len(downward)))
```

```
16 downward images in dataset
```

In [4]:
```
fig, ax = plt.subplots(ncols=2, nrows=len(downward)//2)
[a.axis('off') for a in ax.flatten()]

for d,a in zip(downward, ax.flatten()):
    a.imshow(d, cmap='gray')

fig.set_size_inches(10,28);
```

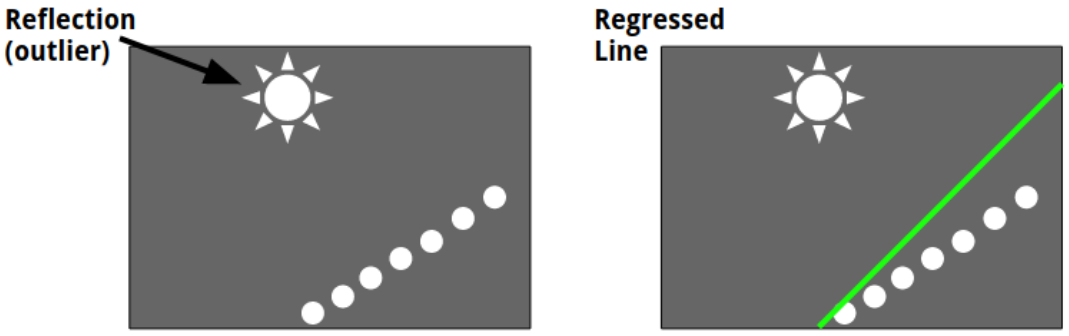


# Image Analysis: moving beyond linear regression

## Contour Analysis

You will have to implement more image processing routines beyond linear regression in order to handle several edge cases onboard the drone.

Consider a reflection (from the ceiling lights or the sun) which causes a bright spot in the image at a different position from the LED rope:




This condition represents an **outlier** - the regressed line will be "pulled" in the direction of the outlier, and not sit directly on the LED rope. One remedy for designing an outlier-robust line detector is to apply some *image morphology* plus *contour analysis*.

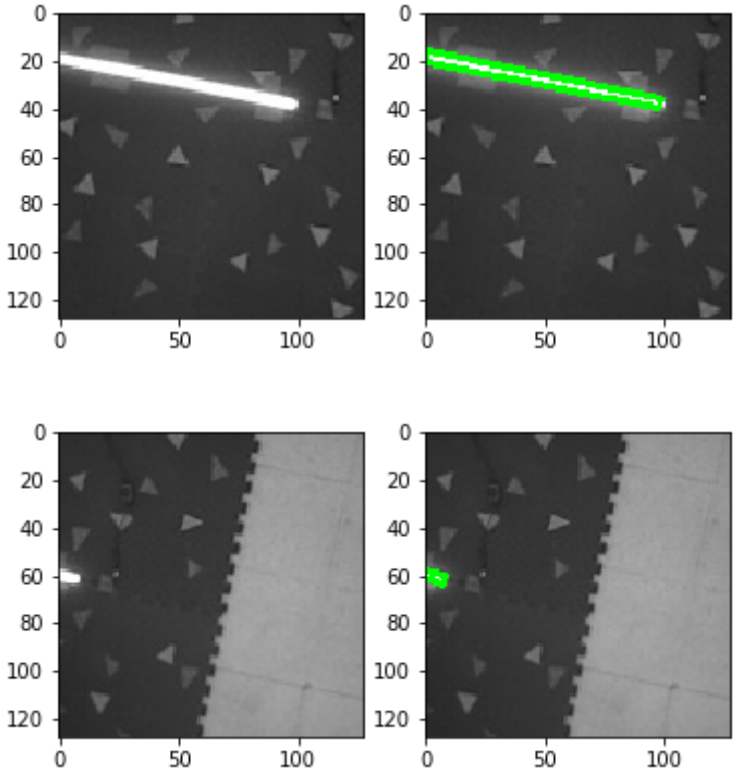
For instance, you can use **dilation** in order to connect the individual LED bulbs together into a single contour, and then regress a line through the largest contour found.

# Contour Analysis in OpenCV

The first two tutorials are very helpful to read through for completing this lab exercise (and the rest of the tutorials on contour analysis are linked below, and also worth reading for future reference in the course).

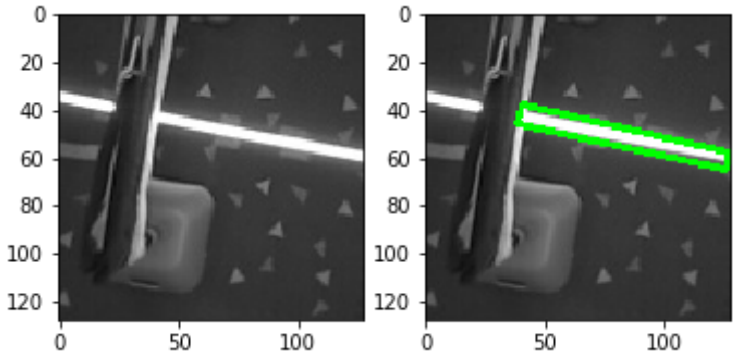
1. https://docs.opencv.org/3.3.1/d4/d73/tutorial_py_contours_begin.html
2. https://docs.opencv.org/3.3.1/dd/d49/tutorial_py_contour_features.html
3. https://docs.opencv.org/3.3.1/d3/d05/tutorial_py_table_of_contents_contours.html

Another example where contour analysis can help is deciding whether or not enough of the line is **visibile** in order to correctly compute a direction in which to fly the drone. Consider the two examples below. Arguably, not enough of the line is visible in the second example to correctly compute a flight direction. (In fact, you might instead want to ensure the line stretches across the entire image, and if not, hover the drone in place.)




