

1 **PlantCV v2.0: Image analysis software for high-throughput plant phenotyping**

2 Malia A Gehan<sup>\*, 1</sup>, Noah Fahlgren<sup>\*, 1</sup>, Arash Abbasi<sup>1</sup>, Jeffrey C Berry<sup>1</sup>, Steven T Callen<sup>1, 2</sup>,

3 Leonardo Chavez<sup>1</sup>, Andrew N Doust<sup>3</sup>, Max J Feldman<sup>1</sup>, Kerrigan B Gilbert<sup>1</sup>, John G Hodge<sup>3</sup>, J

4 Steen Hoyer<sup>1, 4</sup>, Andy Lin<sup>1, 5</sup>, Suxing Liu<sup>6, 7</sup>, César Lizárraga<sup>1, 8</sup>, Argelia Lorence<sup>9</sup>, Michael

5 Miller<sup>1, 10</sup>, Eric Platon<sup>11</sup>, Monica Tessman<sup>1, 2</sup>, Tony Sax<sup>12</sup>

6 <sup>1</sup> Donald Danforth Plant Science Center, St. Louis, Missouri, United States of America

7 <sup>2</sup> Current Address: Monsanto Company, St. Louis, Missouri, United States of America

8 <sup>3</sup> Department of Plant Biology, Ecology, and Evolution, Oklahoma State University, Stillwater,

9 Oklahoma, United States of America

10 <sup>4</sup> Computational and Systems Biology Program, Washington University in St. Louis, St. Louis,

11 Missouri, United States of America

12 <sup>5</sup> Current Address: Unidev, St. Louis, Missouri, United States of America

13 <sup>6</sup> Arkansas Biosciences Institute, Arkansas State University, Jonesboro, Arkansas, United States

14 of America

15 <sup>7</sup> Current Address: Department of Plant Biology, University of Georgia, Athens, Georgia, United

16 States of America

17 <sup>8</sup> Current Address: CiBO Technologies, Cambridge, Massachusetts, United States of America

18 <sup>9</sup> Arkansas Biosciences Institute, Department of Chemistry and Physics, Arkansas State

19 University, Jonesboro, Arkansas, United States of America

20 <sup>10</sup> Current Address: Department of Agronomy and Horticulture, Center for Plant Science

21 Innovation, Beadle Center for Biotechnology, University of Nebraska - Lincoln, Lincoln,

22 Nebraska, United States of America

23 <sup>11</sup> Cosmos X, Tokyo, Japan

24 <sup>12</sup> Missouri University of Science and Technology, Rolla, Missouri, United States of America

25

26 \* These authors contributed equally to this work

27

28 Corresponding Author:

29 Malia Gehan; Noah Fahlgren

30 975 North Warson Road, St. Louis, MO 63132, USA

31 Email address: [mgehan@danforthcenter.org](mailto:mgehan@danforthcenter.org); [nfahlgren@danforthcenter.org](mailto:nfahlgren@danforthcenter.org)

32 **ABSTRACT**

33 Systems for collecting image data in conjunction with computer vision techniques are a powerful  
34 tool for increasing the temporal resolution at which plant phenotypes can be measured non-  
35 destructively. Computational tools that are flexible and extendable are needed to address the  
36 diversity of plant phenotyping problems. We previously described the Plant Computer Vision  
37 (PlantCV) software package, which is an image processing toolkit for plant phenotyping  
38 analysis. The goal of the PlantCV project is to develop a set of modular, reusable, and  
39 repurposable tools for plant image analysis that are open-source and community-developed. Here  
40 we present the details and rationale for major developments in the second major release of  
41 PlantCV. In addition to overall improvements in the organization of the PlantCV project, new  
42 functionality includes a set of new image processing and normalization tools, support for  
43 analyzing images that include multiple plants, leaf segmentation, landmark identification tools  
44 for morphometrics, and modules for machine learning.

45

## 46 INTRODUCTION

47 All approaches for improving crops eventually require measurement of traits (phenotyping)  
48 (Fahlgren, Gehan & Baxter, 2015). However, manual plant measurements are time-consuming  
49 and often require destruction of plant materials in the process, which prevents measurement of  
50 traits for a single plant through time. Consequently, plant phenotyping is widely recognized as a  
51 major bottleneck in crop improvement (Furbank & Tester, 2011). Targeted plant phenotypes can  
52 range from measurement of gene expression, to flowering time, to grain yield; therefore, the  
53 software and hardware tools used are often diverse. Here, we focus on the software tools  
54 required to nondestructively measure plant traits through images. This is a challenging area of  
55 research because the visual definition of phenotypes vary depending on the target species. For  
56 example, for identification of flowering time, floral structures might be yellow like those of  
57 *Camelina sativa* but also can be green like those of *Brachypodium distachyon*. Therefore,  
58 software tools needed to process high-throughput image data need to be flexible and amenable to  
59 community input.

60

61 The term ‘high-throughput’ is relative to the difficulty to collect the measurement. The scale that  
62 might be considered high-throughput for root phenotyping might not be the same for shoot  
63 phenotyping, which can be technically easier to collect depending on the trait and species. Here  
64 we define high-throughput as thousands or hundreds of thousands of images per dataset. PlantCV  
65 is an open-source, open-development suite of analysis tools capable of analyzing high-  
66 throughput image-based phenotyping data (Fahlgren et al., 2015). Version 1.0 of PlantCV  
67 (PlantCV v1.0) was released in 2015 alongside the introduction of the Bellwether Phenotyping  
68 Facility at the Donald Danforth Plant Science Center (Fahlgren et al., 2015). PlantCV v1.0 was

69 envisioned as a base suite of tools that the community could build upon, which lead to several  
70 design decisions aimed at encouraging participation. First, GitHub was used as a platform to  
71 organize the community by integrating version control, code distribution, documentation, issue  
72 tracking, and communication between users and contributors (Perez-Riverol et al., 2016).  
73 Second, PlantCV was written in Python, a high-level language widely used for both teaching and  
74 bioinformatics (Mangalam, 2002; Dudley & Butte, 2009), to facilitate contribution from both  
75 biologists and computer scientists. Additionally, the use of Python allows extension of PlantCV  
76 with the many tools available from the Python scientific computing community (Oliphant, 2007;  
77 Millman & Aivazis, 2011). Third, a focus on modular development fosters code reuse and makes  
78 it easier to integrate PlantCV with new or existing systems. Finally, the use of a permissive,  
79 open-source license (MIT) allows PlantCV to be used, reused, or repurposed with limited  
80 restrictions, for both academic and proprietary applications. The focus of the paper associated  
81 with the original release of PlantCV v1.0 (Fahlgren et al., 2015) was not the structure and  
82 function of PlantCV for image analysis, but rather an example of the type of biological question  
83 that can be answered with high-throughput phenotyping hardware and software platforms. Since  
84 the release of PlantCV v1.0 major changes have been made to increase the flexibility, usability,  
85 and functionality of PlantCV. Here we document the structure of PlantCV v2.0 along with  
86 examples that demonstrate new functionality.

87

## 88 **RESULTS AND DISCUSSION**

89 The following are details on improvements to the structure, usability, and functionality of  
90 PlantCV since the v1.0 release. Further documentation for using PlantCV can be found at the  
91 project website (<http://plantcv.danforthcenter.org/>).

## 92 **Organization of the PlantCV project**

93 PlantCV is made up of modular Python functions, which are reusable units of Python code with  
94 defined inputs and outputs (Fig. 1). PlantCV functions can be assembled into simple sequential  
95 or branching/merging pipelines. A pipeline can be as long or as short as it needs to be, allowing  
96 for maximum flexibility for users using different imaging systems and analyzing features of  
97 seed, shoot, root, or other plant systems. Each function has a debugging option to allow users to  
98 view and evaluate the output of a single step and adjust parameters as necessary. Once a  
99 satisfactory pipeline script is developed, the PlantCV parallelization script can be used to deploy  
100 the pipeline across a large set of image data (Fig. 1). The parallelization script also functions to  
101 manage data by consolidating measurements and metadata into an SQLite database (Fig. 1). In  
102 terms of speed, the user is only limited by the complexity of the pipeline and the number of  
103 available processors.

104

105 The modular structure of the PlantCV package makes it easier for members of the community to  
106 become contributors. Contributors to PlantCV submit bug reports, develop new functions, or  
107 extend existing functionality or documentation. Core PlantCV developers do not filter additions  
108 of new functions in terms of perceived impact or number of users but do check that new  
109 functions follow the PlantCV contribution guide, which can be found in the PlantCV GitHub  
110 repository and in the online documentation (<http://plantcv.danforthcenter.org>). PlantCV  
111 contributors are asked to follow the PEP8 Python style guide  
112 (<https://www.python.org/dev/peps/pep-0008/>). Additions or revisions to the PlantCV code or  
113 documentation are submitted for review using pull requests via GitHub. The pull request  
114 mechanism is essential to protect against merge conflicts, which are sections of code that have

115 been edited by multiple users in potentially incompatible ways. In PlantCV v2.0, several service  
116 integrations were added to automate common tasks during pull requests and updates to the code  
117 repository. A continuous integration framework using the Travis CI service (<https://travis-ci.org/>)  
118 was added so that software builds and unit tests can be run automatically upon pull requests and  
119 other software updates. Continuous integration provides a safeguard against code updates that  
120 break existing functionality by providing a report that shows which tests passed or failed for each  
121 build (Wilson et al., 2014). The effectiveness of continuous integration depends on having  
122 thorough unit test coverage of the PlantCV code base. Unit test coverage of the PlantCV Python  
123 package is monitored through the Coveralls service (<https://coveralls.io/>), which provides a  
124 report on which parts of the code are covered by existing unit tests. In addition to the code, the  
125 PlantCV documentation was updated to use a continuous documentation framework using the  
126 Read the Docs service (<https://readthedocs.org/>), which allows documentation to be updated  
127 automatically and versioned in parallel with updates to PlantCV. The documentation was  
128 updated to cover all functions in the PlantCV library, tutorials on building pipelines and using  
129 specialized tools (e.g. multi-plant analysis and machine learning tools), a frequently asked  
130 questions section, and several guides such as installation, Jupyter notebooks, and instructions for  
131 contributors.

132

### 133 **Improved usability**

134 PlantCV v1.0 required pipeline development to be done using the command line, where debug  
135 mode is used to write intermediate image files to disk for each step. In command-line mode, the  
136 entire pipeline script must be executed, even if only a single step is being evaluated. To improve  
137 the pipeline and function development process in PlantCV v2.0, the debugging system was

138 updated to allow for seamless integration with the Jupyter Notebook system (<http://jupyter.org/>)  
139 (Kluyver et al., 2016). Jupyter compatibility allows users to immediately visualize output and to  
140 iteratively rerun single steps in a multi-step PlantCV pipeline, which makes parameters like  
141 thresholds or regions of interest much easier to adjust. Once a pipeline is developed in Jupyter, it  
142 is relatively straightforward to convert the notebook into a PlantCV pipeline script (see the  
143 online documentation). Jupyter notebooks are also a convenient method of sharing pipelines with  
144 collaborators and can be downloaded in multiple formats for publication purposes. Jupyter  
145 notebooks are also a helpful tool to teach others to use PlantCV.

146  
147 PlantCV was initially created to analyze data generated by the Bellwether Phenotyping Facility  
148 at the Donald Danforth Plant Science Center. Several updates to PlantCV v2.0 addressed the  
149 need to increase the flexibility of PlantCV to analyze data from other plant phenotyping systems.  
150 The PlantCV SQLite database schema was simplified so that new tables do not need to be added  
151 for every new camera system. The full database schema is available on GitHub (see Materials  
152 and Methods). New utilities were added to PlantCV v2.0 that allow data to be quickly and  
153 efficiently exported from the SQLite database into text files that are compatible with R (R Core  
154 Team, 2017) for further statistical analysis and data visualization.

155  
156 Because standards for data collection and management for plant phenotyping data are still being  
157 developed (Pauli et al., 2016), image metadata is often stored in a variety of formats on different  
158 systems. A common approach is to include metadata within image filenames, but because there  
159 is a lack of file naming standards, it can be difficult to robustly capture this data automatically. In  
160 PlantCV v2.0, a new metadata processing system was added to allow for flexibility in file



161 naming both within and between experiments and systems. The PlantCV metadata processing  
162 system is part of the parallelization tool and works by using a user-provided template to process  
163 filenames. User-provided templates are built using a restricted vocabulary so that metadata can  
164 be collected in a standardized way. The vocabulary used can be easily updated to accommodate  
165 future community standards.

166

### 167 **Performance**

168 In PlantCV v1.0, image analysis parallelization was achieved using a Perl-based multi-threading  
169 system that was not thread-safe, which occasionally resulted in issues with data output that had  
170 to be manually corrected. Additionally, the use of the Python package Matplotlib (Hunter, 2007)  
171 in PlantCV v1.0 limited the number of usable processors to 10-12. For PlantCV v2.0, the  
172 parallelization framework was completely rewritten in Python using a multiprocessing  
173 framework, and the use of Matplotlib was updated to mitigate the issues and processor  
174 constraints in v1.0. The output of image files mainly used to assess image segmentation quality  
175 is now optional, which should generally increase computing performance. Furthermore, to  
176 decentralize the computational resources needed for parallel processing and prepare for future  
177 integration with high-throughput computing resources that use file-in-file-out operations, results  
178 from PlantCV pipeline scripts (one per image) are now written out to temporary files that are  
179 aggregated by the parallelization tool after all image processing is complete.

180

### 181 **New Functionality**

182 PlantCV v2.0 has added new functions for image white balancing, auto-thresholding, size marker  
183 normalization, multi-plant detection, combined image processing, watershed segmentation,

184 landmarking, and a trainable naive Bayes classifier for image segmentation (machine learning).

185 The following are short descriptions and sample applications of new PlantCV functions.

186

187 *White balancing*

188 If images are captured in a greenhouse, growth chamber, or other situation where light intensity

189 is variable, image segmentation based on global thresholding of image intensity values can

190 become variable. To help mitigate image inconsistencies that might impair the ability to use a

191 single global threshold and thus a single pipeline over a set of images, a white balance function

192 was developed. If a white color standard is visible within the image, the user can specify a region

193 of interest. If a specific area is not selected then the whole image is used. Each channel of the

194 image is scaled relative to the reference maximum.

195

196 *Auto-thresholding functions*

197 An alternative approach to using a fixed, global threshold for image segmentation is to use an

198 auto-thresholding technique that either automatically selects an optimal global threshold value or

199 introduces a variable threshold for different regions in an image. Triangle, Otsu, mean, and

200 Gaussian auto-thresholding functions were added to PlantCV to further improve object detection

201 when image light sources are variable. The ‘triangle\_auto\_threshold’ function implements the

202 method developed by Zack et al. 1977 (Zack, Rogers & Latp, 1977). The triangle threshold

203 method uses the histogram of pixel intensities to differentiate the target object (plant) from

204 background by generating a line from the peak pixel intensity (Duarte, 2015) to the last pixel

205 value and then finding the point (i.e., the threshold value) on the histogram that maximizes

206 distance to that line. In addition to producing the thresholded image in debug mode, the

207 ‘triangle\_auto\_threshold’ function outputs the calculated threshold value and the histogram of  
208 pixel intensities that was used to calculate the threshold. In cases where the auto-threshold value  
209 does not adequately separate the target object from background, the threshold can be adjusted by  
210 modifying the stepwise input. Modifying the stepwise input shifts the distance calculation along  
211 the x-axis, which subsequently calculates a new threshold value to use.

212

213 The Otsu, mean, and Gaussian threshold functions in PlantCV are implemented using the  
214 OpenCV library (Bradski, 2000). Otsu’s binarization [‘otsu\_auto\_threshold;’ (Otsu, 1979)] is  
215 best implemented when a grayscale image histogram has two peaks since the Otsu method  
216 selects a threshold value that minimizes the weighted within-class variance. In other words, the  
217 Otsu method identifies the value between two peaks where the variances of both classes are  
218 minimized. Mean and Gaussian thresholding are executed by indicating the desired threshold  
219 type in the function ‘adaptive\_threshold.’ The mean and Gaussian methods will produce a  
220 variable local threshold where the threshold value of a pixel location depends on the intensities  
221 of neighboring pixels. For mean adaptive thresholding, the threshold of a pixel location is  
222 calculated by the mean of surrounding pixel values; for Gaussian adaptive thresholding, the  
223 threshold value of a pixel is the weighted sum of neighborhood values using a Gaussian window  
224 (Gonzalez & Woods, 2002; Kaehler & Bradski, 2016).

