Dynamic Watering Point Localization for Soil Channeling Prevention Using Computer Vision

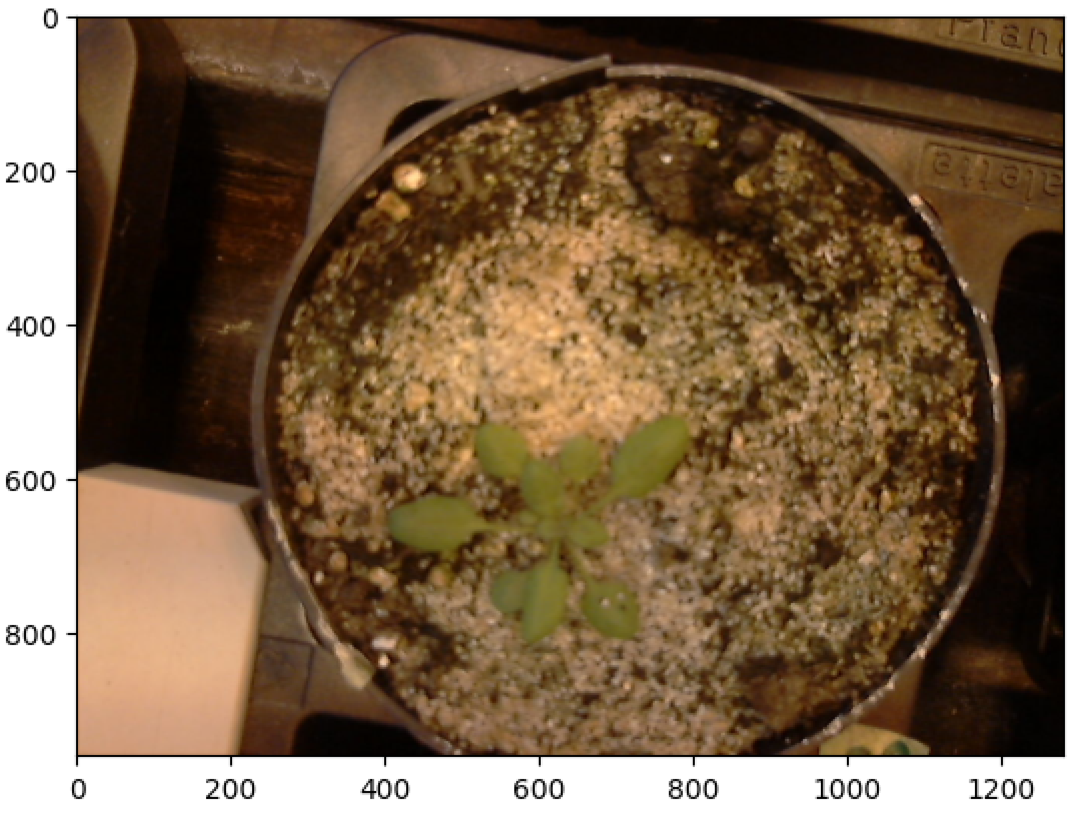
Teng Tian

In [ ]:
```
%matplotlib inline
#import library
from plantcv import plantcv as pcv
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
#import time library to measure the execution time
import time
import math
```

In [ ]:
```
pcv.params.debug="plot"
```

The images used in this work were captured by the on-board camera of Farmbot and were taken in the University Greenhouse.

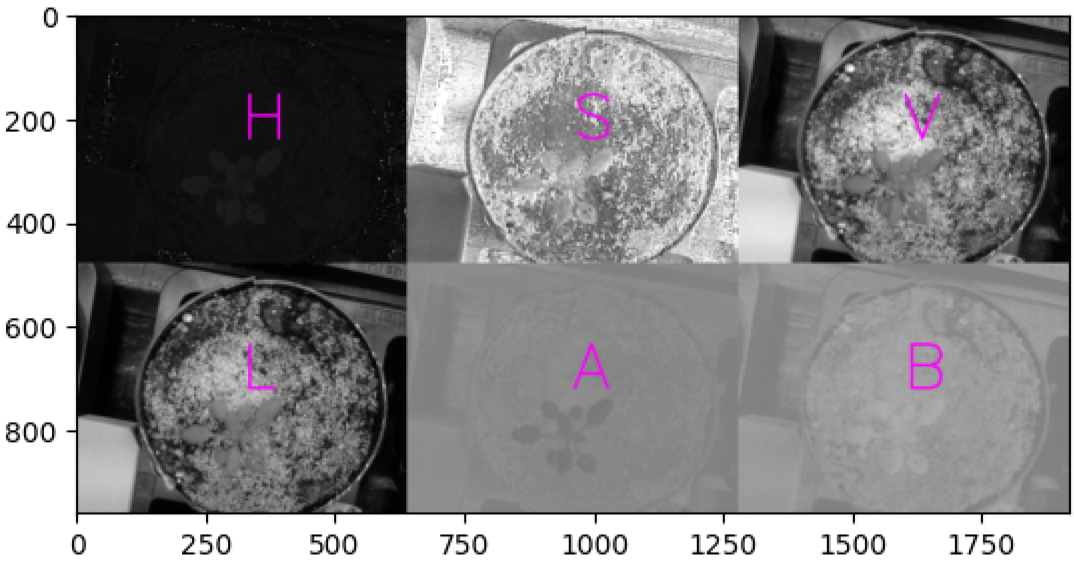
In [ ]:
```
# read an image
img,path,filename=pcv.readimage(filename="../data/CAM_greenhouse/fotos_ideal_condition/Farmbo
```



Step 1: finde the location of the plants

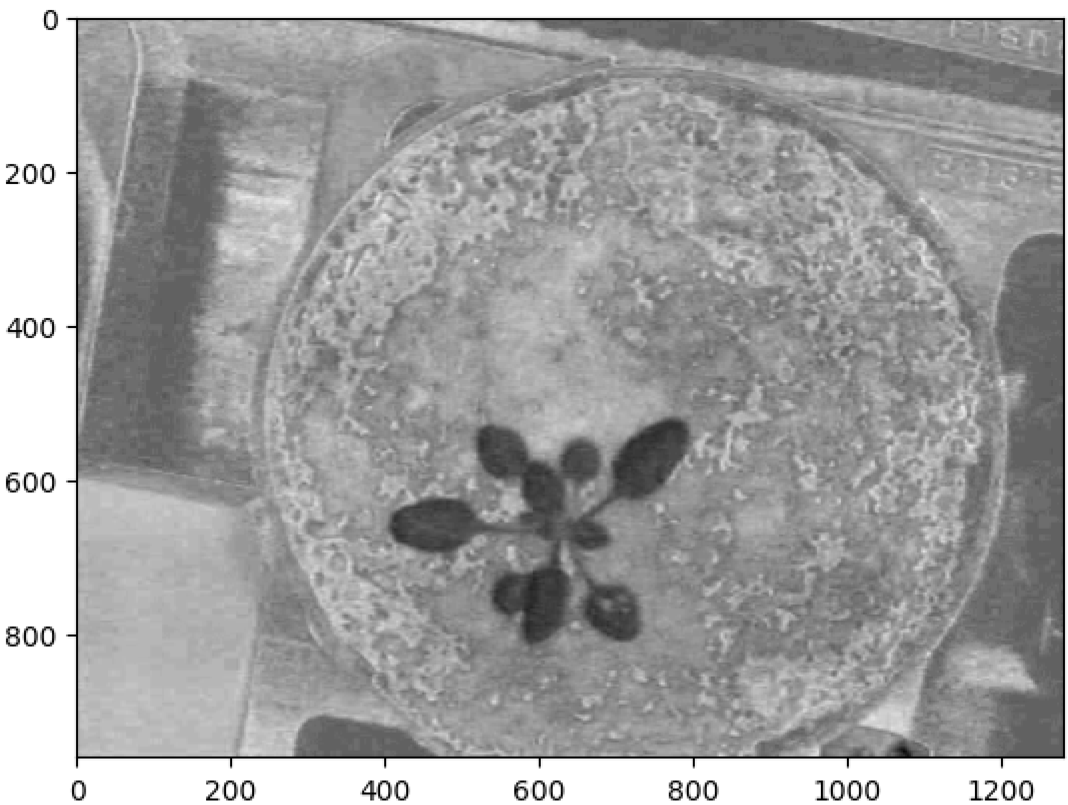
Analyzing the image in different color spaces provides a clear understanding of the image properties.

In [ ]:
```
colorspaces=pcv.visualize.colorspaces(rgb_img=img, original_img=False)
pcv.params.text_size = 8
pcv.params.text_thickness = 10
```



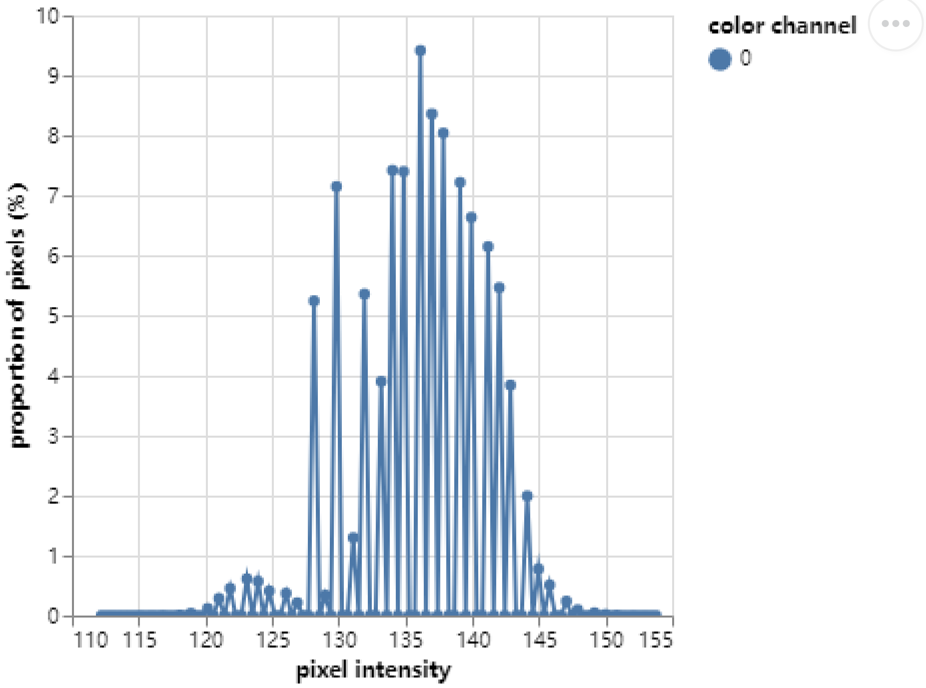
Using channel A will provide the most significant color difference between plants and surrounding objects, as well as the soil beneath them.

In [ ]:
```
# have a close look at the A-channel
img_A=pcv.rgb2gray_lab(rgb_img=img, channel="A")
```



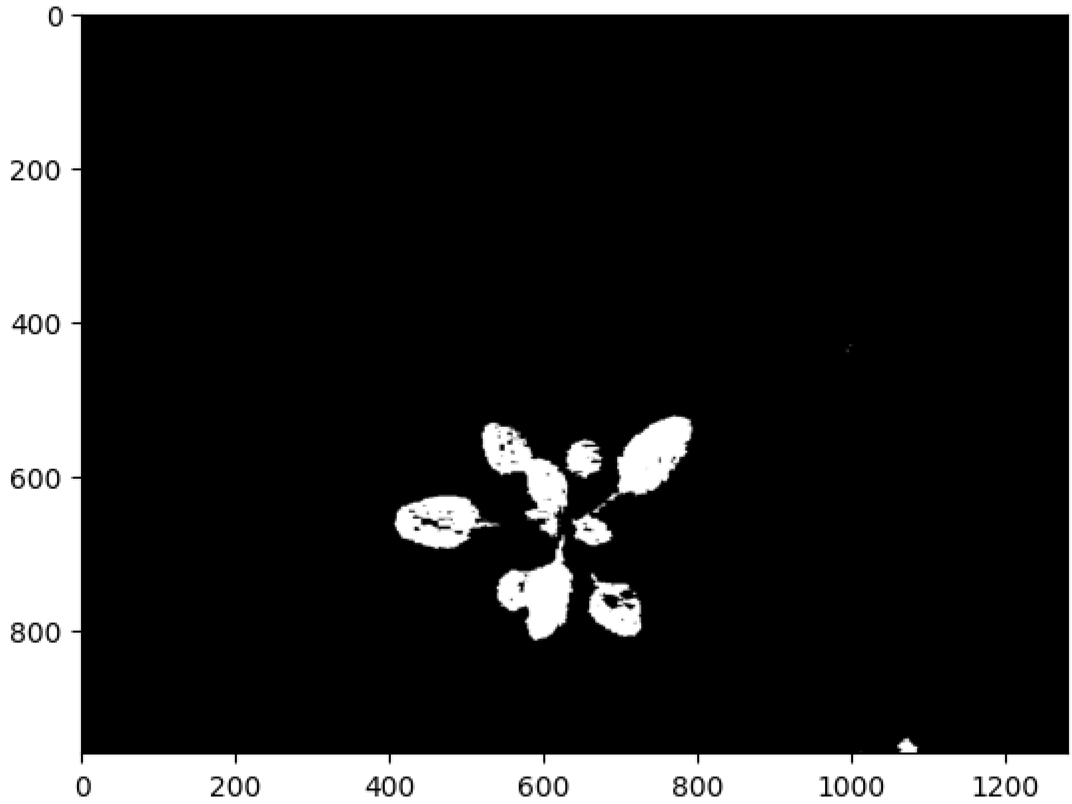
The plants appear to have a darker color. Let's check the histogram to confirm this.

In [ ]:
```
hist = pcv.visualize.histogram(img=img_A)
```



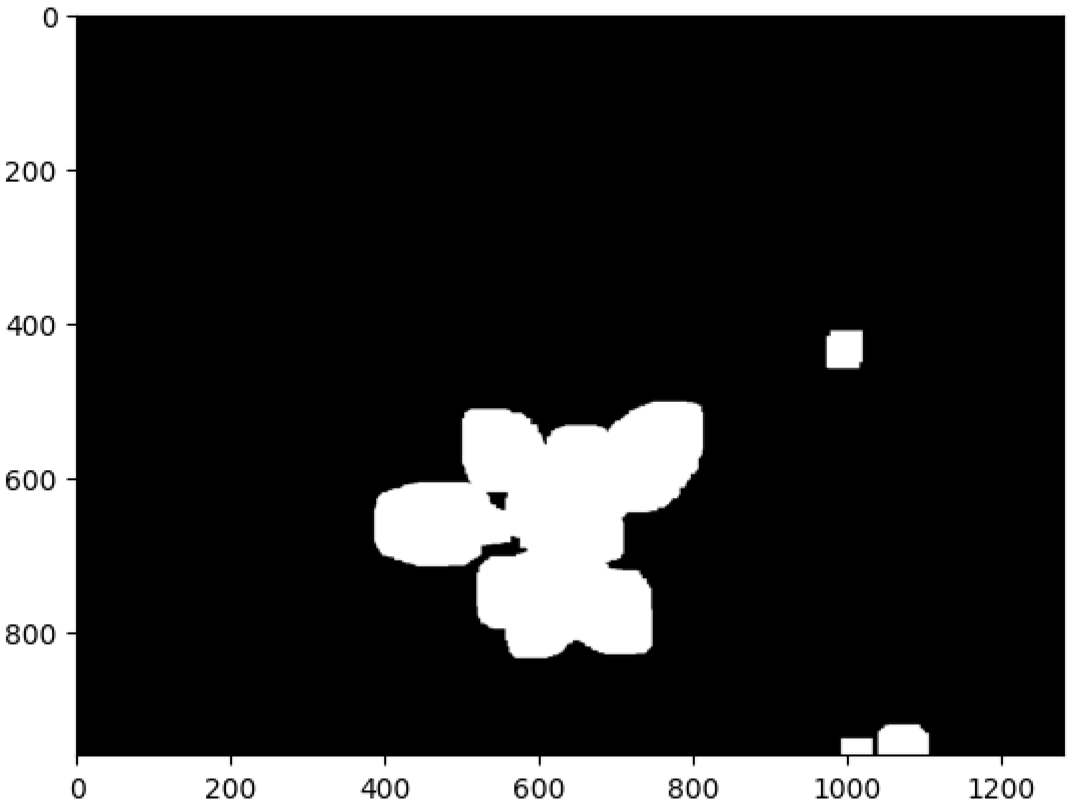
In [ ]:

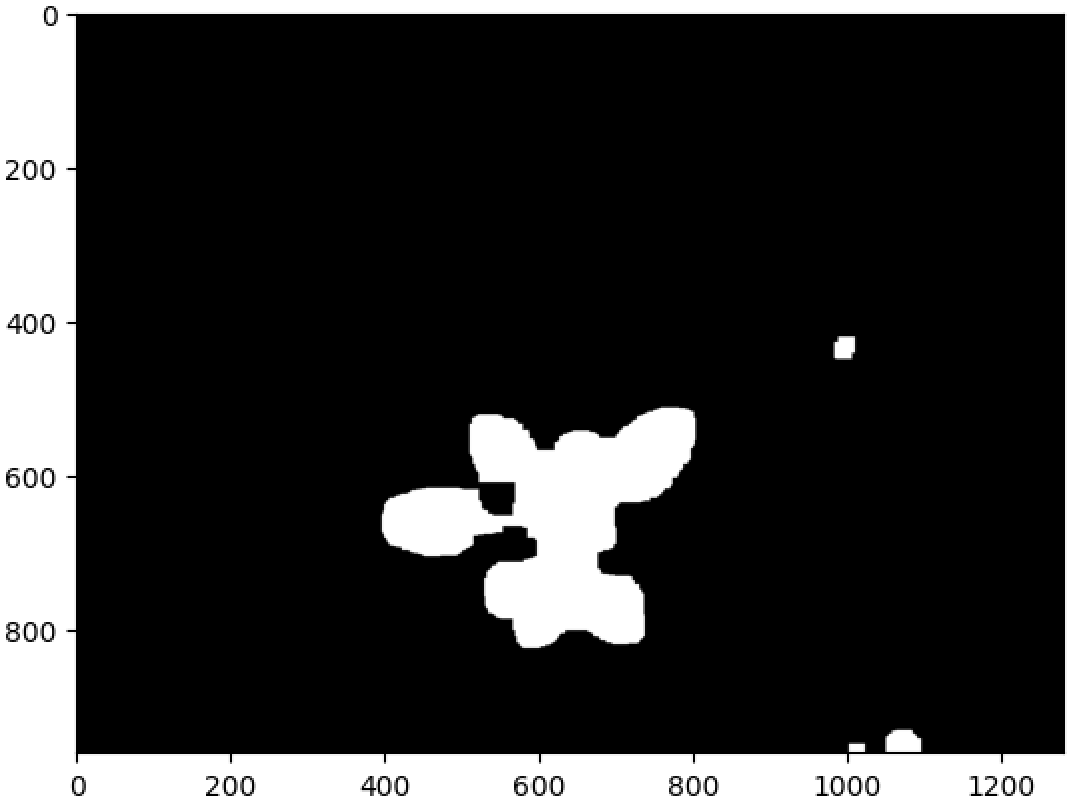
```
# remove all the pixels, which has a value greater than 125
# in this way we habe now a binary mask
img_A_thresh = pcv.threshold.binary(gray_img = img_A, threshold = 125, object_type = 'dark')
```



In [ ]:

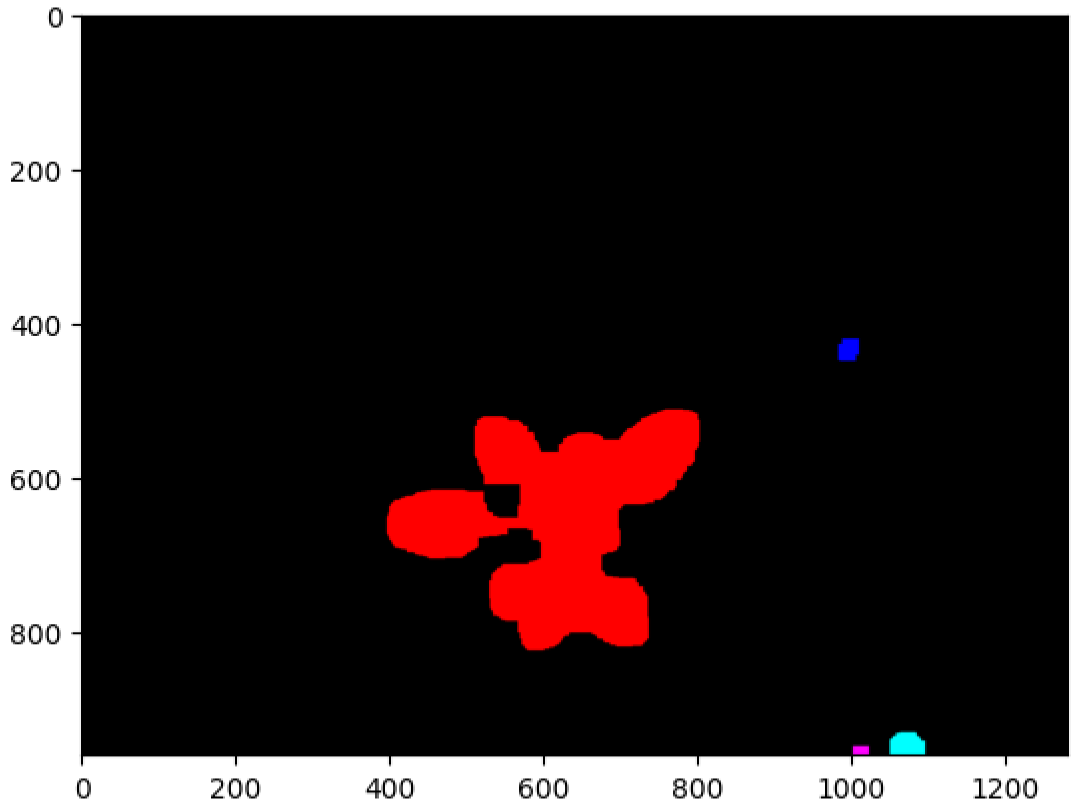
```
# using closing methode to have a better image for region finding methode
mask_dilated = pcv.dilate(gray_img = img_A_thresh, ksize = 3, i = 20)
mask_erode = pcv.erode(gray_img = mask_dilated, ksize = 5, i =5)
mask = mask_erode
```

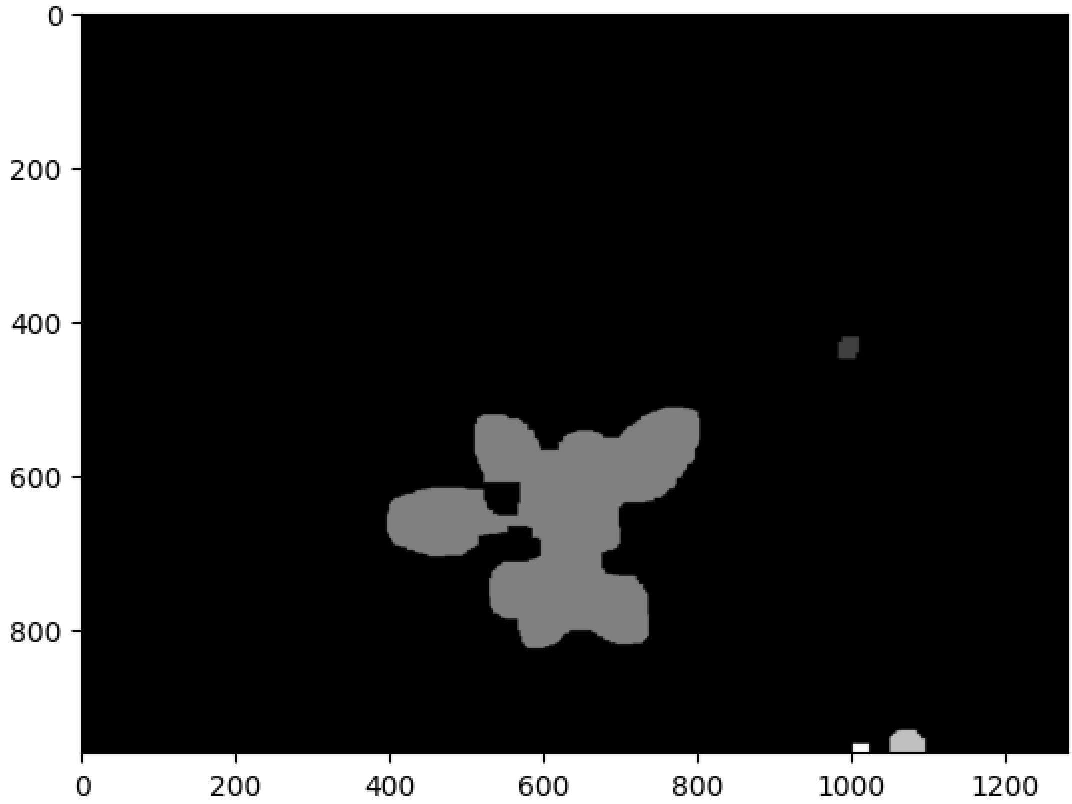




In [ ]:

```
# labeled the regions on the mask image
labeled_mask, num_mask = pcv.create_labels(mask=mask)
pcv.plot_image(labeled_mask)
print('total', num_mask, 'region(s) found!')
```



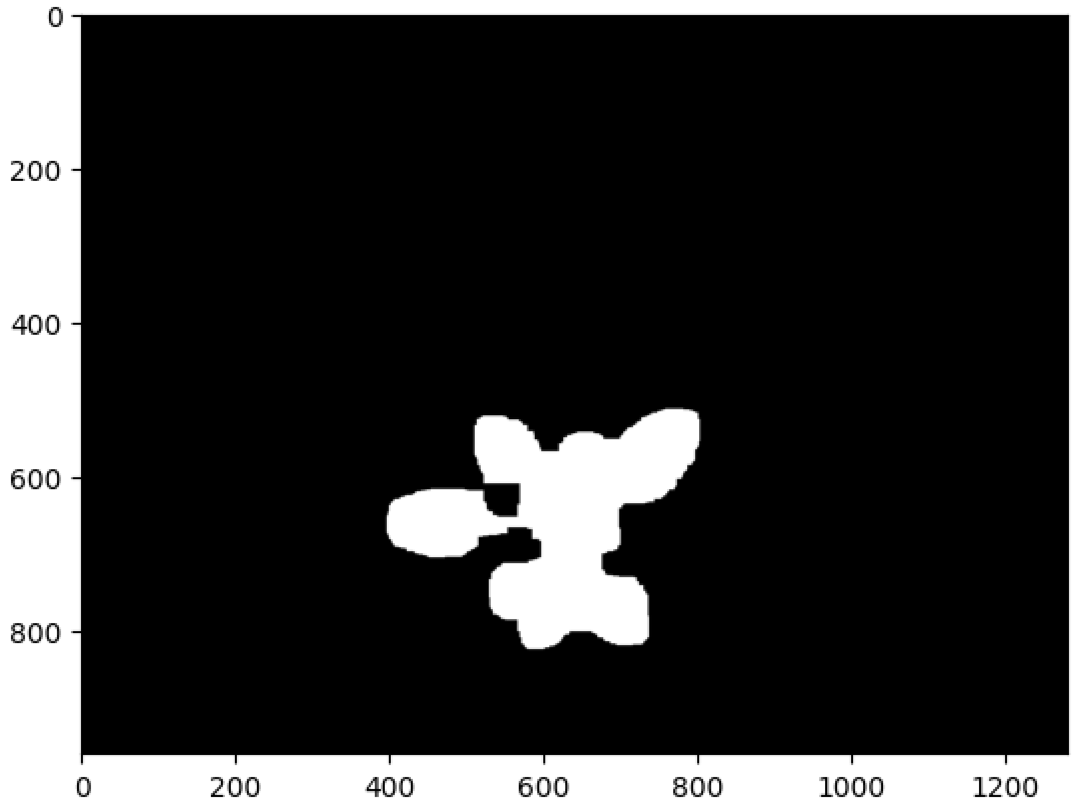


total 4 region(s) found!

In [ ]:

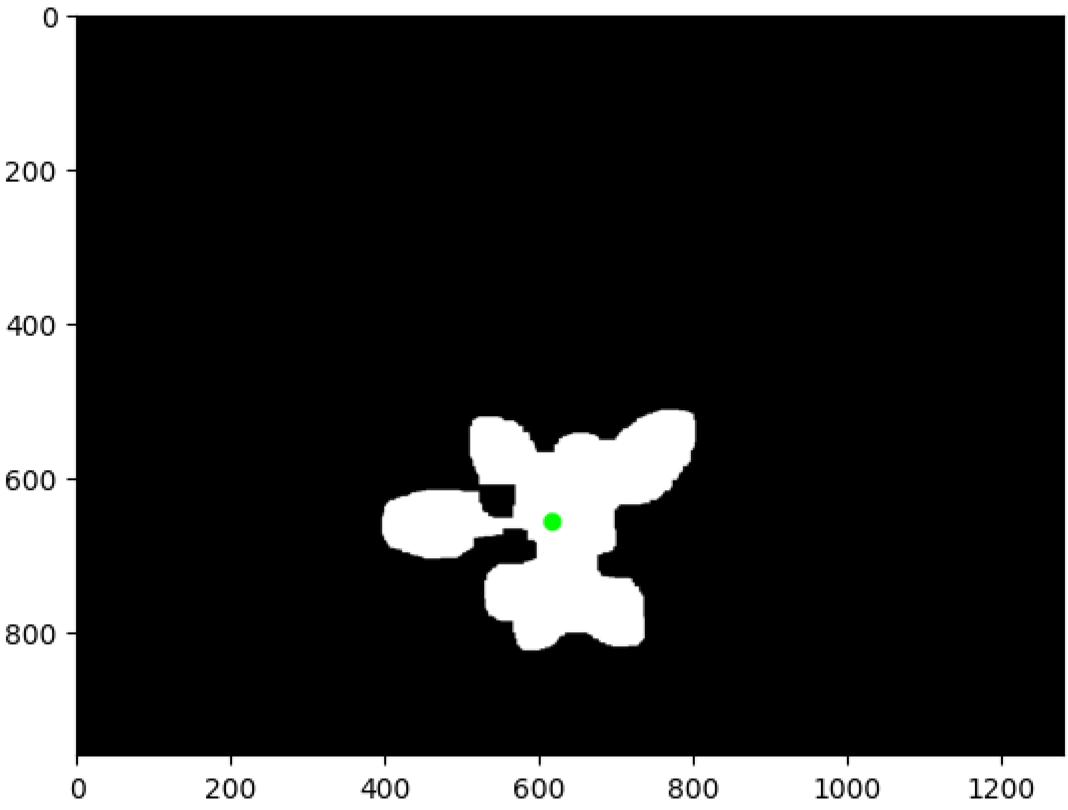
```
# just keep the biggst region on the mask
cnt_prev = 0
for region_id in range(1,num_mask+1):
    mask_region_cnt = cv.inRange(labeled_mask, region_id,region_id)
    cnt = cv.countNonZero(mask_region_cnt)
    print(cnt)
    if cnt<cnt_prev:
        mask_cop = cv.inRange(labeled_mask, region_id-1,region_id-1)
        break
    else: cnt_prev = cnt
pcv.plot_image(mask_cop)
```