Furthermore, note that applying linear regression on an individal contour is not a panacea to outliers, and indeed, it can lead to undesirable results. In the example below, there is an obstacle sitting above the LED rope. Thus, the largest contour found will not contain the entire LED rope.

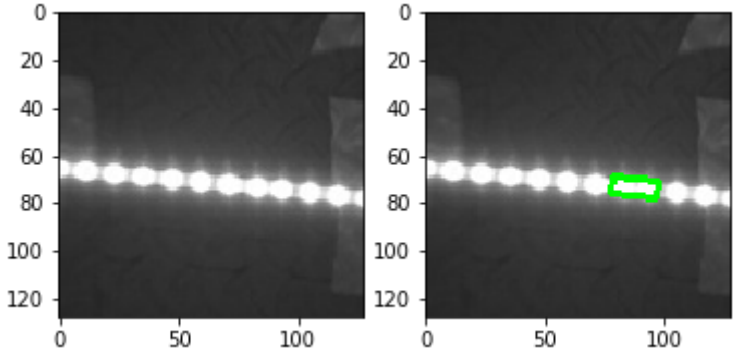
You might thus want to use contour analysis to determine that yes, enough of the LED rope is visible to compute a valid velocity command for the drone (by inspecting the width and height of the contour), but then perform linear regression on the entire (thresholded) image, rather than only on the largest contour found. Alternatively, you might want to consider multiple contours. There are many strategies available to choose from. This is part of the design exercise that your team will proceed through.




# Morphology

As a final example of "contour analysis gone wrong," observe that the largest contour found in the below image does not contain the entire LED rope - it only contains a couple of individual LED bulbs. This is a case in which **dilation, erosion, opening** or

other morphological operations can greatly benefit your image processing pipeline, e.g., by "closing the gaps" between each LED bulb.




# Your task

Process all of the images in this small dataset. For each image, write code to detect whether it contains enough of the line to justify further processing (or no line at all), and if your algorithm decides it does contain a sufficient enough portion of the LED rope, plot the regressed line on top of the image.

Using contour analysis for this task is certainly not required (there are many ways you could tackle this problem), but it's a good place to start!

Note that you may need to tune your thresholding algorithm. Several images in the dataset also contain gray floor tiles, which should *not* be detect as white LED pixels.

# Stretch goals:

You now have implemented linear regression on your own. Compare your answers on all of the above images with two "off-the-shelf" python functions (one from OpenCV, another from scipy) which implement linear regression. Visualize the outputs of both functions, alongside your own implementation from the Linear Regression notebook.

**OpenCV:**

