



SprayDAT

Spray Droplet Analysis Tool

SprayDAT (Spray Droplet Analysis Tool):
Python-based Image Analysis Tool for Spray Quality Assessment

Daewon Koo¹

Caleb A. Henderson¹

Shawn D. Askew²

¹Graduate Research Assistant ²Professor
School of Plant and Environmental Sciences
Virginia Tech

Contact: saskew@vt.edu

Table of Contents

<i>Introduction.....</i>	<i>2</i>
<hr/>	
<i>Requirements.....</i>	<i>4</i>
<hr/>	
<i>Prepare Scanned Images.....</i>	<i>5</i>
<hr/>	
<i>Operate SprayDAT.....</i>	<i>6</i>

Introduction

SprayDAT is a python-based image analysis tool that can assess the spray quality of agricultural pesticide application. It is optimized to process the scanned images of white Kraft paper with stains from 50% blue colorant solution. Utilization of white Kraft paper as a sampler, and blue colorant solution as a proxy of pesticides to assess the spray deposition was described in Koo et al. (2024). SprayDAT offers the batch-process of large sets of scanned images, which will allow end-users to sample larger areas to better understand the spray quality by utilizing the cost-effective Kraft paper and blue colorant method.

Reference: Koo, D., Gonçalves, C. G., & Askew, S. D. (2024). Agricultural spray drone deposition, Part 1: Methods for high throughput spray pattern analysis. *Weed Science*. (in press).

Requirements

1. Any type of scanner that can scan the white Kraft paper samples.
2. Minimum requirements to operate Python 3.9. or higher
 - Modern Operating System
 - Windows 7 or 10
 - Mac OS X 10.11 or higher, 64-bit
 - Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)
 - x86 64-bit CPU (Intel / AMD architecture). ARM CPUs are not supported.
 - 4 GB RAM
 - 5 GB free disk space
3. Essential Python packages to download
 - OpenCV
 - Matplotlib
 - Numpy
 - Pandas

Prepare Scanned Images

After white Kraft paper samples are prepared, they needed to be scanned by with any types of scanners. 600 dpi resolution has been tested to precisely estimate spray coverage, droplet density, droplet spectra, and total spray volume. Using scanned images with lower or higher than 600 dpi resolution is possible with SprayDAT, but with the cost of the lower detection accuracy or the possibility of detecting erroneous small stains, respectively. If there is an option for color selection, choose “full color” options rather than automatic selection. Automatic selection might alter the hue of stains, especially for the small stains, which may not be fully captured by the current threshold for binarization of SprayDAT. The scanned image samples in the repository was scanned by Ricoh MP C307 color scanner (Ricoh, Tokyo, Japan) at 600 dpi with “Full Color : Text / Photo” setting. Scanned images in a format of your choice should be stored in one or more folders.

Operate SprayDAT

Once SprayDAT.ipynb is open, you can run the code line by line by either clicking “▶” or typing “Ctrl + Enter”

1. Import essential python packages

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
```

2. Define functions

1) img2binary – binarization

```
def img2binary(filename,threshold=154):
    """This function takes in an image path /(relative or absolute/) as a string and converts it to a binary mask"""
    rawname = r'{}'.format(filename)
    img = cv2.imread(rawname)
    imggray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    _,binary = cv2.threshold(imggray,threshold,255,cv2.THRESH_BINARY_INV)
    return binary
```

2) binaryStats – generates stats from binary images

```
def binaryStats(binary):
    """This function takes in a binary image and returns the percent cover of white pixels and a sorted contour list"""
    white_pix = np.sum(binary == 255)
    black_pix = np.sum(binary == 0)
    total = binary.shape[0]*binary.shape[1]
    percentcover = (white_pix/total)*100
    contours, _ = cv2.findContours(binary, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
    cnt = sorted(contours, key=cv2.contourArea )
    return percentcover, cnt
```

3) PercVol – interpolation method for Dv0.1, VMD (Dv0.5), and Dv0.9 estimation

```
def PercVol(lis, perc):
    """This function interpolates between two droplet volumes depending on the set threshold (cumulative volume)"""
    if len(lis) == 1:
        return lis[0] # Return the empty list if it is empty

    lis.sort()
    percentile = np.sum(lis) * perc
    sum2 = 0

    for i in range(0, len(lis)):
        sum2 = sum2 + lis[i]
        if sum2 > percentile:
            break

    if i == 0:
        return 0 + ((percentile - np.sum(lis[0:i])) / (np.sum(lis[0:i+1]) - np.sum(lis[0:i]))) * (lis[i])
    else:
        return lis[i-1] + ((percentile - np.sum(lis[0:i])) / (np.sum(lis[0:i+1]) - np.sum(lis[0:i]))) * (lis[i] - lis[i-1])
```

3. Set the directory

directory = r"Path of the folder where you store scanned images"

4. Calculate stain coverage and stain diameters for all scanned image (.jpg, .png, etc.) stored in the directory (it will take times)

```
#This Code Block is verified to work, if the photos are all within one folder
#pcarray = []
contours=[]
dpi=600
um2perpix = (1/(dpi/25400))
for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    print(f)
    # checking if it is a file
    if os.path.isfile(f):
        binary=img2binary(f,154)
        binary2 = binary[1000:5000, 1000:4000]
        total = binary2.shape[0]*binary2.shape[1]
        PC,cnt = binaryStats(binary2)
        #pcarray.append(PC)
        contours.append([filename,PC,cnt,[round(2*np.sqrt(((cv2.contourArea(i)+1)/np.pi))*um2perpix,4)for i in cnt]])
```

* This code block works when all scanned images are stored in one folder. Use another code blocks (Pg 14) if the files are arranged in a folder with multiple sub-folder.

(1) Change dpi if your scanned images have different resolution than 600.

dpi=600

(2) If you want to change the threshold for binarization, change value here:

binary=img2binary(f,154)

* The threshold 154 was found to best encompass all of the detected droplets without highlighting potential noise in the scanned images, such as dust or dirt particles.

(3) If you want to select certain area in your scanned images, change values as:

binary2 = binary[y:y+height, x:x+width]

5. Put the generated data in a dataframe

```
df2 = pd.DataFrame(contours, columns=['imageID', 'PercentCover', 'contours', 'stain_diameter(um)'])
```

df2

	imageID	PercentCover	contours	stain_diameter(um)
0	img369.jpg	8.242858	[[[1706 3999]], [[1708 3999]]], [[1674 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
1	img367.jpg	6.759967	[[[2071 3999]], [[2075 3999]]], [[1710 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
2	img365.jpg	7.248508	[[[2357 3999]], [[2359 3999]]], [[2008 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
3	img363.jpg	7.008417	[[[1959 3999]]], [[462 3999]], [[465 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
4	img361.jpg	5.341667	[[[2715 3999]], [[2716 3999]]], [[2568 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
...
115	img169.jpg	16.201942	[[[2765 3999]], [[2766 3999]]], [[2489 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
116	img157.jpg	14.854517	[[[1666 3999]], [[1667 3999]]], [[1419 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
117	img171.jpg	20.874367	[[[2942 3999]], [[2945 3999]]], [[2867 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
118	img173.jpg	18.131700	[[[2924 3999]], [[2929 3999]]], [[2681 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...
119	img175.jpg	29.483650	[[[2558 3999]], [[2559 3999]]], [[2548 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...

120 rows × 4 columns

Filename of each image, stain coverage, and an array of detected stain diameter (µm) are sorted in the dataframe

6. Apply spread factor to estimate actual droplet diameter (µm)

```
df2['actual_diameter(um)']=df2['stain_diameter(um)'].apply(lambda x: [0.9726*d*0.7836 for d in x])
```

df2

	imageID	PercentCover	contours	stain_diameter(um)	actual_diameter(um)
0	img369.jpg	8.242858	[[[1706 3999]], [[1708 3999]]], [[1674 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
1	img367.jpg	6.759967	[[[2071 3999]], [[2075 3999]]], [[1710 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
2	img365.jpg	7.248508	[[[2357 3999]], [[2359 3999]]], [[2008 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
3	img363.jpg	7.008417	[[[1959 3999]]], [[462 3999]], [[465 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
4	img361.jpg	5.341667	[[[2715 3999]], [[2716 3999]]], [[2568 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
...
115	img169.jpg	16.201942	[[[2765 3999]], [[2766 3999]]], [[2489 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
116	img157.jpg	14.854517	[[[1666 3999]], [[1667 3999]]], [[1419 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
117	img171.jpg	20.874367	[[[2942 3999]], [[2945 3999]]], [[2867 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
118	img173.jpg	18.131700	[[[2924 3999]], [[2929 3999]]], [[2681 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...
119	img175.jpg	29.483650	[[[2558 3999]], [[2559 3999]]], [[2548 3999]...	[47.7681, 47.7681, 47.7681, 47.7681, 47.7681, ...	[20.123650981419132, 20.123650981419132, 20.12...

120 rows × 5 columns

7. Calculate total droplet counts in the scanned images

```
df2['count']=df2['actual_diameter(um)'].str.len()
```

```
columns_to_drop = ['contours','stain_diameter(um)']
```

```
df3=df2.drop(columns_to_drop, axis=1)
```

```
df3
```

	imageID	PercentCover	actual_diameter(um)	count
0	img369.jpg	8.242858	[20.123650981419132, 20.123650981419132, 20.12...	20420
1	img367.jpg	6.759967	[20.123650981419132, 20.123650981419132, 20.12...	16833
2	img365.jpg	7.248508	[20.123650981419132, 20.123650981419132, 20.12...	17869
3	img363.jpg	7.008417	[20.123650981419132, 20.123650981419132, 20.12...	15211
4	img361.jpg	5.341667	[20.123650981419132, 20.123650981419132, 20.12...	18315
...
115	img169.jpg	16.201942	[20.123650981419132, 20.123650981419132, 20.12...	23835
116	img157.jpg	14.854517	[20.123650981419132, 20.123650981419132, 20.12...	26924
117	img171.jpg	20.874367	[20.123650981419132, 20.123650981419132, 20.12...	30246
118	img173.jpg	18.131700	[20.123650981419132, 20.123650981419132, 20.12...	25769
119	img175.jpg	29.483650	[20.123650981419132, 20.123650981419132, 20.12...	28454

120 rows × 4 columns

8. (Optional) exclude erroneously small or large droplets

```
# filtering erroneous droplets - this might not be necessary for actual image analysis - change threshold value as needed
```

```
df3['actual_diameter(um)'] = df3.apply(lambda row: [value for value in row['actual_diameter(um)'] if value > 40]
                                     if len(row['actual_diameter(um)']) > 3 else row['actual_diameter(um)'], axis=1)
```

```
df3['actual_diameter(um)'] = df3.apply(lambda row: [value for value in row['actual_diameter(um)'] if value < 400]
                                     if len(row['actual_diameter(um)']) > 3 else row['actual_diameter(um)'], axis=1)
```

```
df3
```

If you are certain about droplets in certain size ranges were erroneous due to overlapping or merging stains, these lines of code will exclude droplets at certain threshold.

9. Calculate droplet density (droplets cm⁻²)

```
df3['count']=df3['actual_diameter(um)'].str.len()
```

```
# calculate droplet density (droplets/cm2) - change values according to the sample (cropped) area
```

```
df3['droplet density (droplets/cm2)'] = df3['count'].apply(lambda x: x / 215.05)
```

```
df3
```

	imageID	PercentCover	actual_diameter(um)	count	droplet density (droplets/cm2)
0	img369.jpg	8.242858	[20.123650981419132, 20.123650981419132, 20.12...	20420	94.954662
1	img367.jpg	6.759967	[20.123650981419132, 20.123650981419132, 20.12...	16833	78.274820
2	img365.jpg	7.248508	[20.123650981419132, 20.123650981419132, 20.12...	17869	83.092304
3	img363.jpg	7.008417	[20.123650981419132, 20.123650981419132, 20.12...	15211	70.732388
4	img361.jpg	5.341667	[20.123650981419132, 20.123650981419132, 20.12...	18315	85.166240
...
115	img169.jpg	16.201942	[20.123650981419132, 20.123650981419132, 20.12...	23835	110.834690
116	img157.jpg	14.854517	[20.123650981419132, 20.123650981419132, 20.12...	26924	125.198791
117	img171.jpg	20.874367	[20.123650981419132, 20.123650981419132, 20.12...	30246	140.646361
118	img173.jpg	18.131700	[20.123650981419132, 20.123650981419132, 20.12...	25769	119.827947
119	img175.jpg	29.483650	[20.123650981419132, 20.123650981419132, 20.12...	28454	132.313415

120 rows × 5 columns

Alter the value depending on the sample size (cropped area)

```
df3['droplet density (droplets/cm2)'] = df3['count'].apply (lamda x: x  
/ 215.05)
```

10. Calculate the estimated volume of each detected droplet

```
df3['volume(um3)'] = df3['actual_diameter(um)'].apply(lambda x: ((d ** 3 / 6) * np.pi for d in x))
```

```
df3
```

11. Calculate mean diameter, cumulative total volume of detected droplets, and threshold for $Dv_{0.1}$, VMD, and $Dv_{0.9}$

```
df3['mean_diam']=df3['imageID']
df3['sum']=df3['imageID']
df3['10th_vol']=df3['imageID']
df3['50th_vol']=df3['imageID']
df3['90th_vol']=df3['imageID']
```

```
for i in range(0,len(df3)):
    length=len(df3['actual_diameter(um)'][i])
    try:
        df3['mean_diam'][i]=np.mean(df3['actual_diameter(um)'][i])
    except:
        df3['mean_diam'][i]=0
```

C:\Users\User\AppData\Local\Temp\ipykernel_17312\2482018279.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
df3['mean_diam'][i]=np.mean(df3['actual_diameter(um)'][i])

```
for i in range(0,len(df3)):
    length=len(df3['volume(um3)'][i])
    try:
        df3['sum'][i]=np.sum(df3['volume(um3)'][i])
        df3['10th_vol'][i]=PercVol(df3['volume(um3)'][i],0.1)
        df3['50th_vol'][i]=PercVol(df3['volume(um3)'][i],0.5)
        df3['90th_vol'][i]=PercVol(df3['volume(um3)'][i],0.9)
    except:
        df3['sum'][i]=0
        df3['10th_vol'][i]=0
        df3['50th_vol'][i]=0
        df3['90th_vol'][i]=0
```

C:\Users\User\AppData\Local\Temp\ipykernel_17312\2329719623.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

Warning message will appear, but it is okay to ignore.

12. Calculate $Dv_{0.1}$, VMD, and $Dv_{0.9}$

```
: df3['DV0.1'] = (df3['10th_vol']*6/np.pi)**(1/3)
df3['VMD'] = (df3['50th_vol']*6/np.pi)**(1/3)
df3['DV0.9'] = (df3['90th_vol']*6/np.pi)**(1/3)
```

```
: df3
```

13. Calculate spray volume of blue colorant ($\mu\text{l cm}^2$)

```
df3['deposition (ul/cm2)']=0.0045*df3['PercentCover']**1.2789
```

Replace the equation if another standard curve was generated between stain coverage and spray volume of colorant.

14. Cleanse unnecessary data

```
columns_to_drop = ['actual_diameter(um)', 'volume(um3)', 'sum', '10th_vol', '50th_vol', '90th_vol']
```

```
df4=df3.drop(columns=columns_to_drop, axis=1)
```

df4

	imageID	PercentCover	count	droplet density (droplets/cm2)	mean_diam	DV0.1	VMD	DV0.9	deposition (ul/cm2)
0	img369.jpg	8.242858	20420	94.954662	73.235807	66.336405	114.458727	191.174526	0.066801
1	img367.jpg	6.759967	16833	78.274820	73.30281	66.336405	112.849067	188.551806	0.051836
2	img365.jpg	7.248508	17869	83.092304	73.047731	66.950797	115.250531	190.635937	0.056674
3	img363.jpg	7.008417	15211	70.732388	78.55967	69.325145	118.33645	193.404966	0.054284
4	img361.jpg	5.341667	18315	85.166240	65.023227	58.894993	92.083127	150.479324	0.038356
...
115	img169.jpg	16.201942	23835	110.834690	89.147946	84.118203	148.23907	237.268142	0.158536
116	img157.jpg	14.854517	26924	125.198791	82.196733	75.784896	133.645714	219.836542	0.141874
117	img171.jpg	20.874367	30246	140.646361	85.801976	86.217451	165.627076	270.003647	0.219213
118	img173.jpg	18.131700	25769	119.827947	89.412214	85.387354	154.929727	248.42178	0.183075
119	img175.jpg	29.483650	28454	132.313415	95.703947	107.792082	224.869214	369.00826	0.340926

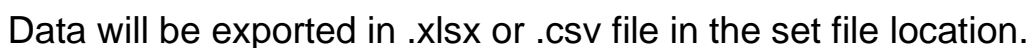
120 rows × 9 columns

15. Export file to .xlsx or .csv file

```
df4.to_excel('C:/Users/User/Downloads/SprayDAT.xlsx', index=False)
```

```
df4.to_csv('C:/Users/User/Downloads/SprayDAT.csv', index=False)
```

Change the **file path/filename.xlsx** (or .csv) of your interest.



- Code blocks when scanned images are arranged in a folder with multiple sub-folder.

```
#This code block is designed to work with a folder full of folders of photos

contours=[]
dpi=600
um2perpix = (1/(dpi/25400))
for folder in os.listdir(directory):
    fol= os.path.join(directory, folder)
    print(fol)
    if os.path.isdir(fol):
        for filename in os.listdir(fol):
            f = os.path.join(fol, filename)
            # checking if it is a file
            if os.path.isfile(f) and f[-3:]=='jpg':
                binary=img2binary(f,154)
                binary2 = binary[2000:4000, 1500:4000]
                total = binary2.shape[0]*binary2.shape[1]
                PC,cnt = binaryStats(binary2)
                #pcarray.append(PC)
                contours.append([directory[-11:], folder, filename, PC, cnt, [round(2*np.sqrt(((cv2.contourArea(i)+1)/np.pi))*um2perpix,4) for i in cnt]])
```