714
63615
1181



In [ ]:
```
# calculation the center of mass of the region
# this will locate the plant
contours, hierarchy = cv.findContours(mask_cop,1,2)
cnt = contours[0]
M = cv.moments(cnt)
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
print(cx,cy)
mask_RGB=cv.cvtColor(mask_cop,cv.COLOR_GRAY2BGR)
cv.circle(mask_RGB,(cx,cy),2,(0,255,0),20)
pcv.plot_image(mask_RGB)
```

617 658



Now we have a fairly accurate mask that shows where the plants are located.

Step 2: Determine the inner area of the flower pot

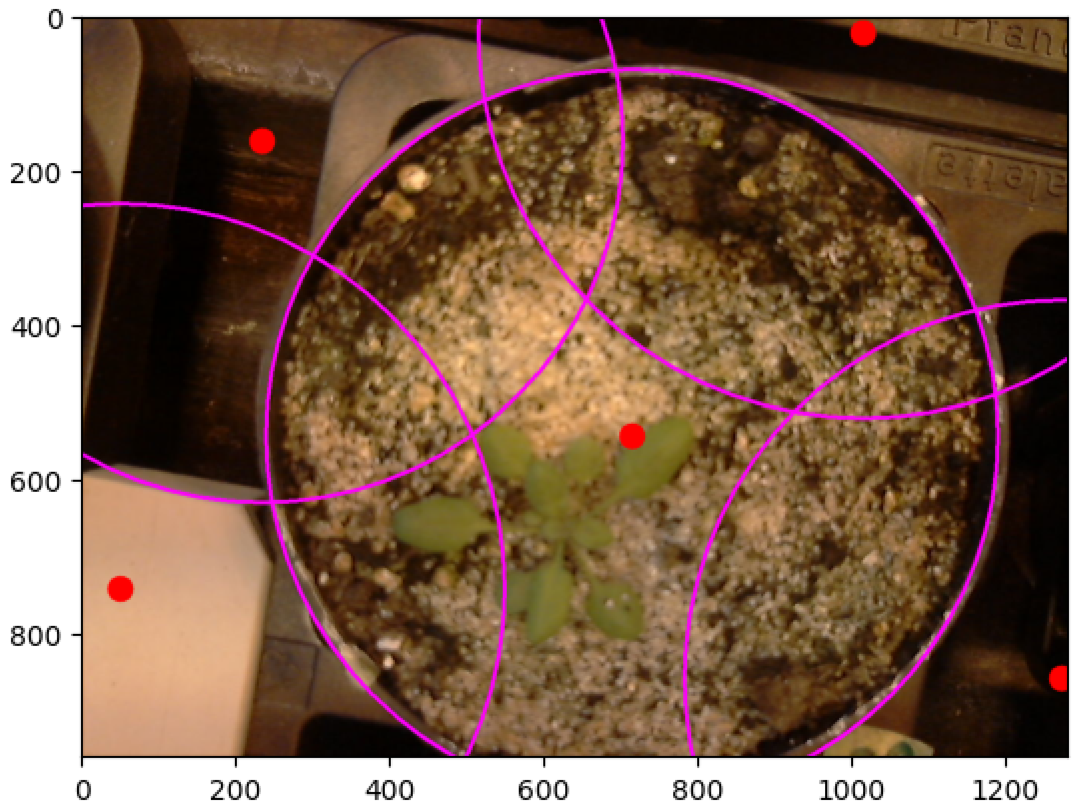
The hough-circle method is being used to detect flower pots in the image. As a prerequisite, it is necessary to determine the minimum distance between two pots and the pot radius in pixels. This refers to these parameters:

- mindDist
- minRadius
- maxRadius

In [ ]:
```
# using hough-circle to dectect flower pot(s) in the image
# as a precondition, we need to know the smallest distance between two pots and the radius of
'''
cv.HoughCircles (image, circles, method, dp, minDist, param1 = 100, param2 = 100, minRadius =

Parameters
image   8-bit, single-channel, grayscale input image.
circles output vector of found circles(cv.CV_32FC3 type). Each vector is encoded as a 3-elemen
method  detection method(see cv.HoughModes). Currently, the only implemented method is HOUGH_G
dp      inverse ratio of the accumulator resolution to the image resolution. For example, if d
minDist minimum distance between the centers of the detected circles. If the parameter is too
param1  first method-specific parameter. In case of HOUGH_GRADIENT , it is the higher threshol
param2  second method-specific parameter. In case of HOUGH_GRADIENT , it is the accumulator th
minRadius       minimum circle radius.
maxRadius       maximum circle radius.
'''
img_HoughCircles = img.copy()
start_time = time.time()
circles_list = []
img_GRAY=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
img_mB = cv.medianBlur(img_GRAY,5)
circles= cv.HoughCircles(image=img_mB,method=cv.HOUGH_GRADIENT,dp=2,minDist=600,param1=40,par
circles = np.uint16(np.around(circles))
circles_list.append(circles)
```

```python
for i in circles[0,:]:
    # draw the outer circle
    cv.circle(img_HoughCircles,(i[0],i[1]),i[2],(255,0,255),4)
    # draw the center of the circle
    cv.circle(img_HoughCircles,(i[0],i[1]),2,(0,0,255),30)
pcv.plot_image(img_HoughCircles)
end_time = time.time()
print('Execution time:', round(end_time - start_time, 2), 'seconds')
```



