# Simplifying PlantCV workflows with multiple objects

Haley Schuhl[a], J. David Peery[a, b], Jorge Gutierrez[a], Malia A. Gehan[a], and Noah Fahlgren[a]

[a]Donald Danforth Plant Science Center, St. Louis, MO, USA
[b]University of North Carolina at Chapel Hill, Chapel Hill, NC, USA

## ABSTRACT

Imaging of plants using multi-camera arrays in high-density growth environments is a strategy for affordable high-throughput phenotyping. In multi-camera systems, simultaneous imaging of hundreds to thousands of plants eliminates the time delay in measurements between plants seen in plant-to-camera or camera-to-plant systems, which allows for the analysis of plant growth, development, and environmental responses at a high temporal resolution. On the other hand, high plant density, camera-to-camera variation, and other trade-offs increase the complexity of data analysis. Here we present two recent updates to the PlantCV image analysis package to improve usability when working with multi-plant datasets. First, we introduce a method to automate detection of plants organized in a grid layout, reducing the need to make separate workflows for each camera in a multi-camera system. Second, we reduced the number of input and output parameters for functions handling the shape and location of plants and introduce automatic iteration over multiple objects of interest (e.g. plants), reducing the level of programming needed to build workflows.

**Keywords:** NAPPN 2023, plant phenotyping, computer vision, image analysis, time-lapse

## 1. INTRODUCTION

In high-throughput phenotyping, imaging paired with computer vision is used to non-destructively measure plant features and responses to the environment over a period of growth. PlantCV* is an open-source, Python-based software package for analyzing images in plant phenotyping experiments.[1] One of the goals of the PlantCV project is to make image analysis accessible for both computer programmers and biologists alike, and while coding experience to use PlantCV is relatively minimal, areas where usability can be improved exist. Improvements to PlantCV (and other community-based software) are opportunities for community involvement, including training and mentoring of students and early career researchers.[2]

In a common plant phenotyping setting, plants in growth chambers are imaged to simultaneously collect data for hundreds or thousands of plants.[3–5] In this configuration, plants are arranged in a grid and each camera observes multiple plants from an overhead perspective. Image segmentation is used to separate the plant pixels from the background, but to measure individual plant phenotypes, each plant needs to be labeled uniquely. When plants are grown in a structured grid layout, a simple approach is to set a region of interest (ROI) to denote the location and approximate area of each pot. In PlantCV, multiple regions of interest in a grid layout can be created using the *plantcv.roi.multi* function. The *plantcv.roi.multi* function requires several inputs that the user must manually determine from a representative image. The required input parameters are the following: *coord*, the pixel coordinate of the center of the plant in the top-left corner; *radius*, the desired radius of the circular ROIs; *spacing*, the number of pixels between plant centers in the horizontal and vertical directions; and *nrows* and *ncols*, the number of rows and columns in the grid (Figure 1). A user can potentially reuse the same workflow settings across multiple images if the camera field-of-view remains the same across images in the dataset. However, when there are multiple cameras or any change in position between the trays and cameras, the input variables will likely need to be re-parameterized, which can be time consuming. Parameterization of individual cameras is a common use case with Raspberry Pi equipped growth chambers used at the Donald

Further author information: (Send correspondence to Noah Fahlgren)
Noah Fahlgren: E-mail: nfahlgren@danforthcenter.org
J. David Peery: ORCID: 0000-00002-0192-3427
*https://plantcv.danforthcenter.org

Danforth Plant Science Center.[3] Additionally, PlantCV v3 supports the recording of multiple sets of phenotype measurements for multi-plant images but requires users to iterate over each ROI using a Python *for* loop in their workflow, which requires more advanced understanding of the Python programming language. Here we describe new features of PlantCV v4 that will streamline the analysis workflow development process.

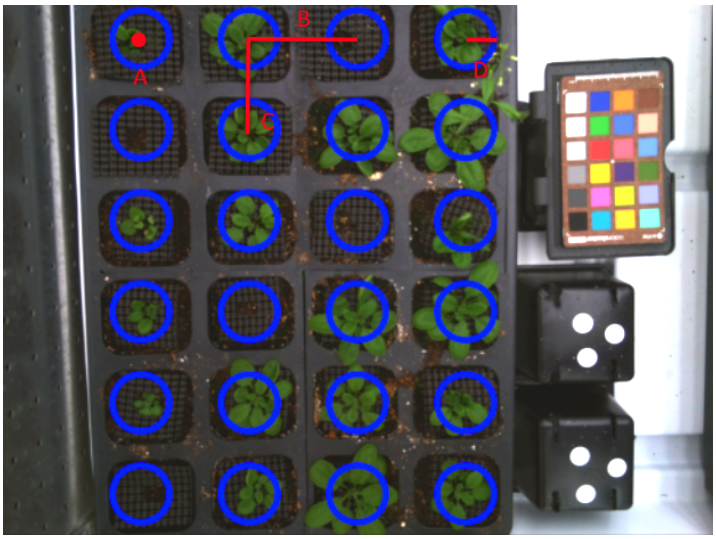


Figure 1: Input image and parameters for *plantcv.roi.multi*. Blue circles are the regions of interest (ROIs). (A) marks the location of the *coord* parameter, (B) and (C) are the lengths of the *spacing* parameters in the horizontal and vertical directions, respectively, and (D) marks the *radius*.

## 2. MATERIALS AND METHODS

Functions were added to or modified from PlantCV. All stable and development versions of PlantCV can be cloned from the PlantCV GitHub repository[†]. The changes and additions for this project were added to PlantCV through pull requests, which were merged into the 4.x development branch after passing automated testing and being reviewed by other PlantCV contributors. Documentation pages detailing how to use the new and updated functions were added to the PlantCV documentation[‡].

Images of *Arabidopsis thaliana* (L.) Heynh. plants were acquired hourly from Zeitgeber time (ZT) 00:05 to 15:05 over several weeks using a Raspberry Pi 3 model B+ computer[§] and 8 megapixel Raspberry Pi Camera Module 2[¶]. All *A. thaliana* plants were of the Columbia (Col) ecotype and grown on Pro-Mix FPX soil in Conviron MTPS-20 growth chambers in diurnal conditions with 16 hours light and 8 hours dark at 22°C and 18°C, respectively, with $200 \mu mol/m^2/s$ light.

PlantCV functions were developed and tested using Python v3.9.12, OpenCV v4.5.54, NumPy v1.22.45, scikit-image v0.19.3, scikit-learn v1.1.16 and Jupyter Notebook v8.4.07. Documentation pages were added and updated using mkdocs v1.3.0.[6–10]

## 3. RESULTS AND DISCUSSION

### 3.1 Automatic plant grid layout detection

Given the regular layout of plants grown as in Figure 1, we aimed to design an algorithm that can automatically detect the grid layout parameters typically input by a user using the *plantcv.roi.multi* function. We hypothesized

[†] https://github.com/danforthcenter/plantcv
[‡] https://plantcv.readthedocs.io/en/4.x/
[§] https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus
[¶] https://www.raspberrypi.com/products/camera-module-v2

that the positions of the columns and rows of the grid of plants could be estimated by clustering the horizontal and vertical positions, respectively, using a Gaussian Mixture Model[11] parameterized with the correct number of centers (the number of rows or columns). First, the input image is segmented into a binary image that labels plant pixels white and background black (Figure 2). Next, the OpenCV function *findContours* is used to identify the contours of each object in the binary image and the center of mass of each object is calculated. A Gaussian Mixture Model is used to calculate the likely x and y positions of each column and row, respectively. The resulting positions are used to identify the coordinate of the center of the top-left pot and the average horizontal and vertical spacing between pots to succinctly and evenly represent the grid. This functionality is available in the PlantCV function *plantcv.roi.auto_grid*, which returns a grid of circular ROIs, like *plantcv.roi.multi*, but only requires an input binary mask and the number of rows and columns.

*plantcv.roi.auto_grid* is robust to missing plants and the presence of background noise. When there are missing plants in the grid (Figure 2), the function still identifies an ROI at the location of the missing plant as long as at least one plant is found in each column and row of the tray. Plants are usually missing because they died part way through an experiment or because their seeds never germinated. While no measurements can be associated with undetected plants, by assigning an ROI to the empty pots, PlantCV will maintain metadata records for these pots.