```
cv2.fitLine()
```

**Scipy:**

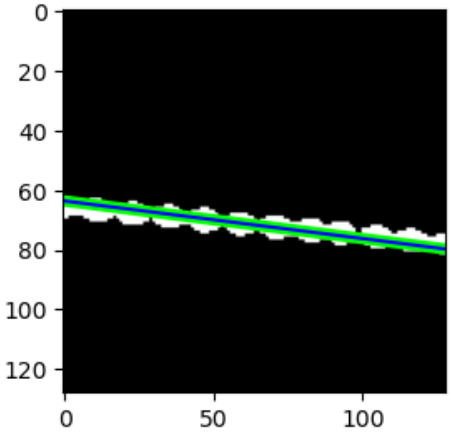
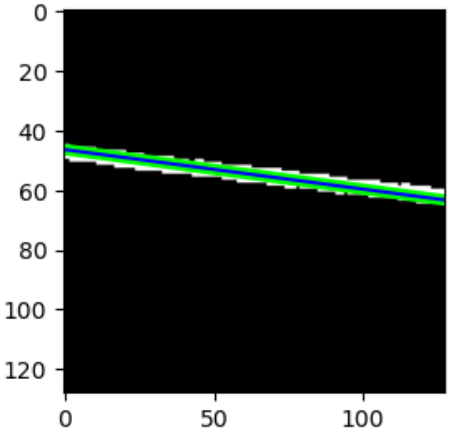
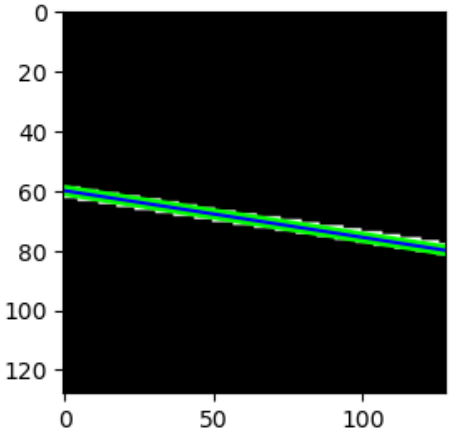
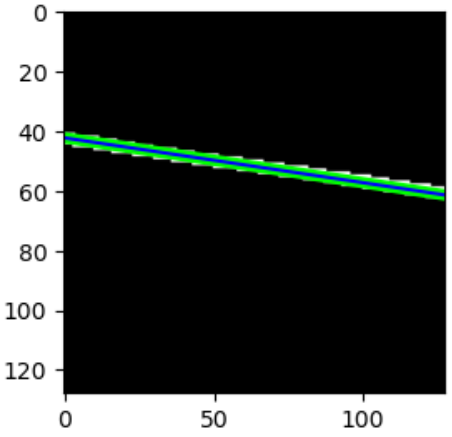
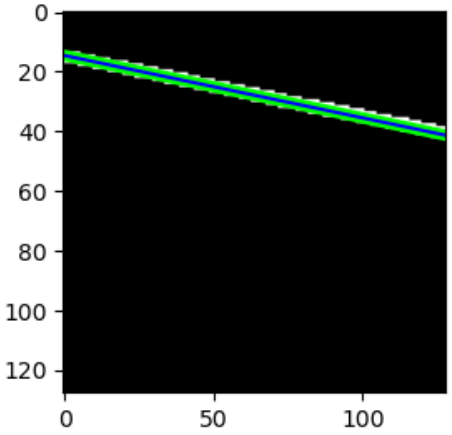
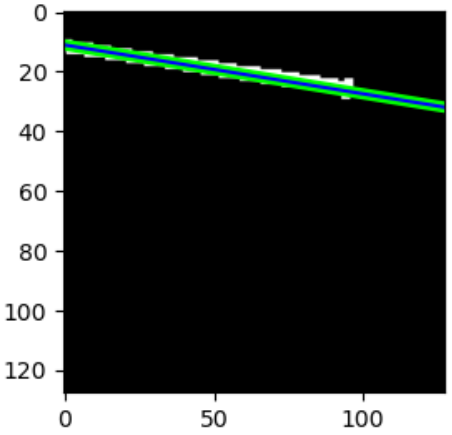
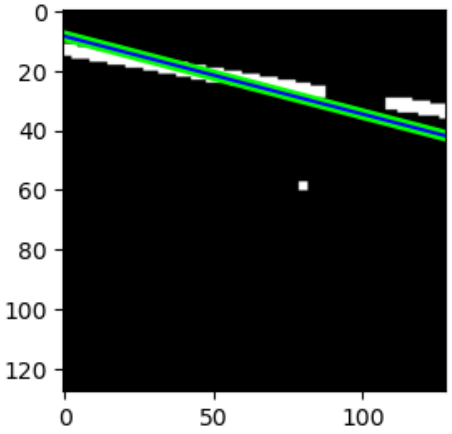
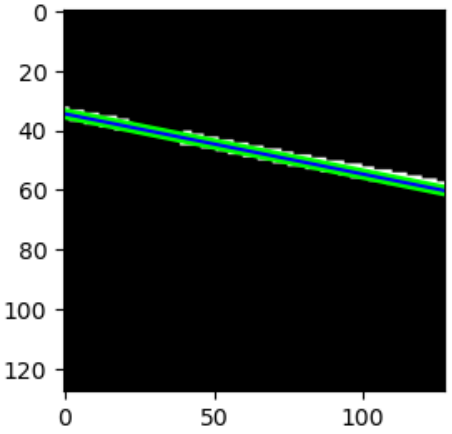
