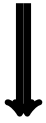
Crop Analytics using Python
Segmentation using Vegetation Mask.

My Project as Intern at Aviac Technologies

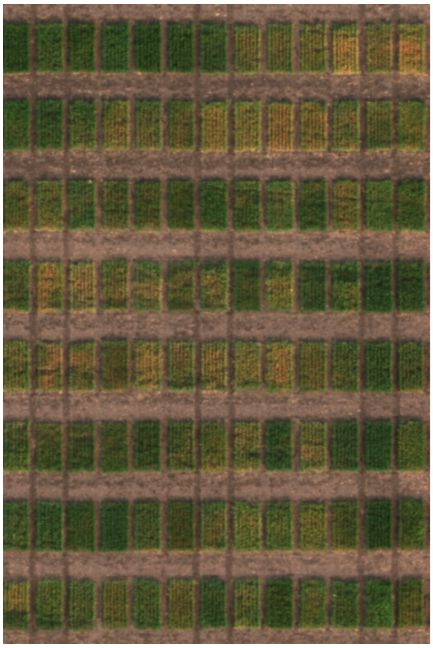
– Nishant –

Please click and scroll here to see the pdf

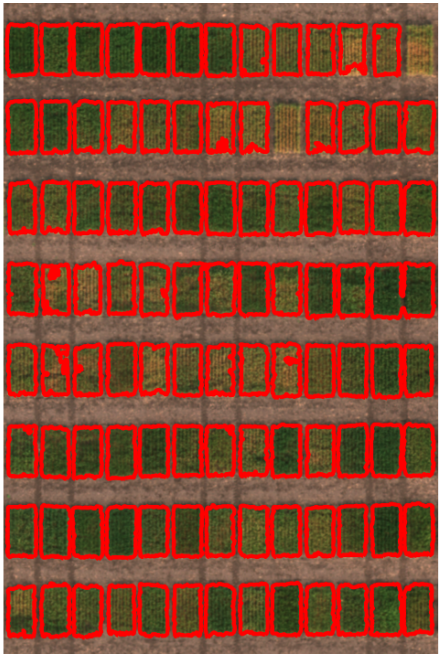


Introduction

This document provides overview of my python code on segmenting green crops from soils and segregate them into individual plots. This helps to initiate a plot wise analysis of complete crop to enhance the process of extracting required information and assessing crop health and yield.



From Drone Image



To Segmented into Plots

Segmentation Using Vegetation Mask

Method and Code

The concept behind segmenting crops from soil into plots is Masking. A mask is prepared using the crop image from the drone by providing a threshold. So for example, I set a threshold for the pixel to have X value or above. Therefor this example mask will only consist of pixels out of all pixels from the crop image which satisfy the threshold set by me. Now that I have obtained the mask, I plot Contours. Contours are like polygon to connect the pixels togethor into a ploygon if they share boundary. This might have given some hint about whats the process.
Lets go in depth of each process I followed.

Step 1: Installing Rasterio, Shaply, Skimage and Geopandas Importing these libraries in Python helps us use predefined functions to obtain required results.

Code 1.1:

```
import rasterio
import numpy as np
import matplotlib.pyplot as plt
import geopandas as gpd
from skimage.measure import find_contours
from shapely.geometry import Polygon
```

Step 2: Reading Drone Captured crop images using Rasterio Library

We read only the required bands from the crop image i.e Red Band and NIR Band and use it to calculate NDVI from the crop image. **NDVI is Normalized Difference Vegetation Index, it is a metric that measures the health and density of vegetation**

Code 1.2:

```
tif_file = "5band/EX_13_DAP_73_2021_Casselton_YT_07-19_5band.tif"

with rasterio.open(tif_file) as src:
    stacked_array = src.read()

red_band = stacked_array[0, :, :]
nir_band = stacked_array[4, :, :]
ndvi = (nir_band - red_band) / (nir_band + red_band)
```

Step 3: Applying Threshold for Vegetation

We apply a threshold to the NDVI values to create a vegetation mask that highlights the areas of vegetation based on their health and density.

Code 1.3:

```
# Applying threshold for vegetation
vegetation_mask = ndvi > np.mean(ndvi)
```

Step 4: Finding Contours of Vegetation Using Skimage

We find contours of vegetation areas in the vegetation mask using the 'find contours' function from the Skimage library.

Code 1.4:

```
# Find contours using skimage.measure.find_contours
contours = find_contours(vegetation_mask, 0.5)
```

Step 5: Creating a GeoDataFrame with Polygons

We create a GeoDataFrame to store polygons of the detected vegetation areas.

Code 1.5:

```
# Create a GeoDataFrame with polygons
polygons = [Polygon(np.flip(contour, axis=1)) for contour in contours]
gdf_polygons = gpd.GeoDataFrame(geometry=polygons, crs=src.crs)
```

Step 6: Filtering Polygons Based on Area

We filter the polygons based on their area, keeping only those whose area is greater than 1.8 times the average area.

Code 1.6:

```
# Filtering polygons with area less than the average area
average_area = gdf_polygons.geometry.area.mean()
filtered_gdf = gdf_polygons[gdf_polygons.geometry.area
                            >= 1.8 * average_area]
```

Step 7: Calculating Total Number of Polygons

We calculate and display the total number of polygons that were filtered and retained. Further we save the Filtered GeoDataFrame to a Shapefile

Code 1.7:

```
# Total number of polygons
total_polygons = len(filtered_gdf)
print(f"Total number of polygons: {total_polygons}")
# Saving the filtered GeoDataFrame to a shapefile
output_shapefile = "filtered_vegetation_polygons.shp"
filtered_gdf.to_file(output_shapefile)
```

Step 9: Plotting the Results

We plot the RGB image and overlay the filtered vegetation polygons.

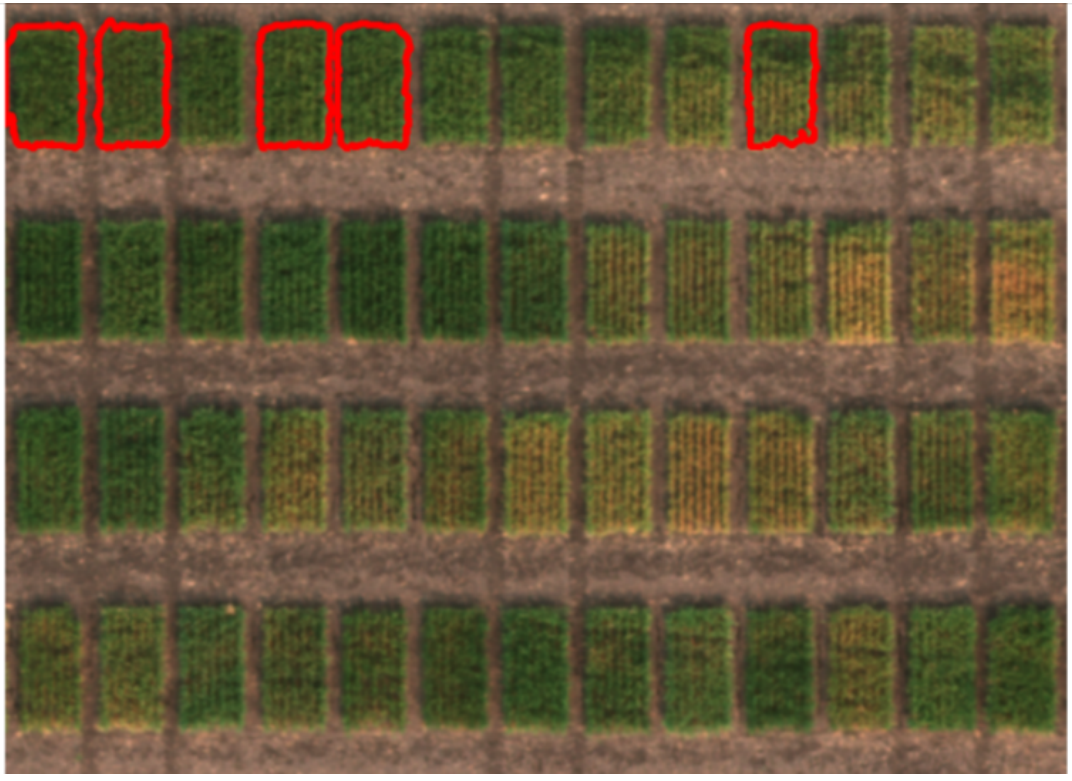
Code 1.8:

```
# Plotting (optional)
fig, ax = plt.subplots(figsize=(25, 25))
# Selecting bands 2, 1, 0 for RGB channels
rgb_image = stacked_array[[0, 1, 2], :, :]
rgb_image = rgb_image / rgb_image.max()
ax.imshow(rgb_image.transpose((1, 2, 0)))
filtered_gdf.plot(ax=ax, color='none', edgecolor='red', linewidth=3)
ax.axis('off')
plt.savefig("Image2.png")
plt.show()
```

Next - Analysis using the segmented crops

•

1. Locating Ith to Jth Polygons



My this code helps to select only plots of intrest and analyse thier characterstics

Code 1.9:

```
# Plotting only the first 5 polygons
fig, ax = plt.subplots(figsize=(35, 35))

# Selecting bands 2, 1, 0 for RGB channels
rgb_image = stacked_array[[0, 1, 2], :, :]
rgb_image = rgb_image / rgb_image.max()

ax.imshow(rgb_image.transpose((1, 2, 0)))
filtered_gdf.iloc[0:5].plot(ax=ax, color='none', edgecolor='red',
                linewidth=3)  # Plot only the first 5 polygons
# plt.title('RGB Image with First 5 Vegetation Polygons')
ax.axis('off')
plt.show()
```

Now - We calculate Mean NDVI for each of the selected plots

.

```
Polygon 0 NDVI Mean -  0.6105210185050964
Polygon 1 NDVI Mean -  0.7594797015190125
Polygon 2 NDVI Mean -  0.32857823371887207
Polygon 3 NDVI Mean -  0.24837400019168854
Polygon 4 NDVI Mean -  0.5669962167739868
```

Output

Code 1.10:

```
mask = np.zeros_like(ndvi, dtype=bool)
def point_in_polygon(point, polygon_vertices):
    x, y = point
    n = len(polygon_vertices), inside = False
    p1x, p1y = polygon_vertices[0]
    for i in range(n + 1):
        p2x, p2y = polygon_vertices[i % n]
        if y > min(p1y, p2y):
            if y <= max(p1y, p2y):
                if x <= max(p1x, p2x):
                    if p1y != p2y:
                        xinters = (y - p1y) * (p2x - p1x)
                                  / (p2y - p1y) + p1x
                    if p1x == p2x or x <= xinters:
                        inside = not inside
        p1x, p1y = p2x, p2y
    return inside
for idx, polygon in selected_polygons.iterrows():
    polygon_coordinates = np.arr(polgon['gomtry'].exterior.xy).T
    polygon_vertices = polygon_coordinates.tolist()
    for y in range(mask.shape[0]):
        for x in range(mask.shape[1]):
            point = (y, x)
            is_inside = point_in_polygon(point, polygon_vertices)
            mask[y, x] = is_inside
    mean_ndvi = np.mean(ndvi[mask])
    print(f"Mean NDVI for Polygon {idx}: Total NDVI Mean -
    {np.mean(ndvi)}, Polygon {idx} NDVI Mean -  {mean_ndvi}")
```