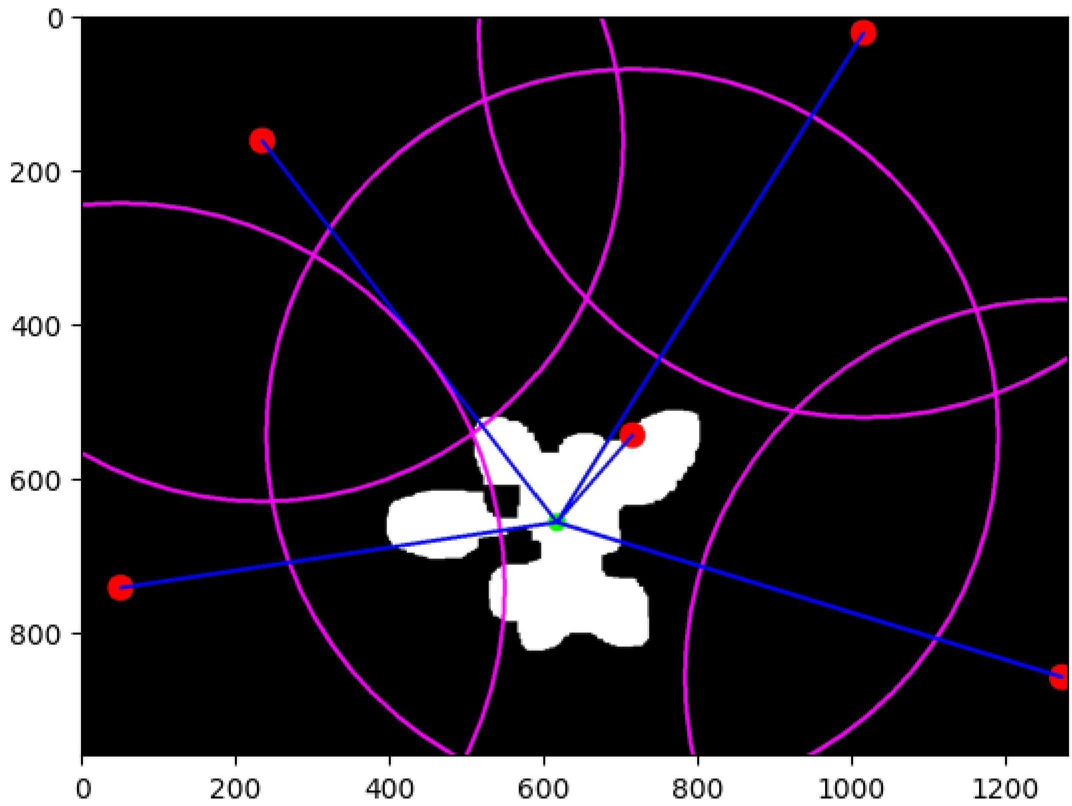

```
Execution time: 3.6 seconds
```

The Hough Circle algorithm may find more than one circle in the image. The circle we are searching for is among them, and this is the most crucial aspect. The next step is to select the filter to exclude incorrect circles.

In [ ]:

```python
# we take only the circle, whose center is closest to center of the plant
cnt = 1
index = 0
distance_prev = 0
for i in circles[0,:]:
    # draw the outer circle
    cv.circle(mask_RGB,(i[0],i[1]),i[2],(255,0,255),4)
    # draw the center of the circle
    cv.circle(mask_RGB,(i[0],i[1]),2,(0,0,255),30)
    # draw a line from the center of the circle to center of mass
    cv.line(mask_RGB,(i[0],i[1]), (cx,cy),(255,0,0),4)
    distance = (int)(math.sqrt((cx-i[0])**2+(cy-i[1])**2))
    if distance<distance_prev:
        index = cnt - 1
    print('Distance to ',cnt,'. circle =', distance)
    cnt=cnt+1
print('The',index+1, 'circle is the wanted circle')
pot_x = circles[0][index][0]
pot_y = circles[0][index][1]
pot_radius = circles[0][index][2]
print('Position of the flowerpot is', 'x=', pot_x,'y=',pot_y,'radius=',pot_radius)
pcv.plot_image(mask_RGB)
```

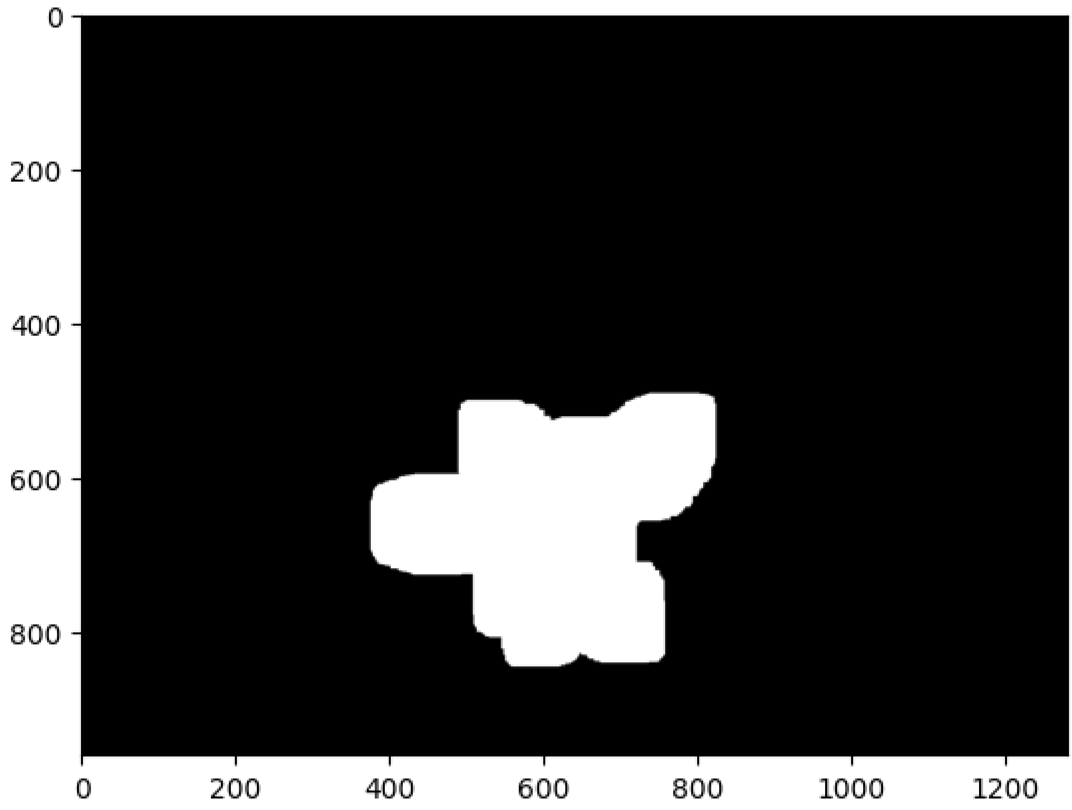
```
Distance to  1 . circle = 149
Distance to  2 . circle = 625
Distance to  3 . circle = 749
Distance to  4 . circle = 686
Distance to  5 . circle = 572
The 1 circle is the wanted circle
Position of the flowerpot is x= 715 y= 545 radius= 475
```