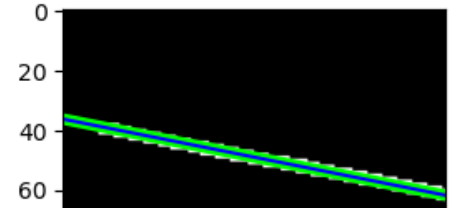
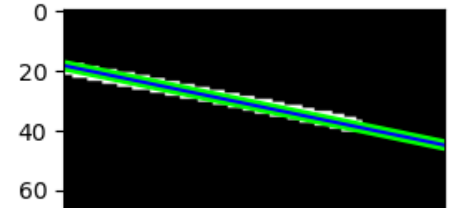
```
from scipy.stats import linregress
```

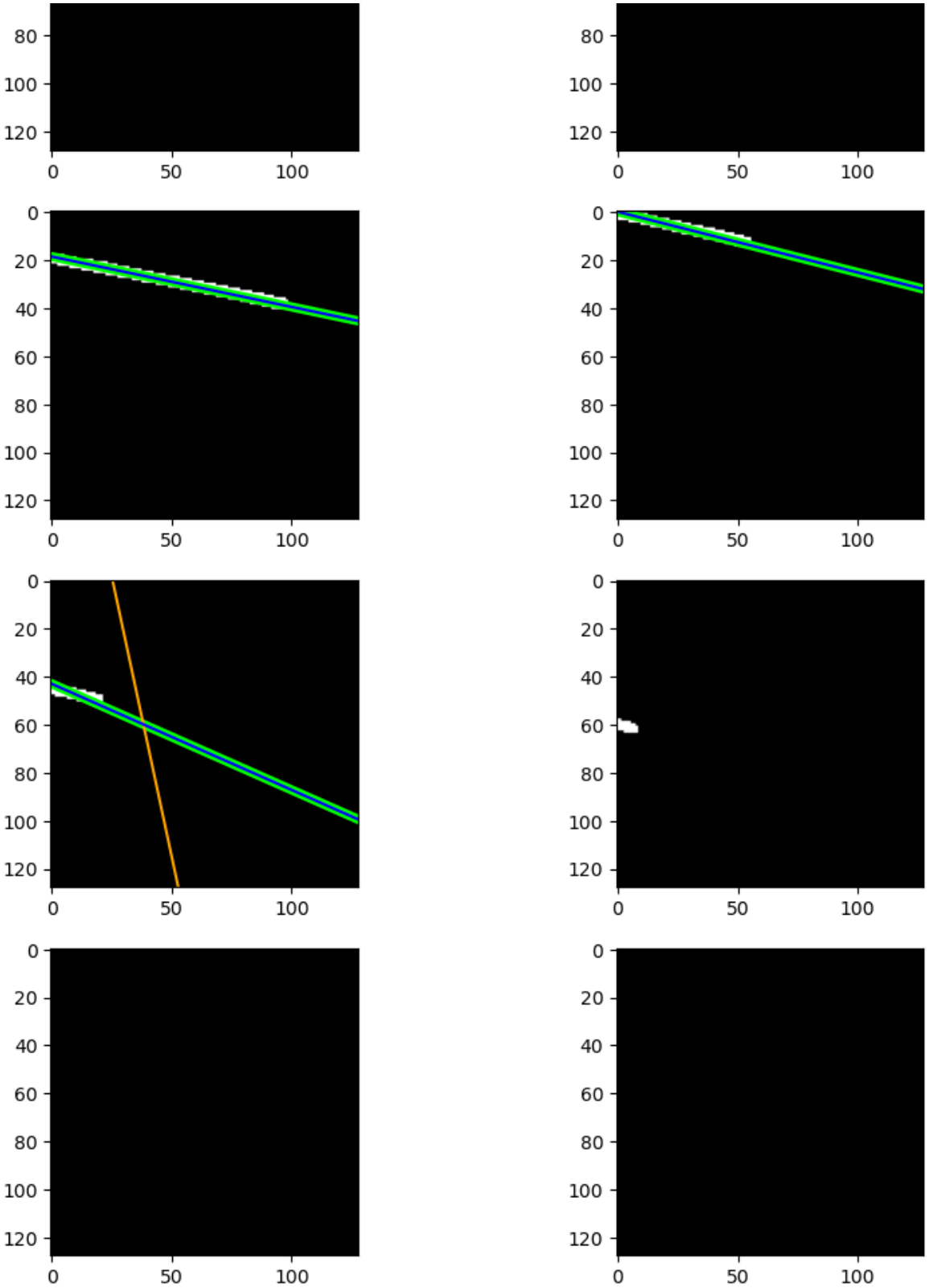
In [7]:
```
from __future__ import print_function
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
from matplotlib.patches import Rectangle
from scipy.stats import linregress
import matplotlib.patches as patches
```