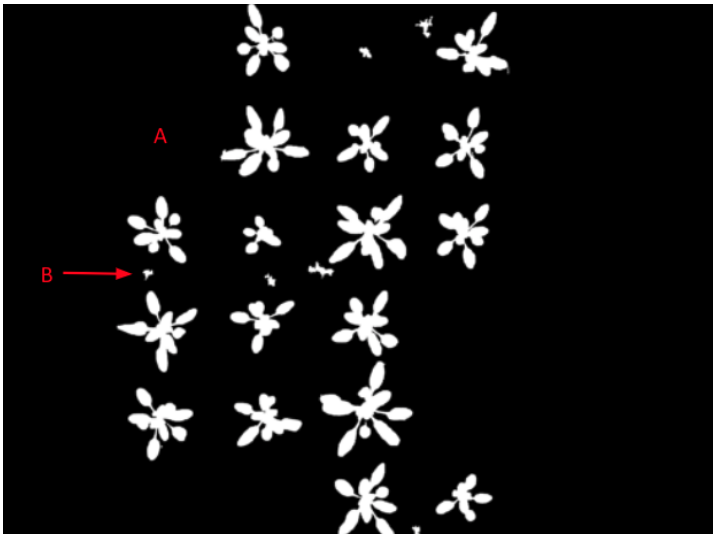


Figure 2: A binary mask with (A) missing plants and (B) objects that are not part of a plant.

The streamlined *plantcv.roi.auto_grid* function was tested on images corresponding to several different trays of *A. thaliana*. In the experiment, up to 544 images were taken over the lifetime of a single tray of plants, and 24 trays of plants were imaged. Since the trays are in slightly different locations in images from each camera, using the existing *plantcv.roi.multi* would require the user to configure a separate workflow for each camera. Instead, we were able to design a single workflow containing the *plantcv.roi.auto_grid* step to analyze images from the entire experiment (Figure 3). The PlantCV workflow was able to identify the grid layout correctly in all images despite variation between images. Some of the images also had missing plants, disjoint leaves, or noise remaining in the binary masks used for clustering, but identification of the grid layout was still correct.

### 3.2 Automating the use of multiple ROIs

In addition to ROIs, PlantCV uses OpenCV contours to describe the structure of objects (e.g. plants, seeds, leaves, etc.) in an image. OpenCV contours are comprised of two data structures: contours are the coordinates of the outlines of objects, and hierarchy is the relationship between individual contours. In previous versions of PlantCV, contours and hierarchy are frequency both inputs and outputs to functions that create ROIs, filter out noise, or measure plants and other objects of interest.[1] To reduce this complexity in PlantCV workflows, we

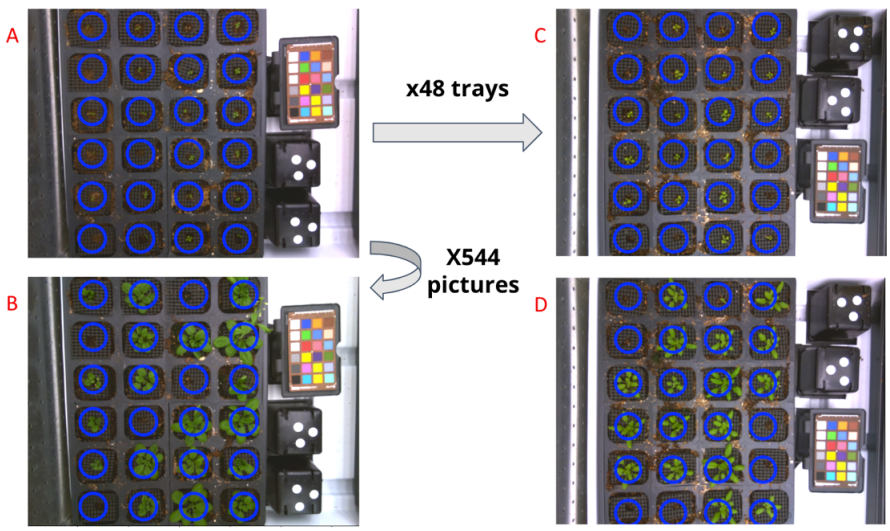


Figure 3: Output images from *roi.auto_grid* on several different images. Images come from two different trays: (A) and (B) are from one tray and (C) and (D) are from another. Images from the same tray were taken at different time points.

added a new data structure, *Objects*, to PlantCV v4 that encapsulates one or more pairs of OpenCV contours and hierarchy, which results in half as many input and/or output parameters for ten or more functions.