Step 3: Setting watering points

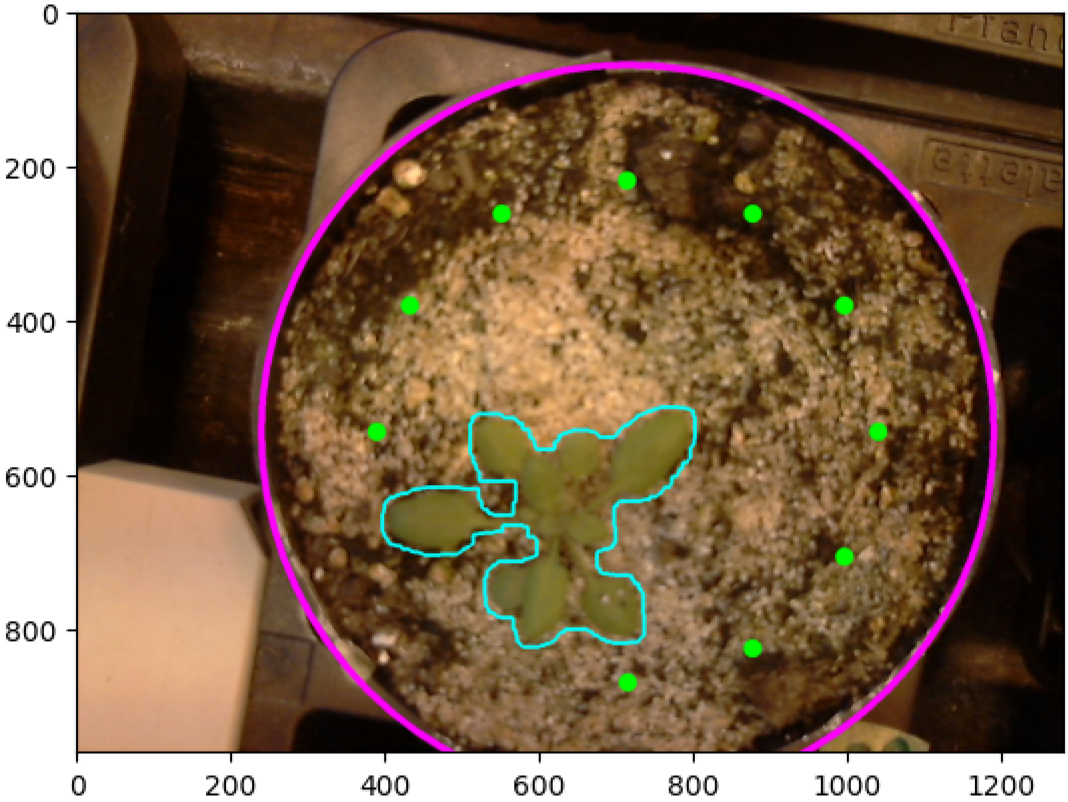
Now we know the exact location of the plants and the pot. The area in the pot can be divided for dynamic watering. Yesterday, I utilized a straightforward method. An edge area was set to prevent water from flowing out of the pot and to prevent soil channels from forming at the edge of the pot. The pot is divided into 12 parts and checked for plant leaves. If there are no leaves, a watering point is set in the area.

In [ ]:
```
# let us define the watering point.
# It cannot be at the edge area, and either on top of the plant
edge_area = 150 # 100 Pixel to the edge will not be watered
# create watering point
angel = 30 # must be 15, 30, 45, 60, 90
# the previous mask will be enlarged, so that there will be a safty zone, that we will not wa
mask_with_saftyzone = pcv.dilate(gray_img = mask_cop, ksize = 15, i = 3)
watering_point_list = []
for i in range(0, (int)(360/angel)):
    x_watering_point = (int)(math.cos(i*angel/180*math.pi)*(pot_radius - edge_area)+pot_x)
    y_watering_point = (int)(math.sin(i*angel/180*math.pi)*(pot_radius - edge_area)+pot_y)
    if mask_with_saftyzone[y_watering_point, x_watering_point] != 255:
        watering_point_list.append((x_watering_point, y_watering_point))
    #watering_point_list.append((x_watering_point, y_watering_point))
```



In [ ]:
```
# lets draw everything on image
img_out = img.copy()
cv.circle(img_out,(pot_x,pot_y),pot_radius,(255,0,255),8)
for i in watering_point_list:
    cv.circle(img_out,i,2,(0,255,0),20)
    print('prossible watering point:',i)
cv.drawContours(img_out, contours, 0, (255,255,0), 3)
pcv.plot_image(img_out)
```

```
prossible watering point: (1040, 545)
prossible watering point: (996, 707)
prossible watering point: (877, 826)
prossible watering point: (715, 870)
prossible watering point: (390, 545)
prossible watering point: (433, 382)
prossible watering point: (552, 263)
prossible watering point: (714, 220)
prossible watering point: (877, 263)
prossible watering point: (996, 382)
```


0
200
400
600
800
0
200
400
600
800
1000
1200