```
from matplotlib.lines import Line2D

def find_contours(mask):
    return cv2.findContours(mask, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

def draw_contour(image, contour, color=(0, 255, 0)):
    cv2.drawContours(image, [contour], 0, color, 3)

def calculate_regression(points): # input is the result of np.argwhere; t
    points = points.astype(float)
    l=len(points)
    xs=[]
    ys=[]
    for i in range(l):
        xs.append(points[i][0])
        ys.append(points[i][1])
    xys=np.multiply(xs,ys)
    x_squared=np.multiply(xs,xs)
    x_mean = np.mean(xs)
    y_mean = np.mean(ys)
    xy_mean = np.mean(xys)
    x_squared_mean = np.mean(x_squared)
    m = (x_mean*y_mean-xy_mean)/(x_mean*x_mean-x_squared_mean)
    b = y_mean-m*x_mean
    return (m,b)

def find_inliers(m, b, shape):
    x1, y1, x2, y2 = max(0,-b)/m, max(0,-b)+b, min(m*shape[0],shape[1]-b)
    return [x1,y1,x2,y2]

def get_mask(image, hsv_lower, hsv_upper):
    hsv_lower = np.array(hsv_lower)
    hsv_upper = np.array(hsv_upper)
    image_HSV=cv2.cvtColor(image,cv2.COLOR_RGB2HSV)
    mask=cv2.inRange(image_HSV, hsv_lower, hsv_upper)
    return mask

def get_largest_contour(contours, min_area=30, max_area=1000000):
    most_area=min_area
    greatest_contour=None
    for contour in contours:
        if cv2.contourArea(contour)>most_area and cv2.contourArea(contour
            greatest_contour=contour
            most_area=cv2.contourArea(contour)
    return greatest_contour




fig, ax = plt.subplots(ncols=2, nrows=len(downward)//2)

for d, a in zip(downward, ax.flatten()):
    # convert image to RGB values
    drgb=cv2.cvtColor(d,cv2.COLOR_BGR2RGB)

    # set HSV bounds for mask creation
    lower=(0,0,210)
    upper=(180,200,255)

    # create mask from HSV bounds and image
```

```
    dfilter=get_mask(drgb,lower,upper)

    #filter out noise
    kernel=np.ones((3,3),dtype=int)
    dfilter = cv2.morphologyEx(dfilter, cv2.MORPH_OPEN, kernel)

    # find largest contour if it exists
    contours=find_contours(dfilter)
    largest_contour=get_largest_contour(contours)
    a.imshow(dfilter,cmap="gray")

    if largest_contour is not None:
        # finds the indices that aren't 0 (black)
        args=np.argwhere(dfilter)

        # calc my linear regression line
        m,b = calculate_regression(args)
        pts = find_inliers(m,b, d.shape)
        regression = Line2D([pts[1],pts[3]],[pts[0],pts[2]], color='lime'
        # display line
        a.add_line(regression)

        # calc OpenCV line
        cv2line=cv2.fitLine(largest_contour,distType=2,param=0,reps=1,aep
        m1=cv2line[1]/cv2line[0]
        b1=cv2line[3]-m*cv2line[2]
        pts1=find_inliers(m1,b1,d.shape)
        cvregression = Line2D([pts1[1],pts1[3]],[pts1[0],pts1[2]], color=
        # display line
        a.add_line(cvregression)
        contourx=[]
        contoury=[]

        # add all white x values to countourx and white y values to conto
        for pt in args:
            contourx.append(pt[0])
            contoury.append(pt[1])

        # calc scipy linear regression line
        scipyline=linregress(contourx,contoury)
        m2=scipyline.slope
        b2=scipyline.intercept
        pts2=find_inliers(m2,b2,d.shape)
        scipyregression = Line2D([pts2[1],pts2[3]],[pts2[0],pts2[2]], col

        # display
        a.add_line(scipyregression)

fig.set_size_inches(10,28)

"""I made my custom lime regression line extra thick because otherwise if
hidden beneath the SciPy line, so that's why the images look a little wei
```

Out[7]:
```
"I made my custom lime regression line extra thick because otherwise if
would be \nhidden beneath the SciPy line, so that's why the images look
a little weird"
```



## When you are done

You should have all sixteen images processed, some with regression lines plotted on top of them, others detected as not containing a line (or not containing enough of the line).

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`

In [ ]: