CS445 Final Project

In [41]:

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
from itertools import groupby, combinations
%matplotlib inline
```

color pixel count

green pixel out of all the pixels. sensitive to color range

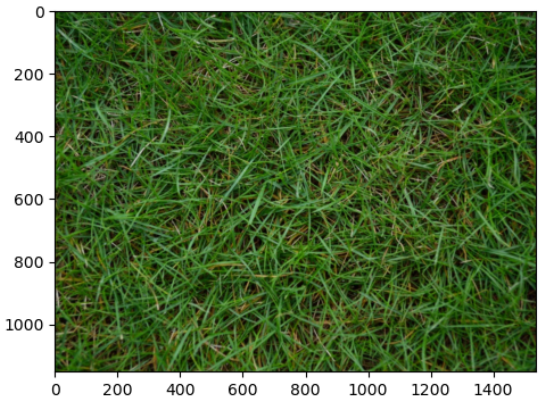
In [2]:

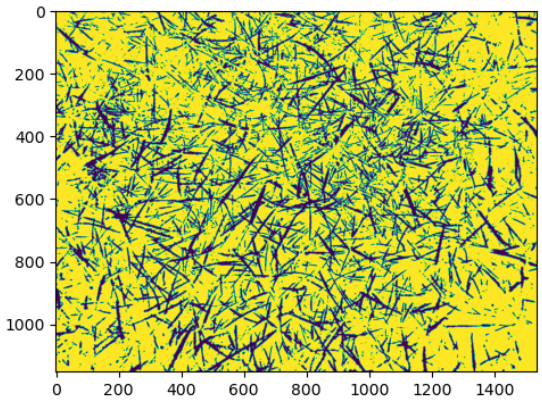
```
def count_grass_pixels(image, lower_green, upper_green):
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv_image, lower_green, upper_green)
    count = np.sum(mask > 0)
    return count, mask
```

In [83]:

```
img = cv2.imread('./images/green-bermuda-grass.jpg')
lower_green = np.array([40, 0, 0])
upper_green = np.array([120, 255, 256])

hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

count, mask = count_grass_pixels(hsv_img, lower_green, upper_green)
cv2.imwrite("mask.jpg", mask)  # save for next parts

# Calculate the percentage
percentage = count / (img.shape[0] * img.shape[1])
print(f"score: {percentage*100}%")

plt.figure(figsize=(12,6))

plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.plot()

plt.subplot(1,2,2)
plt.imshow(mask)
plt.plot();

```

```
score: 75.73281747323496%
```





lines count (HoughLinesP)

number of line segments to estimate number of blades. Lines have to be very straight. not accurate

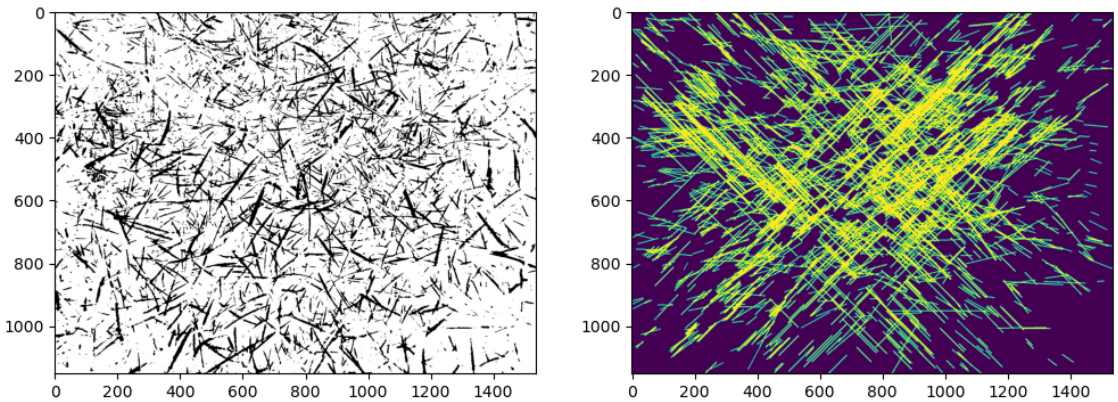
In [4]:

```python
def count_lines_houghP(img):
    # Load image
    mask = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Apply edge detection
    edges = cv2.Canny(mask, 50, 150)

    # Use Probabilistic Hough Transform to detect line segments
    lines = cv2.HoughLinesP(edges, 1, np.pi / 180, threshold=150, minL

    # Draw the line segments on the original image
    lines_img = np.zeros_like(mask)
    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            cv2.line(lines_img, (x1, y1), (x2, y2), 255, 2)

    # Count the number of lines
    num_lines = 0
    if lines is not None:
        num_lines = len(lines)

    plt.figure(figsize=(12,6))

    plt.subplot(1,2,1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.plot()

    plt.subplot(1,2,2)
    plt.imshow(lines_img)
    plt.plot();

    print(f'Number of lines: {num_lines}')

```

In [5]:

```python
img = cv2.imread('mask.jpg')
count_lines_houghP(img)

```

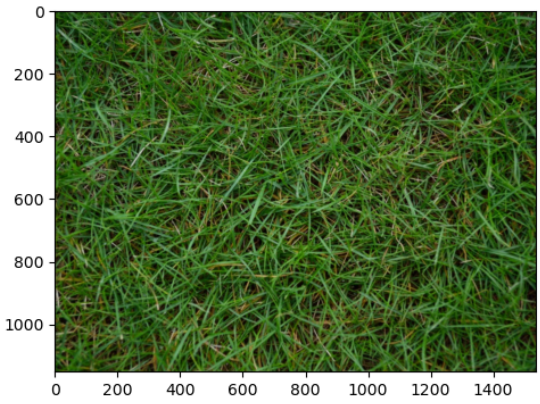
```
Number of lines: 2225
```

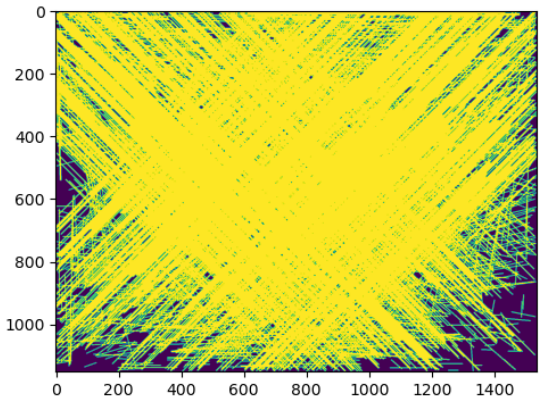


In [6]:

```
1 img = cv2.imread('./images/green-bermuda-grass.jpg')
2 count_lines_houghP(img)
3
```

```
Number of lines: 2472
```





lines count (LineSegmentDetector)

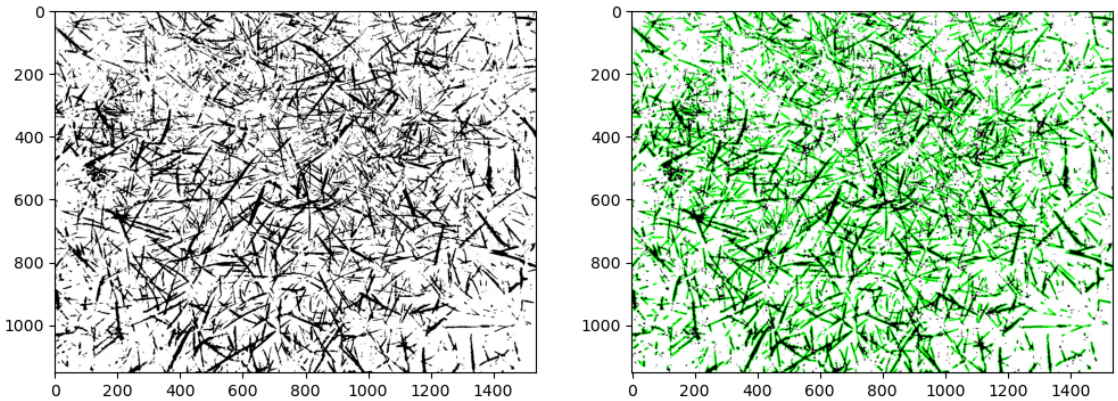
number of line segments to estimate number of blades.

In [192]:

```python
def count_lines_lsd(img, length_threshold=10):
    # Load your binary mask image
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Create LineSegmentDetector object
    lsd = cv2.createLineSegmentDetector()
    lines, width, prec, nfa = lsd.detect(gray)

    # Draw the line segments on the original image
    lines_img = np.copy(img)
    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = map(int, line[0])
            # Check if the line length is greater than length threshol
            if np.sqrt((x2 - x1)**2 + (y2 - y1)**2) > length_threshold
                cv2.line(lines_img, (x1, y1), (x2, y2), (0, 255, 0), 2

    # Count the number of lines
    num_lines = 0
    if lines is not None:
        num_lines = sum(1 for line in lines if np.sqrt((line[0][2] - l

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(lines_img, cv2.COLOR_BGR2RGB))

    plt.show()

    return round(num_lines / 25) # calibrated from a simple image

```

In [178]:

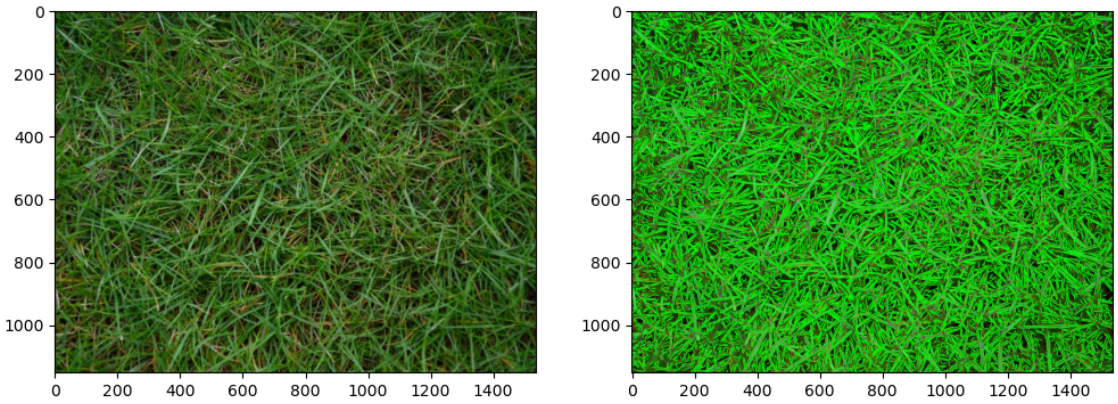
```python
img = cv2.imread('mask.jpg')
print(f'Number of lines: {count_lines_lsd(img)}')
```




```
Number of lines: 308
```

In [179]:

```
1 img = cv2.imread('./images/green-bermuda-grass.jpg')
2 print(f'Number of lines: {count_lines_lsd(img)}')
3
```




```
Number of lines: 571
```

Calibration and Area

find the area by a known measurement in the image (12 in is ~ 30.48 cm) and dpi

In [180]:

```
img = cv2.imread('./images/12in.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.plot();
m_per_pixel = 0.3048 / img.shape[0]

print(f"every pixel is approx {(m_per_pixel)} m long, or {(m_per_pixel

area = img.shape[0] * m_per_pixel * img.shape[1] * m_per_pixel
print(f"area of this image is approx {area} m^2")
```

```
every pixel is approx 7.55952380952381e-05 m long, or 5.714640022675738e-
09 m^2
area of this image is approx 0.06967728000000001 m^2
```

threshold & adaptive threshold

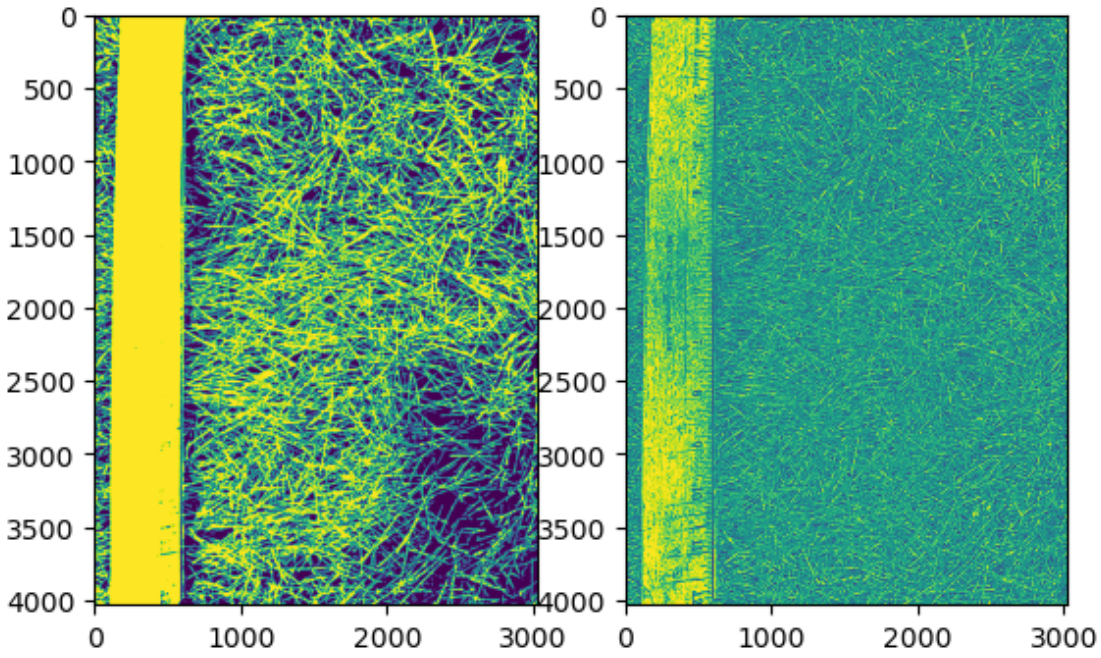
just trying different things out

In [181]:

```
def thresholds(img):
    img = cv2.medianBlur(img,5)
    _, th1 = cv2.threshold(img,100,255,cv2.THRESH_BINARY)
    th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C, cv

    plt.figure()

    plt.subplot(1,2,1)
    plt.imshow(th1)
    plt.plot()

    plt.subplot(1,2,2)
    plt.imshow(th2)
    plt.plot();
```

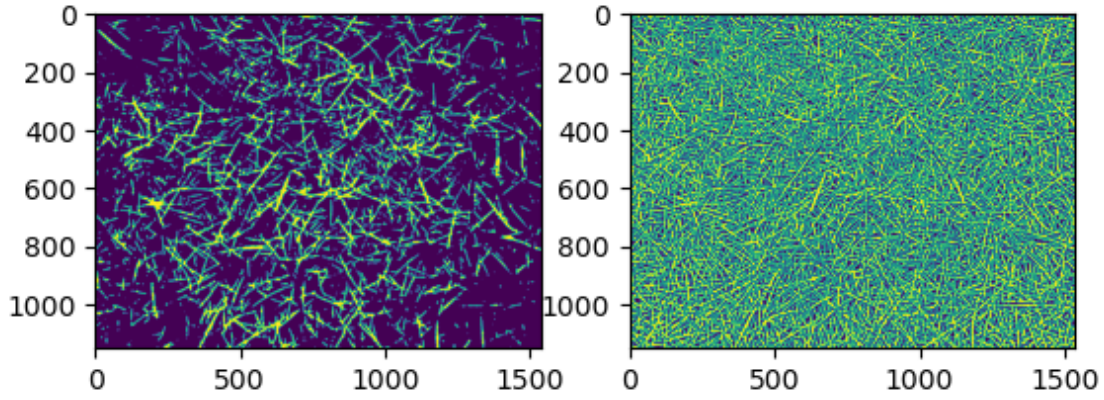
In [182]:

```
img = cv2.imread('./images/12in.jpg', cv2.IMREAD_GRAYSCALE)
thresholds(img)
```



In [183]:

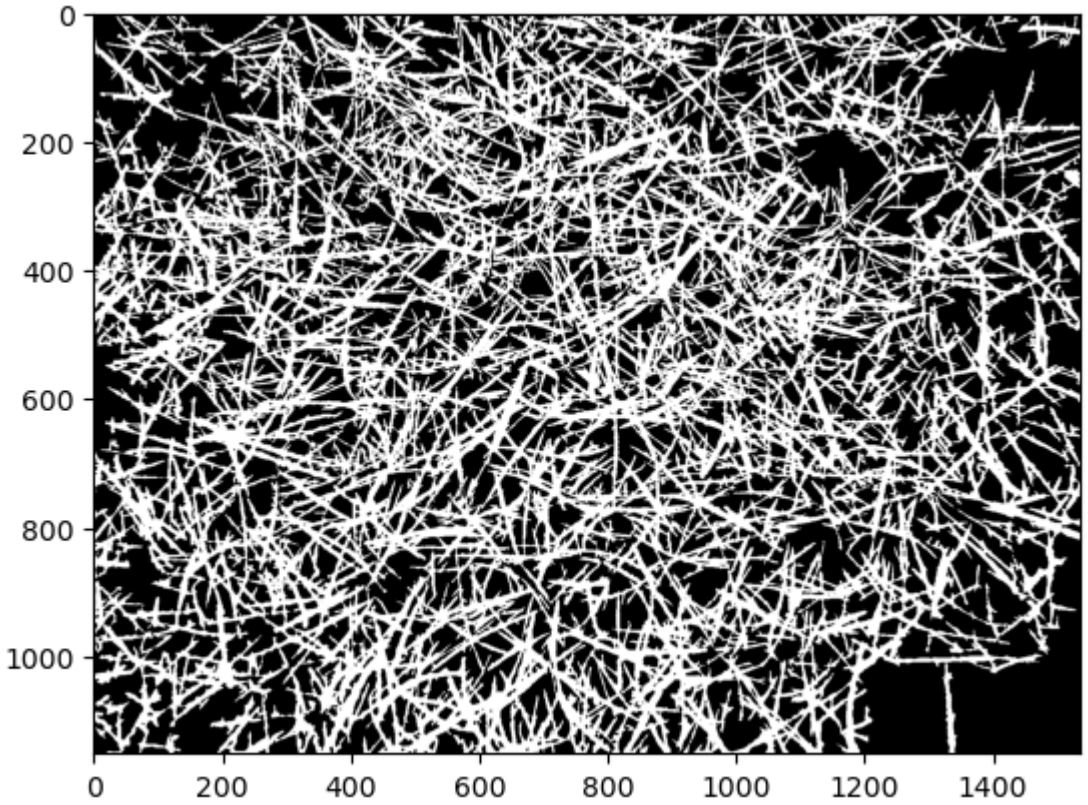
```
img = cv2.imread('./images/green-bermuda-grass.jpg', cv2.IMREAD_GRAYSC
thresholds(img)
```



In [184]:

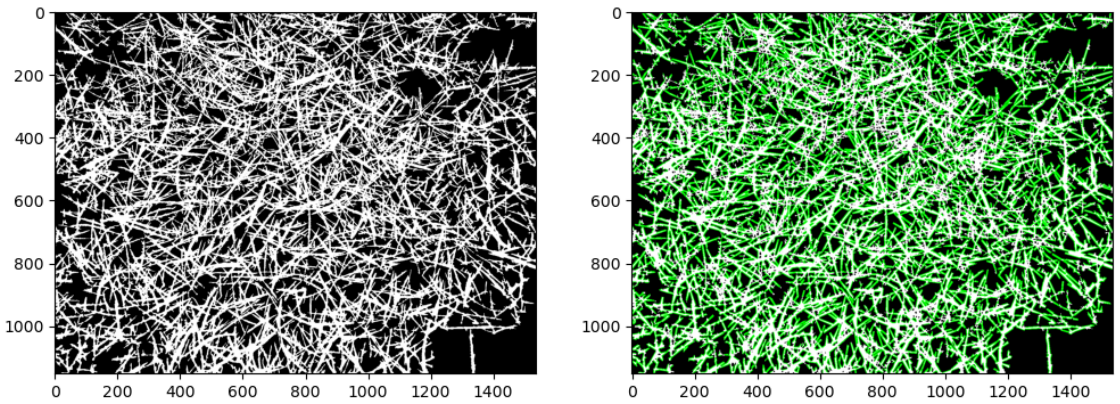
```
image = cv2.imread('./images/green-bermuda-grass.jpg', cv2.IMREAD_UNCH
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Use cv2.THRESH_OTSU as a flag to automatically determine the thresho
_, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH

# getting mask with connectedComponents
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStat
mask = np.zeros_like(labels, dtype=np.uint8)
mask[labels == 1] = 255
print('Area:', stats[1, cv2.CC_STAT_AREA])

plt.figure()
plt.imshow(mask, cmap='gray')
plt.show();
cv2.imwrite("mask_cc.jpg", mask);

```

```
Area: 780630
```



In [185]:

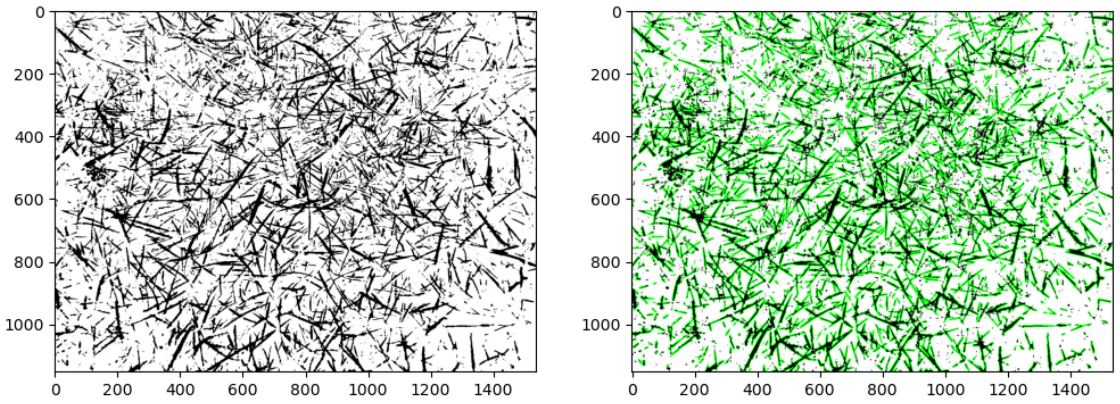
```
img = cv2.imread('mask_cc.jpg')
lines = count_lines_lsd(img)
H, W, _ = img.shape
print(f'density is approx {lines} lines /  {(10000*H*W*(m_per_pixel**2
```



```
density is approx 398 lines /  101.11895510204084 cm^2 =  3.94 blades/cm^
2
```

In [186]:

```
img = cv2.imread('mask.jpg')
lines = count_lines_lsd(img)
H, W, _ = img.shape
print(f'density is approx {lines / (10000*H*W*(m_per_pixel**2)):.2f} b
```



```
density is approx 3.05 blades/cm^2
```

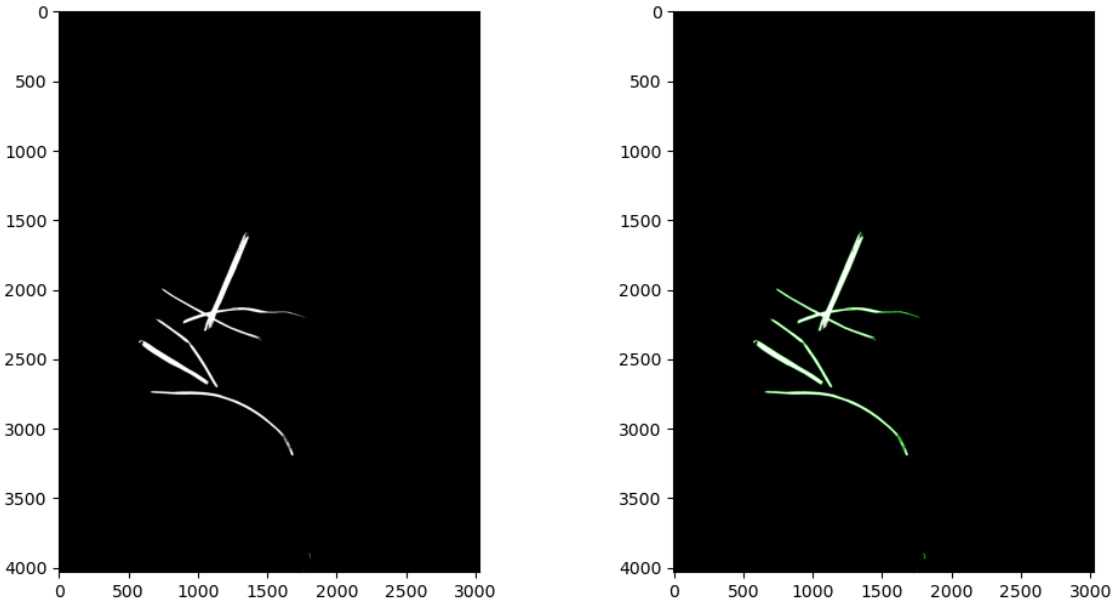
make it a little more organized / streamlined

In [235]:

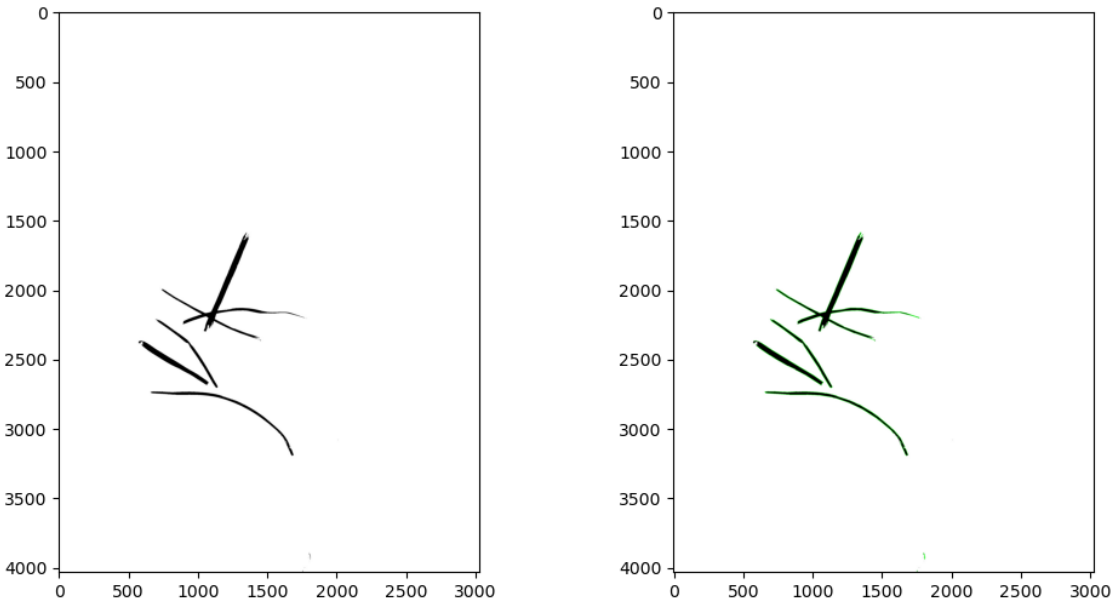
```python
def density_pipeline(img, area_img, ruler_cm=30.48):
    # mask from counting green pixels
    lower_green = np.array([40, 40, 40])
    upper_green = np.array([120, 255, 256])
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    count, mask = count_grass_pixels(hsv_img, lower_green, upper_green
    cv2.imwrite('./results/mask.jpg', mask)

    # mask from connected components
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.TH
    num_labels, labels, stats, centroids = cv2.connectedComponentsWith
    max_area_label = np.argmax(stats[1:, cv2.CC_STAT_AREA]) + 1
    mask_cc = np.zeros_like(labels, dtype=np.uint8)
    mask_cc[labels == max_area_label] = 255
    area_cc = stats[max_area_label, cv2.CC_STAT_AREA]
    cv2.imwrite('./results/mask_cc.jpg', mask_cc)

    # area from ruler
    cm_per_pixel = ruler_cm / area_img.shape[0]
    cm2_per_pixel = cm_per_pixel ** 2

    # find lines for mask and mask_cc
    mask = cv2.imread('./results/mask.jpg')
    lines = count_lines_lsd(mask)
    H, W, _ = mask.shape
    print(f'pixel color: density is approx {lines} lines /  {(H*W*cm2_

    mask_cc = cv2.imread('./results/mask_cc.jpg')
    lines = count_lines_lsd(mask_cc)
    H, W, _ = mask_cc.shape
    print(f'connected component: density is approx {lines} lines /  {(

```

In [236]:

```
img = cv2.imread('./images/bg_overlap_adjusted.jpg')
ruler_img = cv2.imread('./images/bg_overlap_12in.jpg')

assert img is not None, "file could not be read, check with os.path.ex
assert ruler_img is not None, "file could not be read, check with os.p

density_pipeline(img, ruler_img, 24.0)
```



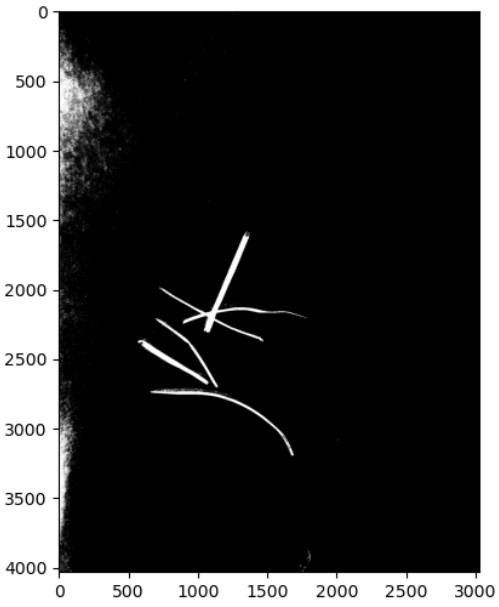

```
pixel color: density is approx 8 lines /   432.00 cm^2 =  0.02 blades/cm^
2
```

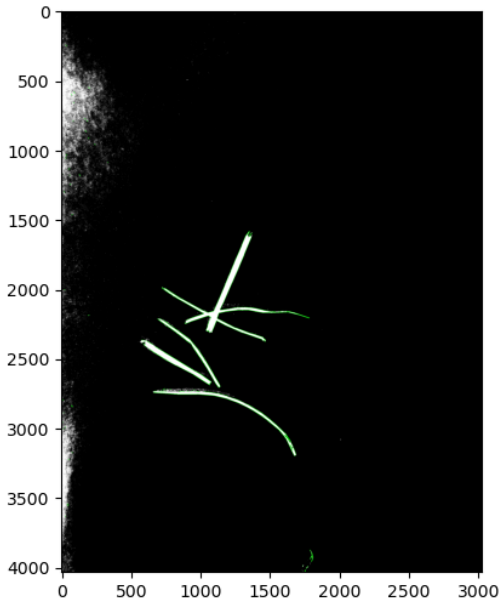



```
connected component: density is approx 7 lines /   428.37 cm^2 =  0.02 bl
ades/cm^2
```

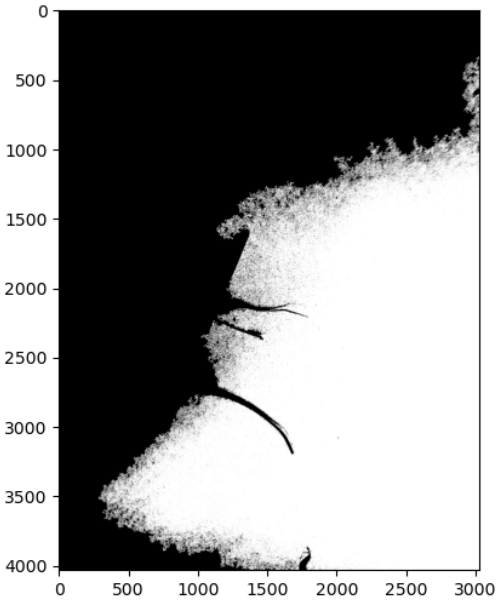
In [237]:

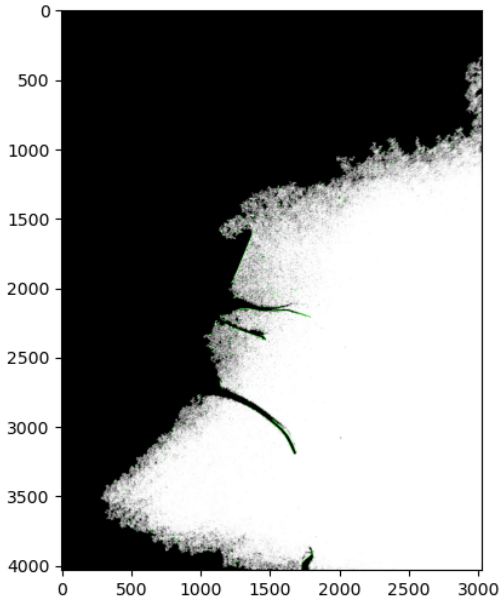
```
img = cv2.imread('./images/bg_overlap.jpg')
ruler_img = cv2.imread('./images/bg_overlap_12in.jpg')

assert img is not None, "file could not be read, check with os.path.ex
assert ruler_img is not None, "file could not be read, check with os.p

density_pipeline(img, ruler_img)
```






```
pixel color: density is approx 11 lines /   696.77 cm^2 =  0.02 blades/cm
^2
```

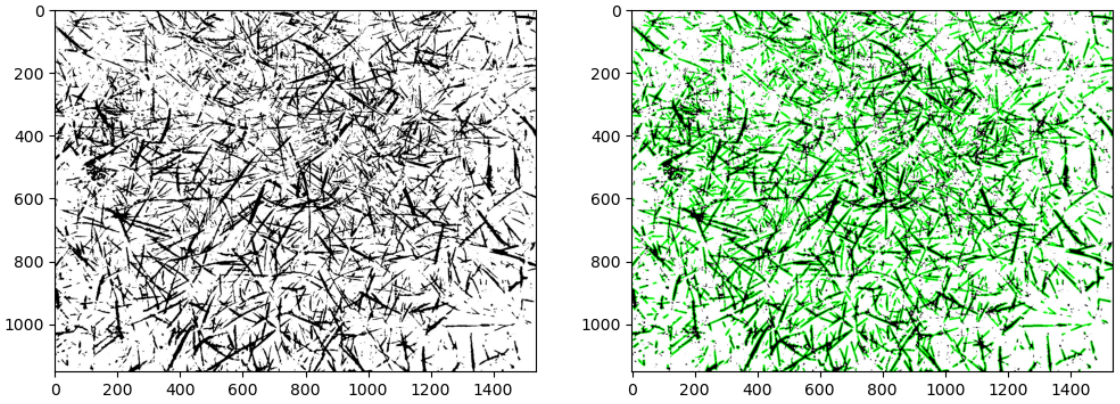




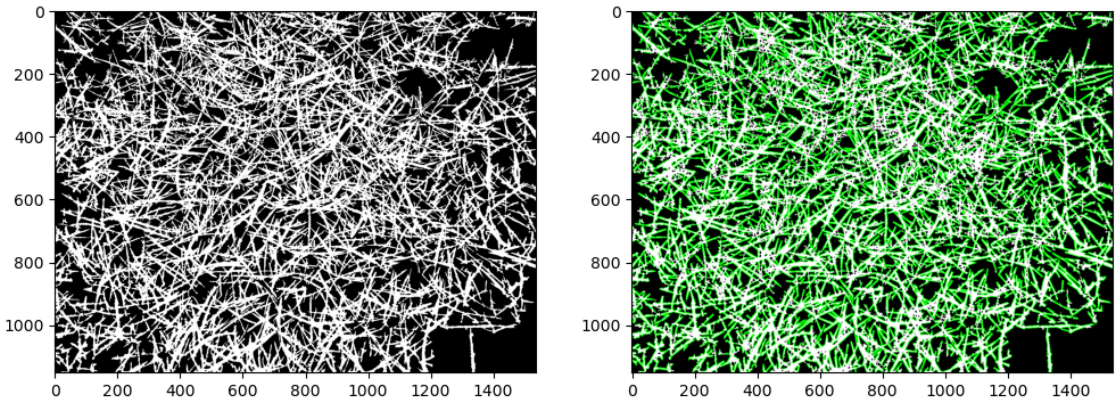

```
connected component: density is approx 17 lines /   320.45 cm^2 =  0.05 b
lades/cm^2
```

In [238]:

```
1 img = cv2.imread('./images/green-bermuda-grass.jpg')
2 ruler_img = cv2.imread('./images/12in.jpg')
3
4 assert img is not None, "file could not be read, check with os.path.ex
5 assert ruler_img is not None, "file could not be read, check with os.p
6
7 density_pipeline(img, ruler_img)
```



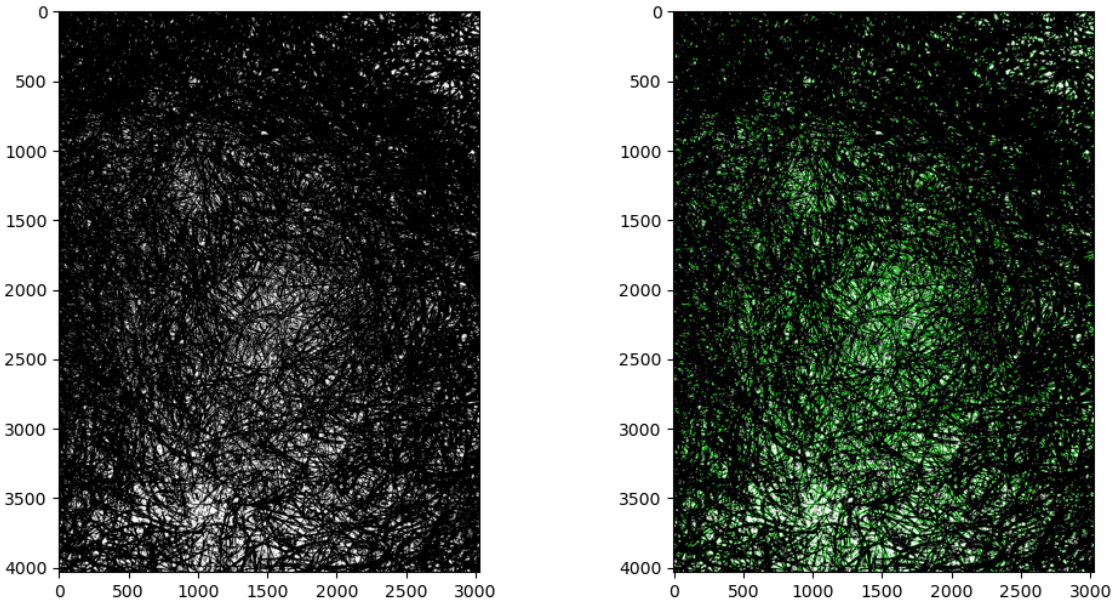

```
pixel color: density is approx 308 lines /   101.12 cm^2 =  3.05 blades/c
m^2
```



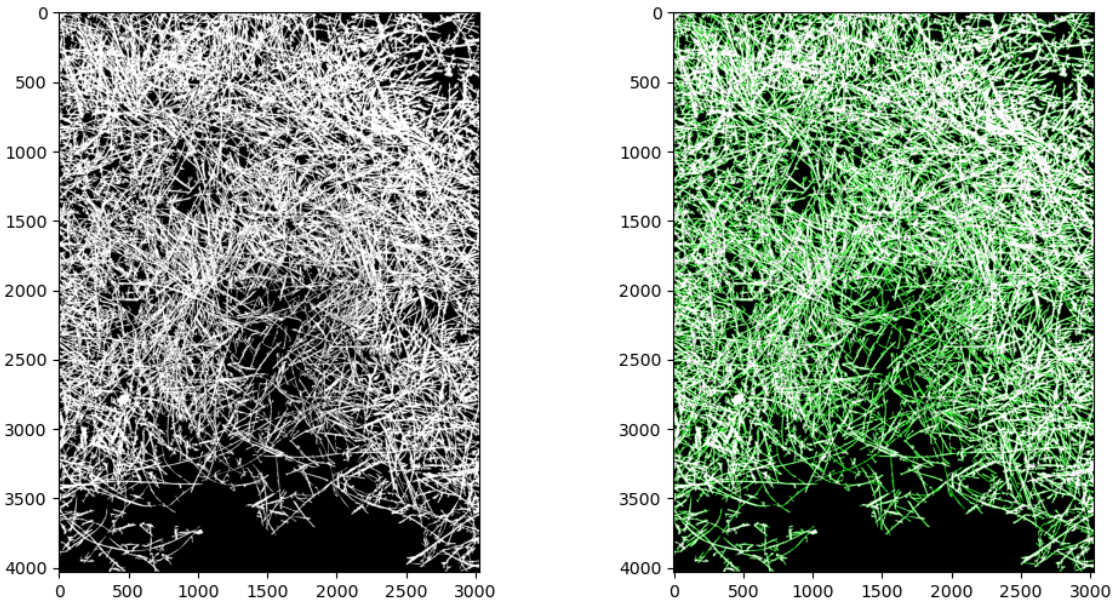

```
connected component: density is approx 398 lines /   44.61 cm^2 =  8.92 b
lades/cm^2
```

In [239]:

```
img = cv2.imread('./images/inside.jpg')
ruler_img = cv2.imread('./images/inside_12in.jpg')

assert img is not None, "file could not be read, check with os.path.ex
assert ruler_img is not None, "file could not be read, check with os.p

density_pipeline(img, ruler_img)
```



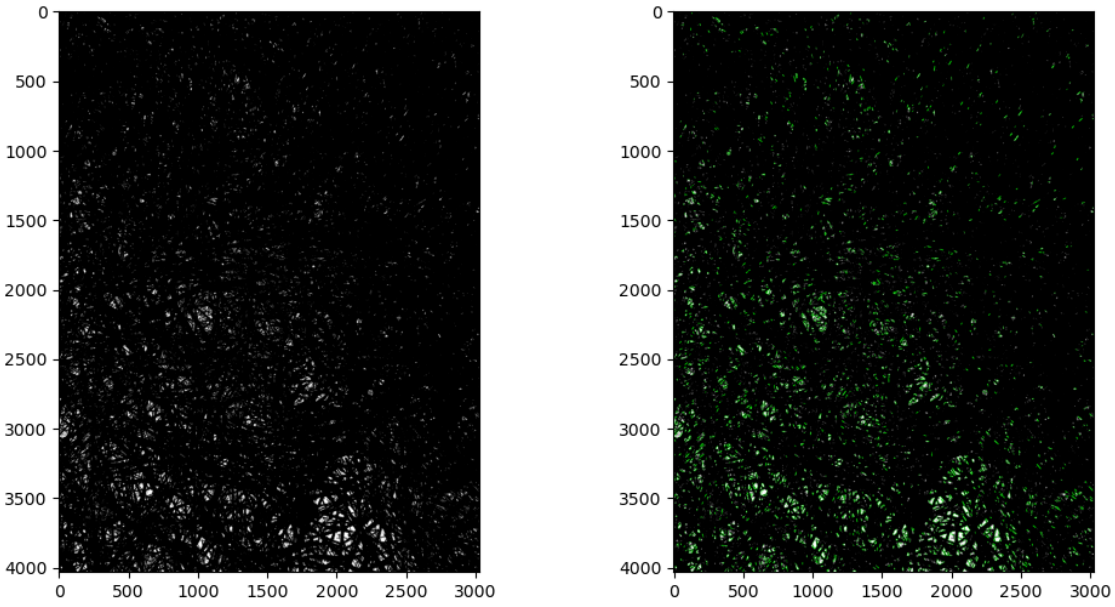
pixel color: density is approx 731 lines / 696.77 cm^2 = 1.05 blades/cm^2



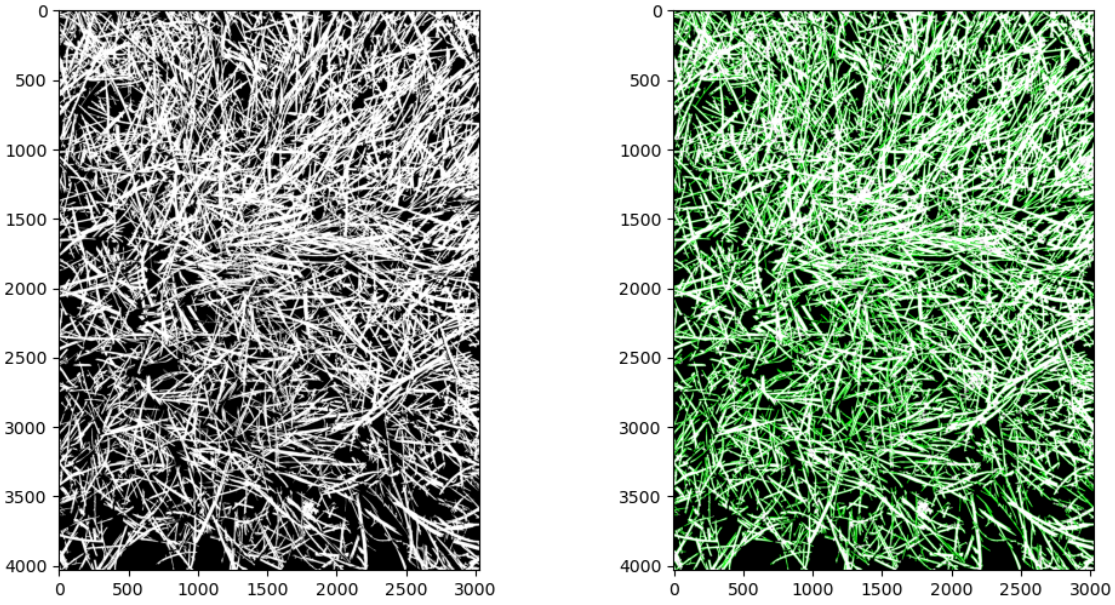
connected component: density is approx 1591 lines / 337.80 cm^2 = 4.71 blades/cm^2

In [240]:

```
img = cv2.imread('./images/outside.jpg')
ruler_img = cv2.imread('./images/outside_12in.jpg')

assert img is not None, "file could not be read, check with os.path.ex
assert ruler_img is not None, "file could not be read, check with os.p

density_pipeline(img, ruler_img)
```




```
pixel color: density is approx 191 lines /   696.77 cm^2 =  0.27 blades/c
m^2
```




```
connected component: density is approx 1303 lines /   352.35 cm^2 =  3.70
blades/cm^2
```

lines counted using lsd does not equal to # of blades, but limiting the length of lines helped estimate the density calculation

which approach gives results closer to actual value or human guesses?

come back to this later: resolving the overlaps

since lsd counts a short line, how to count a curvy grass blade as 1?

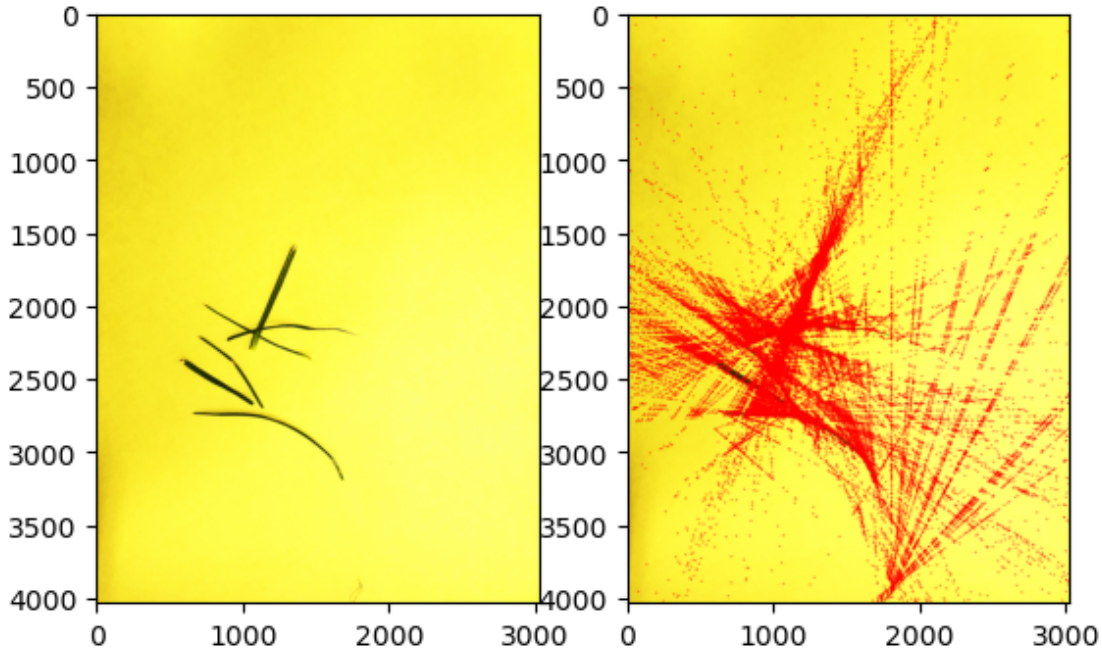
In [78]:

```
def find_line_intersections(lines):
    intersections = []
    for line1, line2 in combinations(lines, 2):
        x1, y1, x2, y2 = line1[0]
        x3, y3, x4, y4 = line2[0]

        # Check if lines are not parallel
        det = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)

        if det != 0:
            intersection_x = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 -
            intersection_y = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 -

            intersections.append((int(intersection_x), int(intersectio

    return intersections

```

In [139]:

```
img = cv2.imread('./images/bg_overlap_adjusted.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Use the LineSegmentDetector to detect lines
lsd = cv2.createLineSegmentDetector()
lines, _, _, _ = lsd.detect(gray)

# Draw the detected lines on the image
lines_img = np.copy(img)
for line in lines:
    x1, y1, x2, y2 = map(int, line[0])
    cv2.line(lines_img, (x1, y1), (x2, y2), (0, 255, 0), 2)

intersections = find_line_intersections(lines)

# Draw the intersections on the image
for point in intersections:
    cv2.circle(lines_img, point, 5, (0, 0, 255), 2)

print(len(intersections), len(lines))
plt.figure()

plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.plot()

plt.subplot(1,2,2)
plt.imshow(cv2.cvtColor(lines_img, cv2.COLOR_BGR2RGB))
plt.plot();

```

```
82621 407
```



come back to this later: compare to known density patches

In [ ]:
```
1
```

come back to this later: use projection from a video like proj5

In [ ]:
```
1
```

come back to this later: TP-LSD

https://github.com/Siyuada7/TP-LSD (https://github.com/Siyuada7/TP-LSD)

In [ ]:
```
1
```

Graveyard

Algorithm from the paper: Run-Length Encoding (RLE)

searching for contiguous pixels along the major axis in an input binary image and replacing them by a single instance of the run count for each set of contiguous pixels util the end of the image is reached.

- Step 1: read image and remove artifacts and extraneous material that do not meet the minimum threshold pixel prequirement
- Step 2: Set minimum and maximum thresholds for the contiguous pixel widths
- Step 3: Create a zero matrix of the image size with a one pixel buffer
- Step 4: Assign original matrix to the zero matrix
- Step 5: Create a vector matrix from the transpose of the original matrix
- Step 6: Find positions where one element is different from the next
- Step 7: Find run lengths and values after determining the end position of each run
- Step 8: Assign zeros for all widths above and below the specified threshold range
- Step 9: Perform run-length decoding
- Step 10: Repeat Steaps 5 through 8 for each image angle from 0 to 90 degrees at 5 degrees interval
- Step 11: Calculate image statistics

In [28]:

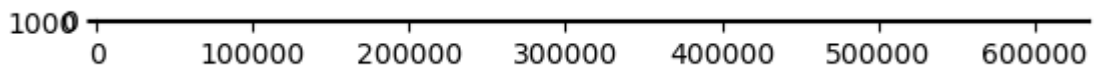
```python
def preprocess_image(img, threshold=128):
    # Step 1: Read image and remove artifacts
    _, binary_image = cv2.threshold(img, threshold, 255, cv2.THRESH_BI
    return binary_image

def run_length_encoding(binary_image, min_threshold=5, max_threshold=1
    # Step 3: Create a zero matrix with a one-pixel buffer
    zero_matrix = np.zeros((binary_image.shape[0] + 2, binary_image.sh

    # Step 4: Assign original matrix to the zero matrix
    zero_matrix[1:-1, 1:-1] = binary_image

    encoded_runs = []

    # Step 10: Repeat Steps 5 through 8 for each image angle from 0 to
    for angle in range(0, 91, 5):
        rotated_image = rotate_image(binary_image, angle)
        runs = find_runs(rotated_image)
        filtered_runs = filter_runs(runs, min_threshold, max_threshold
        decoded_image = run_length_decoding(filtered_runs)

        # Step 11: Calculate image statistics
        calculate_image_statistics(decoded_image)
        print(decoded_image)

        plt.figure()
        plt.imshow(decoded_image)
        plt.show();

def rotate_image(image, angle):
    # Step 5: Create a vector matrix from the transpose of the origina
    rotated_image = np.transpose(image)

    # Rotate the image
    rotation_matrix = cv2.getRotationMatrix2D((rotated_image.shape[1]
    rotated_image = cv2.warpAffine(rotated_image, rotation_matrix, (ro

    return rotated_image

def find_runs(image):
    # Step 6: Find positions where one element is different from the n
    runs = []
    for row in image:
        run_lengths = [sum(1 for _ in group) for key, group in groupby
        run_values = [key for key, group in groupby(row)]
        runs.append((run_lengths, run_values))

    return runs

def filter_runs(runs, min_threshold, max_threshold):
    # Step 7: Find run lengths and values after determining the end po
    filtered_runs = []
    for run_lengths, run_values in runs:
        filtered_lengths = [length if min_threshold <= length <= max_t
        filtered_runs.append((filtered_lengths, run_values))
```

```
56
57      return filtered_runs
58
59  def run_length_decoding(runs):
60      # Step 8: Assign zeros for all widths above and below the specifie
61      decoded_image = np.zeros((len(runs), sum(sum(run_lengths) for run_
62
63      # Perform run-length decoding
64      for i, (run_lengths, run_values) in enumerate(runs):
65          start = 0
66          for length, value in zip(run_lengths, run_values):
67              decoded_image[i, start:start + length] = value
68              start += length
69
70      return decoded_image
71
72  def calculate_image_statistics(image):
73      # Implement image statistics calculation (mean, median, etc.) here
74      pass
```

In [31]:

```
1  # Specify the path to your image
2  image = cv2.imread('./images/green-bermuda-grass.jpg', cv2.IMREAD_GRAY
3
4  # Step 2: Set minimum and maximum thresholds for the contiguous pixel
5  min_threshold = 5
6  max_threshold = 100
7
8  binary_image = preprocess_image(image)
9  run_length_encoding(binary_image, min_threshold, max_threshold)
10
```

```
[[255 255 255 ...   0   0   0]
 [255 255 255 ...   0   0   0]
 [255 255 255 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
```




```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

In [ ]:

```
1
```