225

226 *Gaussian blur*

227 In addition to the ‘median\_blur’ function included in PlantCV v1.0, we have added a Gaussian  
228 blur smoothing function to reduce image noise and detail. Both the median and Gaussian blur  
229 methods are implemented using the OpenCV library (Bradski, 2000) and are typically used to

230 smooth a binary image that has been previously thresholded. Utilizing a rectangular  
231 neighborhood around a center pixel, 'median\_blur' replaces each pixel in the neighborhood with  
232 the median value. Alternatively, 'gaussian\_blur' determines the value of the central pixel by  
233 multiplying its and neighboring pixel values by a normalized kernel and then averaging these  
234 weighted values (i.e., image convolution) (Kaehler & Bradski, 2016). The extent of image  
235 blurring can be modified by increasing (for greater blur) or decreasing the kernel size (which  
236 takes only odd numbers; commonly, 3x3) or by changing the standard deviation in the X and/or  
237 Y directions.

238

#### 239 *Size marker normalization*

240 Images that are not collected from a consistent vantage point require one or more size markers as  
241 references for absolute or relative scale. The size marker function allows users to either detect a  
242 size marker within a user-defined region of interest or to select a specific region of interest to use  
243 as the size marker. The pixel area of the marker is returned as a value that can be used to  
244 normalize measurements to the same scale. For this module to function correctly we assume that  
245 the size marker stays in frame, is unobstructed, and is relatively consistent in position throughout  
246 a dataset, though some movement is allowed as long as the marker remains within the defined  
247 marker region of interest.

248

#### 249 *Multi-plant detection*

250 There is growing interest among the PlantCV user community to process images with multiple  
251 plants grown in flats or trays, but PlantCV v1.0 was built to processes images containing single  
252 plants. The major challenge with analyzing multiple plants in an image is successfully

253 identifying individual whole plants as distinct objects. Leaves or other plant parts can sometimes  
254 be detected as distinct contours from the rest of the plant and need to be grouped with other  
255 contours from the same plant to correctly form a single plant/target object. While creating  
256 multiple regions of interest (ROI) to demarcate each area containing an individual plant/target is  
257 an option, we developed two modules, 'cluster\_contours' and 'cluster\_contours\_split\_img,' that  
258 allow contours to be clustered and then parsed into multiple images without having to manually  
259 create multiple ROIs (Fig. 2).

260

261 The 'cluster\_contours' function takes as input: an image, the contours that need to be clustered, a  
262 number of rows, and a number of columns. Total image size is detected, and the rows and  
263 columns create a grid to serve as approximate ROIs to cluster the contours. The number of rows  
264 and columns approximate the desired size of the grid cells. There does not need to be an object in  
265 each of the grid cells. Several functions were also added to aid the clustering function. The  
266 'rotate\_img' and 'shift\_img' functions allow the image to be adjusted so objects are better  
267 aligned to a grid pattern.

268

269 After objects are clustered, the 'cluster\_contour\_split\_img' function splits images into the  
270 individual grid cells and outputs each as a new image so that there is a single clustered object per  
271 image. If there is no clustered object in a grid cell, no image is outputted. With the  
272 'cluster\_contour\_split\_img' function, a text file with genotype names can be included to add  
273 them to image names. The 'cluster\_contour\_split\_img' function also checks that there are the  
274 same number of names as objects. If there is a conflict in the number of names and objects, a  
275 warning is printed and a correction is attempted. Alternatively, if the file option is not used, all of

276 the object groups are labeled by position. Once images are split, they can be processed like single  
277 plant images using additional PlantCV tools. See the online documentation for an example multi-  
278 plant imaging pipeline ([http://plantcv.readthedocs.io/en/latest/multi-plant\\_tutorial/](http://plantcv.readthedocs.io/en/latest/multi-plant_tutorial/)).

279

280 The current method for multi-plant identification in PlantCV is flexible but relies on a grid  
281 arrangement of plants, which is common for controlled-environment-grown plants. Future  
282 releases of PlantCV may incorporate additional strategies for detection and identification of  
283 plants, such as arrangement-independent  $K$ -means clustering approaches (Minervini,  
284 Abdelsamea & Tsafaris, 2014).

285

#### 286 *Combined image processing*

287 The Bellwether Phenotyping Facility has both RGB visible light (VIS) and near-infrared (NIR)  
288 cameras, and images are captured ~1 minute apart (Fahlgren et al., 2015). Compared to VIS  
289 images, NIR images are grayscale with much less contrast between object and background. It can  
290 be difficult to segment plant material from NIR images directly, even with edge detection steps.  
291 Therefore, several functions were added to allow the plant binary mask that results from VIS  
292 image processing pipelines to be resized and used as a mask for NIR images. Combining VIS  
293 and NIR camera pipelines also has the added benefit of decreasing the number of steps necessary  
294 to process images from both camera types, thus increasing image processing throughput. The  
295 ‘get\_nir’ function identifies the path of the NIR image that matches VIS image. The ‘get\_nir’  
296 function requires that the image naming scheme is consistent and that the matching image is in  
297 the same image directory. The ‘resize’ function then resizes the VIS plant mask in both the x and  
298 y directions to match the size of the NIR image. Resizing values are determined by measuring

299 the same reference object in an example image taken from both VIS and NIR cameras (for  
300 example the width of the pot or pot carrier in each image). The ‘crop\_position\_mask’ function is  
301 then used to adjust the placement of the VIS mask over the NIR image and to crop/adjust the VIS  
302 mask so it is the same size as the NIR image. It is assumed that the pot position changes  
303 consistently between VIS and NIR image datasets. An example VIS/NIR dual pipeline to follow  
304 can be accessed online ([http://plantcv.readthedocs.io/en/latest/vis\\_nir\\_tutorial/](http://plantcv.readthedocs.io/en/latest/vis_nir_tutorial/)).

305

### 306 *Object count estimation with watershed segmentation*

307 While segmentation and analysis of whole plants in images provides useful information about  
308 plant size and growth, a more detailed understanding of plant growth and development can be  
309 obtained by measuring individual plant organs. However, fully automated segmentation of  
310 individual organs such as leaves remains a challenge, due to issues such as occlusion (Schar et  
311 al., 2016). Multiple methods for leaf segmentation have been proposed (Schar et al., 2016), and  
312 in PlantCV v2.0 we have implemented a watershed segmentation approach. The  
313 ‘watershed\_segmentation’ function can be used to estimate the number of leaves for certain plant  
314 architecture types and imaging platforms (Fig. 3). The inputs required are an image, an object  
315 mask, and a minimum distance to separate object peaks. The function uses the input mask to  
316 calculate a Euclidean distance map (Liberti et al., 2014). Marker peaks calculated from the  
317 distance map that meet the minimum distance setting are used in a watershed segmentation  
318 algorithm (van der Walt et al., 2014) to segment and count the objects. Segmented objects are  
319 visualized in different colors, and the number of segmented objects is reported (Fig. 3). An  
320 example of how the watershed segmentation method was used to assess the effect of water

321 deficit stress on the number of leaves of Arabidopsis plants can be found in Acosta-Gamboa et  
322 al. 2017 (Acosta-Gamboa et al., 2017).

323

324 *Landmarking functions for morphometrics*

325 To extend PlantCV beyond quantification of size-based morphometric features, we developed  
326 several landmarking functions. Landmarks are generally geometric points located along the  
327 contours of a shape that correspond to homologous biological features that can be compared  
328 between subjects (Bookstein, 1991). Typical examples of landmarks include eyes between  
329 human subjects or suture joins in a skull. For a growing plant, potential landmarks include the  
330 tips of leaves and pedicel and branch angles. When specified *a priori*, landmarks should be  
331 assigned to provide adequate coverage of the shape morphology across a single dimensional  
332 plane (Bookstein, 1991). Additionally, the identification of landmark points should be repeatable  
333 and reliable across subjects while not altering their topological positions relative to other  
334 landmark positions (Bookstein, 1991). Type I landmarks provide the strongest support for  
335 homology because they are defined by underlying biological features, but it is problematic to  
336 assign Type I landmarks *a priori* when analyzing high-throughput plant imagery. To address  
337 this, PlantCV v2.0 contains functions to identify anatomical landmarks based upon the  
338 mathematical properties of object contours (Type II) and non-anatomical pseudo-  
339 landmarks/semilandmarks (Type III), as well as functions to rescale and analyze biologically  
340 relevant shape properties (Bookstein, 1991, 1997; Gunz, Mitteroecker & Bookstein, 2005; Gunz  
341 & Mitteroecker, 2013).

342



343 The ‘acute’ function identifies Type II landmarks by implementing a pseudo-landmark  
344 identification algorithm that operates using a modified form of chain coding (Freeman, 1961).  
345 Unlike standard chain coding methods that attempt to capture the absolute shape of a contour, the  
346 acute method operates by measuring the angle between a pixel coordinate and two neighboring  
347 pixels on opposite sides of it that fall within a set distance, or window, along the length of the  
348 contour. The two neighboring points are used to calculate an angle score for the center pixel.  
349 When the angle score is calculated for each position along the length of a contour, clusters of  
350 acute points can be identified, which can be segmented out by applying an angle threshold. The  
351 middle position within each cluster of acute points is then identified for use as a pseudo-  
352 landmark (Fig. 4A). The ability to subjectively adjust the window size used for generating angle  
353 scores also helps to tailor analyses for identifying points of interest that may differ in resolution.  
354 For example, an analysis of leaf data might utilize a larger window size to identify the tips of  
355 lobes whereas smaller window sizes would be able to capture more minute patterns such as  
356 individual leaf serrations. Further segmentation can also be done using the average pixel values  
357 output (pt\_vals) for each pseudo-landmark, which estimates the mean pixel intensity within the  
358 convex hull of each acute region based on the binary mask used in the analysis. The average  
359 pixel value output allows for concave landmarks (e.g. leaf axils and grass ligules) and convex  
360 landmarks (e.g. leaf tips and apices) on a contour to be differentiated in downstream analyses.  
361 Additionally, PlantCV v2.0 includes the ‘acute\_vertex’ function that uses the same chain code-  
362 based pseudo-landmark identification algorithm used in the ‘acute’ function except that it uses an  
363 adjustable local search space criteria to reduce the number of angle calculations, which speeds up  
364 landmark identification.  
365

366 For Type III landmarks, the ‘x\_axis\_pseudolandmarks’ and ‘y\_axis\_pseudolandmarks’ functions  
367 identify homologous points along a single dimension of an object (x-axis or y-axis) based on  
368 equidistant point locations within an object contour. The plant object is divided up into twenty  
369 equidistant bins, and the minimum and maximum extent of the object along the axis and the  
370 centroid of the object within each bin is calculated. These sixty points located along each axis  
371 possess the properties of semi/pseudo-landmark points (an equal number of reference points that  
372 are approximately geometrically homologous between subjects to be compared) that approximate  
373 the contour and shape of the object (Fig. 4B). Such semi/pseudo-landmarking strategies have  
374 been utilized in cases where traditional homologous landmark points are difficult to assign or  
375 poorly represent the features of object shape (Bookstein, 1997; Gunz, Mitteroecker & Bookstein,  
376 2005; Gunz & Mitteroecker, 2013).

377

378 Frequently, comparison of shape attributes requires rescaling of landmark points to eliminate the  
379 influence of size on the relative position of landmark points. The landmark functions in PlantCV  
380 output untransformed point values that can either be directly input into morphometric programs  
381 in R [shapes (Dryden & Mardia, 2016) or morpho (Schlager, Jefferis & Schlager, 2016)] or  
382 uniformly rescaled to a 0-1 coordinate system using the PlantCV ‘scale\_features’ function. The  
383 location of landmark points can be used to examine multidimensional growth curves for a broad  
384 variety of study systems and tissue types and can be used to compare properties of plant shape  
385 throughout development or in response to differences in plant growth environment. An example  
386 of one such application is the ‘landmark\_reference\_pt\_dist’ function. This function estimates the  
387 vertical, horizontal, Euclidean distance, and angle of landmark points from two landmarks  
388 (centroid of the plant object and centroid localized to the base of the plant). Preliminary evidence

389 from a water limitation experiment performed using a *Setaria* recombinant inbred population  
390 indicates that vertical distance from rescaled leaf tip points identified by the ‘acute\_vertex’  
391 function to the centroid is decreased in response to water limitation and thus may provide a  
392 proximity measurement of plant turgor pressure (Fig. 4C and 4D).

393

#### 394 *Two-class or multiclass naive Bayes classifier*

395 Pixel-level segmentation of images into two or more classes is not always straightforward using  
396 traditional image processing techniques. For example, two classes of features in an image may be  
397 visually distinct but similar enough in color that simple thresholding is not sufficient to separate  
398 the two groups. Furthermore, even with methods that adjust for inconsistencies between images  
399 (e.g. white balancing and auto-thresholding functions), inconsistent lighting conditions in a  
400 growth chamber, greenhouse, or field can still make bulk processing of images with a single  
401 workflow difficult. Methods that utilize machine learning techniques are a promising approach to  
402 tackle these and other phenotyping challenges (Minervini, Abdelsamea & Tsafaris, 2014; Singh  
403 et al., 2016; Ubbens & Stavness, 2017; Atkinson et al., 2017; Pound et al., 2017). With PlantCV  
404 v2.0, we have started to integrate machine learning methods to detect features of interest (e.g. the  
405 plant), starting with a naive Bayes classifier (Abbasi & Fahlgren, 2016). The naive Bayes  
406 classifier can be trained using two different approaches for two-class or multiclass (two or more)  
407 segmentation problems. During the training phase using the plantcv-train.py script, pixel RGB  
408 values for each input class are converted to the hue, saturation and value (HSV) color space.  
409 Kernel density estimation (KDE) is used to calculate a probability density function (PDF) from a  
410 vector of values for each HSV channel from each class. The output PDFs are used to  
411 parameterize the naive Bayes classifier function (‘naive\_bayes\_classifier’), which can be used to

412 replace the thresholding steps in a PlantCV pipeline. The ‘naive\_bayes\_classifier’ function uses  
413 these PDFs to calculate the probability (using Bayes’ theorem) that a given pixel is in each class.  
414 The output of the ‘naive\_bayes\_classifier’ is a binary image for each class where the pixels are  
415 white if the probability the pixel was in the given class was highest of all classes and is black  
416 otherwise. A tutorial of how to implement naive Bayes plant detection into an image processing  
417 pipeline is online ([http://plantcv.readthedocs.io/en/latest/machine\\_learning\\_tutorial/](http://plantcv.readthedocs.io/en/latest/machine_learning_tutorial/)).

418

419 For the two-class approach, the training dataset includes color images and corresponding binary  
420 masks where the background is black and the foreground (plant or other target object) is white.  
421 PlantCV can be used to generate binary masks for the training set using the standard image  
422 processing methods and the new ‘output\_mask’ function. It is important for the training dataset  
423 to be representative of the larger dataset. For example, if there are large fluctuations in light  
424 intensity throughout the day or plant color throughout the experiment, the training dataset should  
425 try to cover the range of variation. A random sample of 10% of the foreground pixels and the  
426 same number background pixels are used to build the PDFs.

427

428 To assess how well the two-class naive Bayes method identifies plant material in comparison to  
429 thresholding methods, we reanalyzed *Setaria* images (Fahlgren et al., 2015) using the naive  
430 Bayes classifier and compared the pixel area output to pipelines that utilize thresholding steps  
431 (Fig. 5). We used 99 training images (14 top view and 85 side view images) from a total of 6473  
432 images. We found that the plant pixel area calculated by naive Bayes was highly correlated with  
433 that calculated from pipelines that use thresholding for both side-view images ( $R^2=0.99$ ; Fig. 5A)  
434 and top-view images ( $R^2=0.96$ ; Fig. 5B). An additional advantage to using the naive Bayes

435 image processing pipelines was that only five pipelines were needed to process the dataset that  
436 was originally processed with nine threshold-based pipelines.

437

438 The multiclass naive Bayes approach requires a tab-delimited table for training where each  
439 column is a class (minimum two) and each cell is a comma-separated list of RGB pixel values  
440 from the column class. We currently use the Pixel Inspection Tool in ImageJ (Schneider,  
441 Rasband & Eliceiri, 2012) to collect samples of pixel RGB values used to generate the training  
442 text file. As noted above for the two-class approach, it is important to adequately capture the  
443 variation in the image dataset for each class when generating the training text file to improve  
444 pixel classification. If images are consistent, only one image needs to be sampled for generating  
445 the training table; however, if they vary, several images may be needed. For complex  
446 backgrounds (or non-targeted objects), several classes may be required to capture all of the  
447 variation. Once the training table is generated, it is input into the plantcv-train.py script to  
448 generate PDFs for each class. As an example, we used images of wheat leaves infected with  
449 wheat rust to collect pixel samples from four classes: non-plant background, unaffected leaf  
450 tissue, rust pustule, and chlorotic leaf tissue, and then used the naive Bayes classifier to segment  
451 the images into each class simultaneously (Fig. 6). This method can likely be used for a variety  
452 of applications, such as identifying a plant under variable lighting conditions or quantifying  
453 specific areas of stress on a plant.

454

## 455 **CONCLUSIONS**

456 The field of digital plant phenotyping is at an exciting stage of development where it is  
457 beginning to shift from a bottleneck to one that will have a positive impact on plant research,

458 especially in agriculture. The Plant Image Analysis database currently lists over 150 tools that  
459 can be used for plant phenotyping [<http://www.plant-image-analysis.org/>; (Lobet, Draye &  
460 Périlleux, 2013)]. Despite the abundance of software packages, long-term sustainability of  
461 individual projects may become an issue due to the lack of incentives for maintaining  
462 bioinformatics software developed in academia (Lobet, 2017). In a survey of corresponding  
463 authors of plant image analysis tools by Lobet, 60% either said the tool was no longer being  
464 maintained or did not respond (Lobet, 2017). To develop PlantCV as a sustainable project we  
465 have adopted an open, community-based development framework using GitHub as a central  
466 service for the organization of developer activities and the dissemination of information to users.  
467 We encourage contribution to the project by posting bug reports and issues, developing or  
468 revising analysis methods, adding or updating unit tests, writing documentation, and posting  
469 ideas for new features. We aim to periodically publish updates, such as the work presented here,  
470 to highlight the work of contributors to the PlantCV project.

471  
472 There are several areas where we envision future PlantCV development. **Standards and**  
473 **interoperability:** Improved interoperability of PlantCV with data providers and downstream  
474 analysis tools will require adoption of community-based standards for data and metadata [e.g.  
475 Minimum Information About a Plant Phenotyping Experiment; (Ćwiek-Kupczyńska et al.,  
476 2016)]. Improved interoperability will make it easier to develop standardized tools for statistical  
477 analysis of image processing results, both within the PlantCV project or with tools from other  
478 projects. **New data sources:** Handling and analysis of data from specialized cameras that  
479 measure three-dimensional structure or hyperspectral reflectance will require development or  
480 integration of additional methods into PlantCV. **Machine learning:** Our goal is to develop

481 additional tools for machine learning and collection of training data. In some cases, where these  
482 methods can be implemented in a modular and reusable framework, they can be integrated  
483 directly into PlantCV. In other cases, PlantCV can be combined with new and existing tools. A  
484 recent example of this latter approach built on PlantCV, using its image preprocessing and  
485 segmentation functions alongside a modular framework for building convolutional neural  
486 networks (Ubbens & Stavness, 2017). As noted throughout, we see great potential for modular  
487 tools such as PlantCV and we welcome community feedback.

488

## 489 **MATERIALS & METHODS**

490 Scripts, notebooks, and simple input data associated with the present study are available on  
491 GitHub at <https://github.com/danforthcenter/plantcv-v2-paper>. The image of *Arabidopsis*  
492 *thaliana* was captured with a Raspberry Pi computer and camera in a Conviron growth chamber.  
493 Additional details about the imaging set-up are provided in a companion paper (Tovar et al.,  
494 2017). Images of *Setaria viridis* (A10) and *Setaria italica* (B100) are from publicly available  
495 datasets that are available at <http://plantcv.danforthcenter.org/pages/data.html> (Fahlgren et al.,  
496 2015; Feldman et al., 2017). The image of wheat (*Triticum aestivum* L.) infected with wheat  
497 stem rust (*Puccinia graminis* f. sp. *tritici*) was acquired with a flatbed scanner.

498

499 Image analysis was done in PlantCV using Python v2.7.5, OpenCV v2.4.5 (Bradski, 2000),  
500 NumPy v1.12.1 (van der Walt, Colbert & Varoquaux, 2011), Matplotlib v2.0.2 (Hunter, 2007),  
501 SciPy v0.19.0 (Jones, Oliphant & Peterson), Pandas v0.20.1 (McKinney & Others, 2010), scikit-  
502 image v0.13.0 (van der Walt et al., 2014), and Jupyter Notebook v4.2.1 (Kluyver et al., 2016).  
503 Statistical analysis and data visualization was done using R v3.3 (R Core Team, 2017) and

504 RStudio v1.0 (RStudio Team, 2016). Graphs were produced using Matplotlib v2.0.2 (Hunter,  
505 2007) and ggplot2 v2.2.1 (Wickham, 2009).

506

## 507 **ACKNOWLEDGMENTS**

508 We would like to thank Melinda Darnell, Leonardo Chavez, Kevin Reilly, and the staff of both  
509 the Danforth Center Facilities and Support Services group and the Plant Growth Facility for  
510 careful maintenance of the Danforth Center phenotyping facilities. We thank Katie Liberatore  
511 and Shahryar Kianian for images of wheat (*Triticum aestivum L.*). We would also like to thank  
512 all of the other people who have given us input on the PlantCV project in person or on GitHub.

513

## 514 **REFERENCES**

515 Abbasi A., Fahlgren N. 2016. Naive Bayes pixel-level plant segmentation. In: *2016 IEEE*  
516 *Western New York Image and Signal Processing Workshop (WNYISPW)*. 1–4. DOI:  
517 10.1109/WNYIPW.2016.7904790.

518 Acosta-Gamboa LM., Liu S., Langley E., Campbell Z., Castro-Guerrero N., Mendoza-Cozatl D.,  
519 Lorence A. 2017. Moderate to severe water limitation differentially affects the phenome and  
520 ionome of *Arabidopsis*. *Functional plant biology: FPB* 44:94–106. DOI: 10.1071/FP16172.

521 Atkinson JA., Lobet G., Noll M., Meyer PE., Griffiths M., Wells DM. 2017. Combining semi-  
522 automated image analysis techniques with machine learning algorithms to accelerate large  
523 scale genetic studies. *GigaScience*. DOI: 10.1093/gigascience/gix084.

524 Bookstein FL. 1991. Morphometric tools for landmark data Cambridge University Press. *New*  
525 *York*.

526 Bookstein FL. 1997. *Morphometric Tools for Landmark Data: Geometry and Biology*.



- 527 Cambridge University Press.
- 528 Bradski G. 2000. The opencv library. *Doctor Dobbs Journal* 25:120–126.
- 529 Ćwiek-Kupczyńska H., Altmann T., Arend D., Arnaud E., Chen D., Cornut G., Fiorani F.,  
530 Frohberg W., Junker A., Klukas C., Lange M., Mazurek C., Nafissi A., Neveu P., van  
531 Oeveren J., Pommier C., Poorter H., Rocca-Serra P., Sansone S-A., Scholz U., van Schriek  
532 M., Seren Ü., Usadel B., Weise S., Kersey P., Krajewski P. 2016. Measures for  
533 interoperability of phenotypic data: minimum information requirements and formatting.  
534 *Plant methods* 12:44. DOI: 10.1186/s13007-016-0144-4.
- 535 Dryden IL., Mardia KV. 2016. *Statistical Shape Analysis: With Applications in R*. John Wiley &  
536 Sons.
- 537 Duarte M. 2015. Notes on scientific computing for biomechanics and motor control. *GitHub*  
538 *repository*.
- 539 Dudley JT., Butte AJ. 2009. A quick guide for developing effective bioinformatics programming  
540 skills. *PLoS Computational Biology* 5:e1000589. DOI: 10.1371/journal.pcbi.1000589.
- 541 Fahlgren N., Feldman M., Gehan MA., Wilson MS., Shyu C., Bryant DW., Hill ST., McEntee  
542 CJ., Warnasooriya SN., Kumar I., Ficor T., Turnipseed S., Gilbert KB., Brutnell TP.,  
543 Carrington JC., Mockler TC., Baxter I. 2015. A versatile phenotyping system and analytics  
544 platform reveals diverse temporal responses to water availability in *Setaria*. *Molecular Plant*  
545 8:1520–1535. DOI: 10.1016/j.molp.2015.06.005.
- 546 Fahlgren N., Gehan MA., Baxter I. 2015. Lights, camera, action: high-throughput plant  
547 phenotyping is ready for a close-up. *Current Opinion in Plant Biology* 24:93–99. DOI:  
548 10.1016/j.pbi.2015.02.006.
- 549 Feldman MJ., Paul RE., Banan D., Barrett JF., Sebastian J., Yee M-C., Jiang H., Lipka AE.,

- 550 Brutnell TP., Dinneny JR., Leakey ADB., Baxter I. 2017. Time dependent genetic analysis  
551 links field and controlled environment phenotypes in the model C4 grass *Setaria*. *PLoS*  
552 *Genetics* 13:e1006841. DOI: 10.1371/journal.pgen.1006841.
- 553 Freeman H. 1961. On the encoding of arbitrary geometric configurations. *IRE Transactions on*  
554 *Electronic Computers* EC-10:260–268. DOI: 10.1109/TEC.1961.5219197.
- 555 Furbank RT., Tester M. 2011. Phenomics--technologies to relieve the phenotyping bottleneck.  
556 *Trends in Plant Science* 16:635–644. DOI: 10.1016/j.tplants.2011.09.005.
- 557 Gonzalez RC., Woods RE. 2002. *Digital Image Processing*. Prentice Hall.
- 558 Gunz P., Mitteroecker P. 2013. Semilandmarks: a method for quantifying curves and surfaces.  
559 *Hystrix, the Italian Journal of Mammalogy* 24:103–109.
- 560 Gunz P., Mitteroecker P., Bookstein FL. 2005. Semilandmarks in Three Dimensions. In: Slice  
561 DE ed. *Modern Morphometrics in Physical Anthropology*. Developments in Primatology:  
562 Progress and Prospects. Springer US, 73–98. DOI: 10.1007/0-387-27614-9\_3.
- 563 Hunter JD. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*  
564 9:90–95. DOI: 10.1109/MCSE.2007.55.
- 565 Jones E., Oliphant T., Peterson P. *SciPy: Open source scientific tools for Python*.
- 566 Kaehler A., Bradski G. 2016. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV*  
567 *Library*. O'Reilly Media, Inc.
- 568 Kluyver T., Ragan-Kelley B., Pérez F., Granger B., Bussonnier M., Frederic J., Kelley K.,  
569 Hamrick J., Grout J., Corlay S., Ivanov P., Avila D., Abdalla S., Willing C., Jupyter  
570 Development Team. 2016. Jupyter Notebooks – a publishing format for reproducible  
571 computational workflows. In: Loizides F, Schmidt B eds. *Positioning and Power in*  
572 *Academic Publishing: Players, Agents and Agendas: Proceedings of the 20th International*

- 573 *Conference on Electronic Publishing*. Amsterdam: IOS Press, 87–90. DOI: 10.3233/978-1-  
574 61499-649-1-87.
- 575 Liberti L., Lavor C., Maculan N., Mucherino A. 2014. Euclidean Distance Geometry and  
576 Applications. *SIAM Review* 56:3–69. DOI: 10.1137/120875909.
- 577 Lobet G. 2017. Image analysis in plant sciences: Publish then perish. *Trends in Plant Science*.  
578 DOI: 10.1016/j.tplants.2017.05.002.
- 579 Lobet G., Draye X., Périlleux C. 2013. An online database for plant image analysis software  
580 tools. *Plant Methods* 9:38. DOI: 10.1186/1746-4811-9-38.
- 581 Mangalam H. 2002. The Bio\* toolkits--a brief overview. *Briefings in Bioinformatics* 3:296–302.  
582 DOI: 10.1093/bib/3.3.296.
- 583 McKinney W., Others. 2010. Data structures for statistical computing in python. In: *Proceedings*  
584 *of the 9th Python in Science Conference*. SciPy Austin, TX, 51–56.
- 585 Millman KJ., Aivazis M. 2011. Python for Scientists and Engineers. *Computing in Science &*  
586 *Engineering* 13:9–12. DOI: 10.1109/MCSE.2011.36.
- 587 Minervini M., Abdelsamea MM., Tsafaris SA. 2014. Image-based plant phenotyping with  
588 incremental learning and active contours. *Ecological Informatics* 23:35–48. DOI:  
589 10.1016/j.ecoinf.2013.07.004.
- 590 Oliphant TE. 2007. Python for scientific computing. *Computing in Science & Engineering* 9:10–  
591 20. DOI: 10.1109/MCSE.2007.58.
- 592 Otsu N. 1979. A threshold selection method from gray-level histograms. *IEEE Transactions on*  
593 *Systems, Man, and Cybernetics* 9:62–66.
- 594 Pauli D., Chapman SC., Bart R., Topp CN., Lawrence-Dill CJ., Poland J., Gore MA. 2016. The  
595 quest for understanding phenotypic variation via integrated approaches in the field

- 596 environment. *Plant Physiology* 172:622–634. DOI: 10.1104/pp.16.00592.
- 597 Perez-Riverol Y., Gatto L., Wang R., Sachsenberg T., Uszkoreit J., Leprevost F da V., Fufezan  
598 C., Ternent T., Eglen SJ., Katz DS., Pollard TJ., Kononov A., Flight RM., Blin K.,  
599 Vizcaíno JA. 2016. Ten simple rules for taking advantage of Git and GitHub. *PLoS*  
600 *Computational Biology* 12:e1004947. DOI: 10.1371/journal.pcbi.1004947.
- 601 Pound MP., Atkinson JA., Townsend AJ., Wilson MH., Griffiths M., Jackson AS., Bulat A.,  
602 Tzimiropoulos G., Wells DM., Murchie EH., Pridmore TP., French AP. 2017. Deep  
603 Machine Learning provides state-of-the-art performance in image-based plant phenotyping.  
604 *GigaScience*. DOI: 10.1093/gigascience/gix083.
- 605 R Core Team. 2017. R: A Language and Environment for Statistical Computing.
- 606 RStudio Team. 2016. *RStudio: Integrated Development Environment for R*. Boston, MA:  
607 RStudio, Inc.
- 608 Schar H., Minervini M., French AP., Klukas C., Kramer DM., Liu X., Luengo I., Pape J-M.,  
609 Polder G., Vukadinovic D., Yin X., Tsaftaris SA. 2016. Leaf segmentation in plant  
610 phenotyping: a collation study. *Machine Vision and Applications* 27:585–606. DOI:  
611 10.1007/s00138-015-0737-3.
- 612 Schlager S., Jefferis G., Schlager MS. 2016. Package “Morpho.”
- 613 Schneider CA., Rasband WS., Eliceiri KW. 2012. NIH Image to ImageJ: 25 years of image  
614 analysis. *Nature Methods* 9:671–675.
- 615 Singh A., Ganapathysubramanian B., Singh AK., Sarkar S. 2016. Machine learning for high-  
616 throughput stress phenotyping in plants. *Trends in Plant Science* 21:110–124. DOI:  
617 10.1016/j.tplants.2015.10.015.
- 618 Tovar JC., Hoyer JS., Lin A., Tielking A., Tessman M., Miller M., Callen ST., Castillo SE.,

619 Fahlgren N., Carrington JC., Nusinow DA., Gehan MA. 2017. Raspberry Pi powered  
620 imaging for plant phenotyping. *Applications in Plant Sciences* submitted.

621 Ubbens JR., Stavness I. 2017. Deep Plant Phenomics: A deep learning platform for complex  
622 plant phenotyping tasks. *Frontiers in Plant Science* 8:1190. DOI: 10.3389/fpls.2017.01190.

623 van der Walt S., Colbert SC., Varoquaux G. 2011. The NumPy array: A structure for efficient  
624 numerical computation. *Computing in Science & Engineering* 13:22–30. DOI:  
625 10.1109/MCSE.2011.37.

626 van der Walt S., Schönberger JL., Nunez-Iglesias J., Boulogne F., Warner JD., Yager N.,  
627 Gouillart E., Yu T., scikit-image contributors. 2014. scikit-image: image processing in  
628 Python. *PeerJ* 2:e453. DOI: 10.7717/peerj.453.

629 Wickham H. 2009. *ggplot2: Elegant Graphics for Data Analysis*. Springer New York.

630 Wilson G., Aruliah DA., Brown CT., Chue Hong NP., Davis M., Guy RT., Haddock SHD., Huff  
631 KD., Mitchell IM., Plumbley MD., Waugh B., White EP., Wilson P. 2014. Best practices  
632 for scientific computing. *PLoS Biology* 12:e1001745. DOI: 10.1371/journal.pbio.1001745.

633 Zack GW., Rogers WE., Latp SA. 1977. Automatic measurement of sister chromatid exchange  
634 frequency. *Journal of Histochemistry and Cytochemistry* 25:741–753.

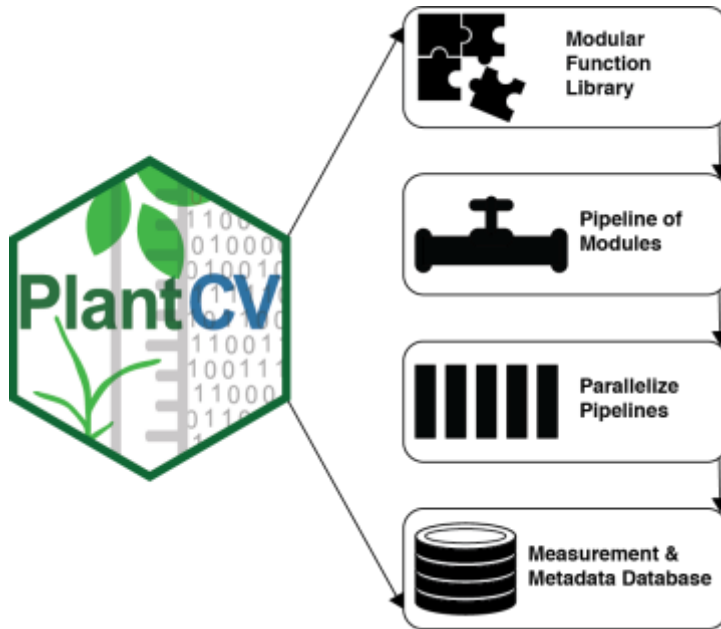
635

636

637

638

639 **FIGURE 1**



640

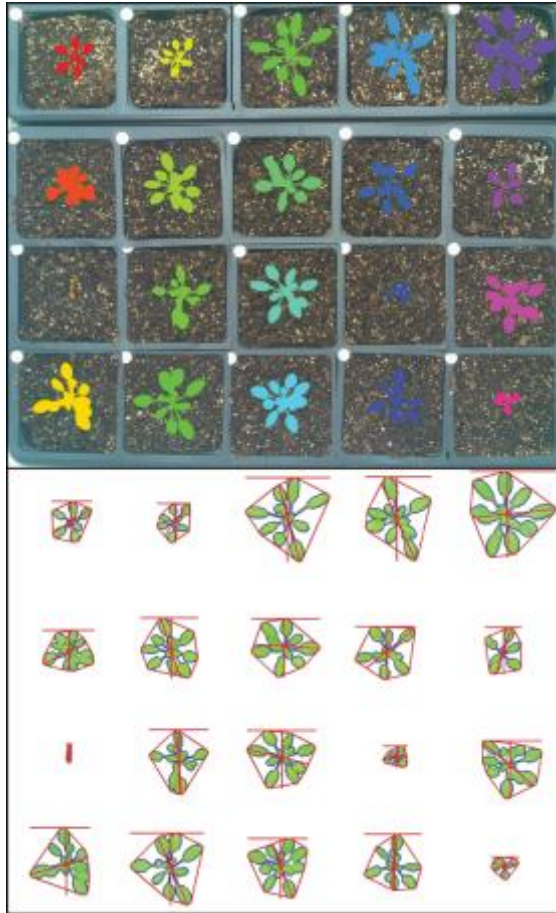
641 **Figure 1: Diagram of the components of PlantCV.**

642 PlantCV is an open-source, open-development suite of image analysis tools. PlantCV contains a  
643 library of modular Python functions that can be assembled into simple sequential or  
644 branching/merging processing pipelines. Image processing pipelines, which process single  
645 images (possibly containing multiple plants), can be deployed over large image sets using  
646 PlantCV parallelization, which outputs an SQLite database of both measurements and  
647 image/experimental metadata.

648

649

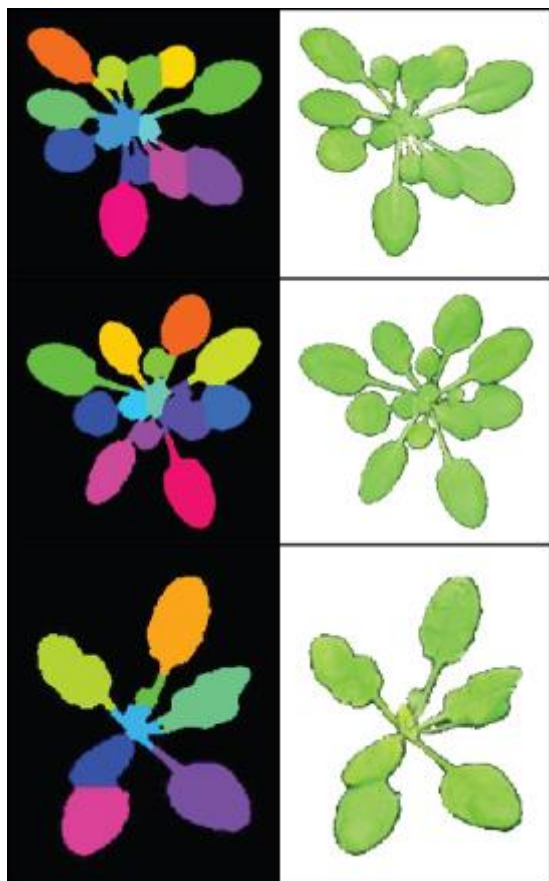
650

651 **FIGURE 2**

652

653 **Figure 2: Analysis of images containing multiple plants.**

654 New functions have been added to PlantCV v2.0 that enable individual plants from images  
655 containing multiple plants to be analyzed. The 'cluster\_contours' function clusters contour  
656 objects using a flexible grid arrangement (approximate rows and columns defined by a user). The  
657 top image, produced by 'cluster\_contours' in debug mode, highlights plants by their cluster  
658 group with unique colors on a sequential scale. The 'cluster\_contours\_split\_img' function creates  
659 a new image for each cluster group. The resulting images of individual plants can be processed  
660 by standard PlantCV methods. In the bottom image, the 'cluster\_contours\_split\_img' function  
661 was used to split the full image into individual plants. The shape of each plant was then analyzed  
662 with 'analyze\_objects' and printed on a common image background.

663 **FIGURE 3**

664

665 **Figure 3: Leaf segmentation by a distance-based watershed transformation.**

666 The watershed segmentation function can be used to segment and estimate the number of objects  
667 in an image. For the three example images, the watershed segmentation function was used to  
668 estimate the number of leaves for *Arabidopsis thaliana* (estimated leaf count for top: 13, middle:  
669 14, and bottom: 8). Images shown are the output from the 'watershed\_segmentation' function  
670 (left) and the segmented plants (right).