In a multi-plant imaging context as seen in Figure 1, tools like *plantcv.roi.auto_grid* return multiple ROIs and measurements need to be done on contours within each ROI individually. In PlantCV v3, a user would need to write a Python *for* loop to iterate over both the OpenCV contours and hierarchy and run a series of PlantCV functions within the loop to get individual plant measurements.[1] In PlantCV v4, we revised functions that use the new *Objects* data structure to automatically iterate over multiple ROIs or contour-hierarchy pairs. Together with new tools such as *plantcv.roi.auto_grid*, these new features reduce the programming complexity of PlantCV workflows, which lowers the barriers of use for all users.

## DATA AVAILABILITY STATEMENT

PlantCV is an open-source image analysis software package targeted for plant phenotyping. PlantCV provides a common programming and documentation interface to a collection of image analysis techniques that are integrated from a variety of source packages and algorithms. PlantCV utilizes a modular architecture that enables flexibility in the design of analysis workflows and rapid assimilation and integration of new methods. For more information about the project, links to recorded presentations, datasets, and publications using PlantCV, please visit our homepage: `https://plantcv.danforthcenter.org`. PlantCV is also available for installation through the Python Package Index (`https://pypi.org`) and Conda-Forge.[12]

## ACKNOWLEDGMENTS

This work was supported by grants from the National Science Foundation (1921724, 2120153) and the United States Department of Agriculture National Institute Food and Agriculture (2019-67021-29926, 2022-67021-36467).

## REFERENCES

[1] Gehan, M. A., Fahlgren, N., Abbasi, A., Berry, J. C., Callen, S. T., Chavez, L., Doust, A. N., Feldman, M. J., Gilbert, K. B., Hodge, J. G., Hoyer, J. S., Lin, A., Liu, S., Lizárraga, C., Lorence, A., Miller, M., Platon, E., Tessman, M., and Sax, T., "PlantCV v2: Image analysis software for high-throughput plant phenotyping," *PeerJ* **5**, e4088 (Dec. 2017).

[2] Goble, C., "Better software, better research," *IEEE Internet Comput.* **18**, 4–8 (Sept. 2014).

[3] Tovar, J. C., Hoyer, J. S., Lin, A., Tielking, A., Callen, S. T., Elizabeth Castillo, S., Miller, M., Tessman, M., Fahlgren, N., Carrington, J. C., Nusinow, D. A., and Gehan, M. A., "Raspberry pi-powered imaging for plant phenotyping," *Appl. Plant Sci.* **6**, e1031 (Mar. 2018).

[4] Taghavi Namin, S., Esmaeilzadeh, M., Najafi, M., Brown, T. B., and Borevitz, J. O., "Deep phenotyping: deep learning for temporal phenotype/genotype classification," *Plant Methods* **14**, 66 (Aug. 2018).

[5] Gutierrez Ortega, J. A., Castillo, S. E., Gehan, M., and Fahlgren, N., "Segmentation of overlapping plants in multi-plant image time series," (Oct. 2021).

[6] Bradski, G., "The OpenCV library," *Dr. Dobb's J.* (2000).

[7] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E., "Array programming with NumPy," *Nature* **585**, 357–362 (Sept. 2020).

[8] van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., and scikit-image contributors, "scikit-image: image processing in python," *PeerJ* **2**, e453 (June 2014).

[9] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É., "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.* **12**(Oct), 2825–2830 (2011).

[10] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., and Jupyter Development Team, "Jupyter notebooks – a publishing format for reproducible computational workflows," in [*Positioning and Power in Academic Publishing: Players, Agents and Agendas: Proceedings of the 20th International Conference on Electronic Publishing*], Loizides, F. and Schmidt, B., eds., 87–90, IOS Press, Amsterdam (2016).

[11] Murtagh, F. and Contreras, P., "Algorithms for hierarchical clustering: an overview," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2**, 86–97 (Jan. 2012).

[12] conda-forge community, "The conda-forge Project: Community-based Software Distribution Built on the conda Package Format and Ecosystem," (July 2015).