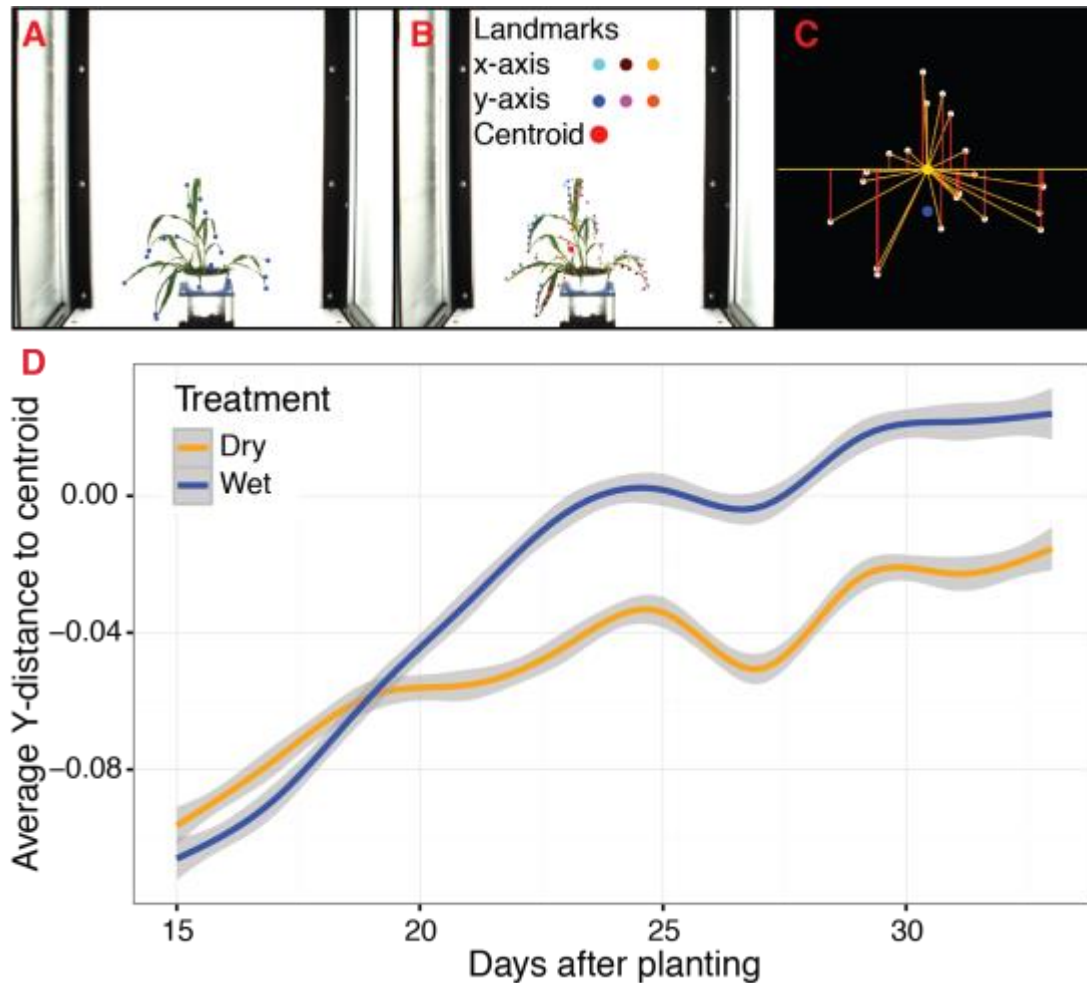
671

672

673

674



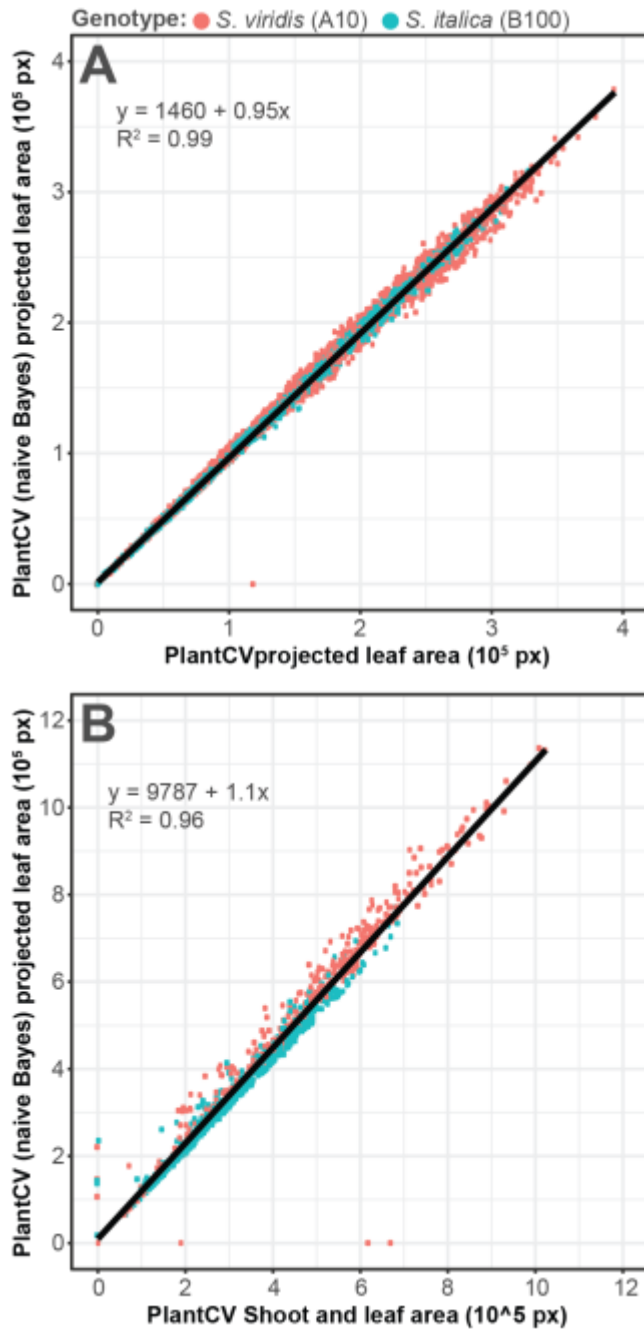
675 **FIGURE 4**

676

677 **Figure 4: Landmark-based analysis of plant shape in PlantCV.**

678 A) Automatic identification of leaf tip landmarks using the ‘acute’ and ‘acute\_vertex’ functions  
 679 (blue dots). B) Geometrically homologous semi/pseudo-landmarks across both the x- and y-axes.  
 680 Semi/pseudo-landmarks identified by scanning the x-axis are denoted by light blue (top side of  
 681 the contour), brown (bottom side of the contour), and light orange (centroid location of  
 682 horizontal bins) dots. Semi/pseudo-landmarks identified by scanning the y-axis are denoted by  
 683 dark blue (left side of the contour), pink (right side of the contour), and dark orange (centroid  
 684 location of vertical bins) dots. The plant centroid is plotted larger in red. C) A representation of  
 685 the rescaled plant landmarks identified in panel (A). White points correspond to the leaf tips. The

686 orange point is the location of the plant centroid. The blue point is the location of the plant  
687 centroid where the plant emerges from the soil. Red lines are the vertical distance from leaf tip  
688 points relative to the plant centroid. D) Analysis of the average scaled vertical distance from each  
689 leaf tip to the centroid diverges in response to water limitation.

690 **FIGURE 5**

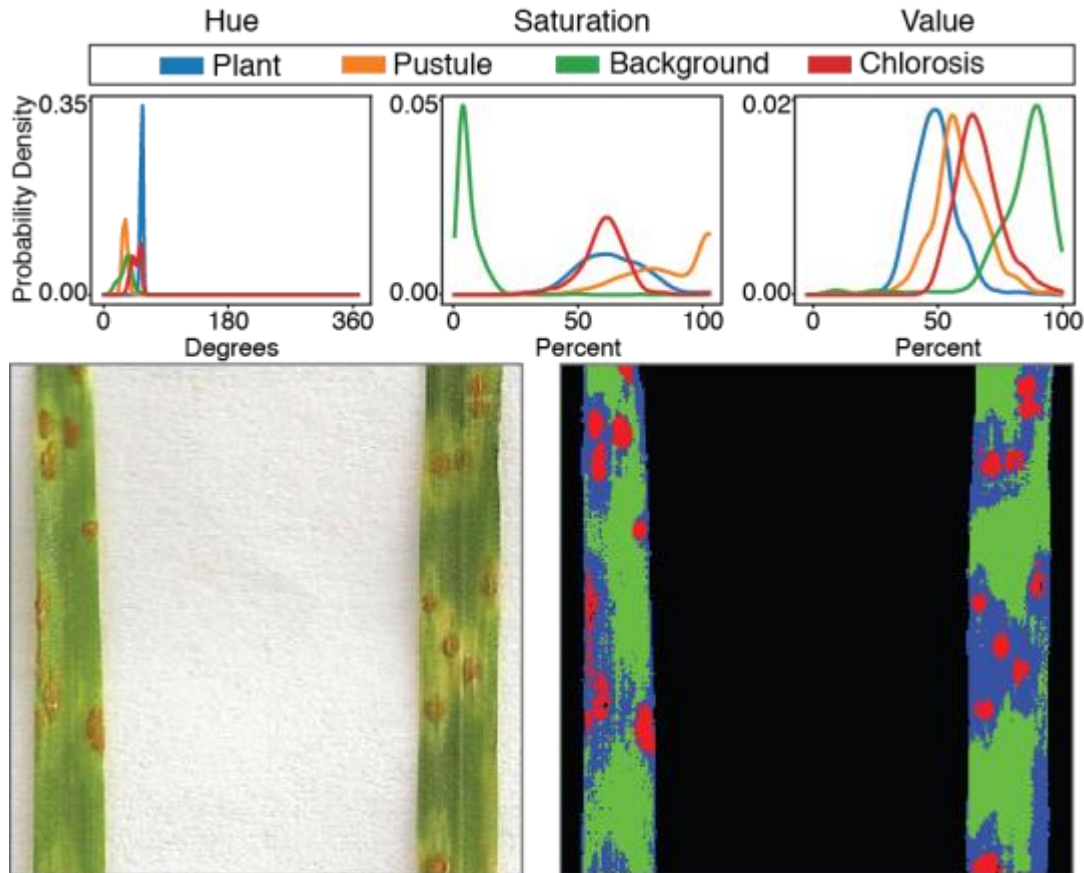
691

692 **Figure 5: Plant segmentation using a naive Bayes classifier.**

693 Correlation between plant area (pixels) detected using thresholding pipelines (Fahlgren et al.,

694 2015) on the x-axis compared to plant area detected using a trained naive Bayes classifier on the

695 y-axis. A) Side-view images. B) Top-view images.

696 **FIGURE 6**

697

698 **Figure 6: Simultaneous segmentation of four feature groups using the naive Bayes**  
 699 **classifier.**

700 An example of the naive Bayes classifier used to assign pixels into 4 classes: background,  
 701 unaffected plant tissue, chlorotic tissue, and wheat stem rust pustules. (Top) Probability density  
 702 functions (PDFs) from the 'plantcv-train.py' script that show hue, saturation, and value color  
 703 channel distributions of four classes estimated from training data. (Bottom) Example of a  
 704 classified image with the original image (left) and merged pseudocolored image (right) with  
 705 pixels classified by the 'naive\_bayes\_classifier' as background (black), unaffected leaf tissue  
 706 (green), chlorotic leaf tissue (blue), and pustules (red).