Leaf Area Estimation by Semantic Segmentation of Point Cloud of Tomato Plants

Takeshi Masuda
National Institute of Advanced Industrial Science and Technology (AIST), Japan
t.masuda@aist.go.jp

Abstract

Growth monitoring is an essential task in agriculture for obtaining good crops and sustainable management of cultivation. Though it is essential, it is also a hard task requiring much labor and working time, and many automation approaches have been proposed. We present an attempt to estimate the leaf area of the tomatoes grown in a sunlight-type plant factory. We scanned tomato plants by an RGB-D sensor that moves vertically to scan one side of the plants from the pathway. We built a point cloud by merging the scanned data, and we segmented it into four classes (Stem, Leaf, Fruit, and Other) based on annotation. With a limited amount of data, we estimated the stem from Stem points, and from the number of Leaf points around the stem, we estimate the leaf area of a specific tomato plant in a plant factory with the relative error of about 20%.

1. Introduction

Plant phenotype is determined by genotype and environment. To obtain good crops, proper management of the environment is necessary. For improving management, growth monitoring is one of the most important tasks in many agricultural fields and plant factories. Many kinds of research have been proposed on phenotyping plants, and most of them are targetting small and simple-shaped plants in the environment in laboratories, artificial-light or completely controlled plant factories, or specialized phenotyping facilities [2, 9, 8, 14].

There are trials of reconstruction of much larger and complex objects. Tan et al. [22] reconstructed tree shapes from multiview images. Their method only requires 10 to 20 images but from 120° to 200° around the tree. They extracted branches and leaves and finally generated graphical models for view synthesis. Instead of preparing many images, Okura et al. [19] used synthesized images from generated 3D models for learning to estimated occluded branches. After learning, their method can estimate the occluded branches from a single shot. Li et al. [13] modeled a tomato plant using Kinect and segmented plant parts, but the plant is potted, small, and simple-shaped.

Figure 1. Target plant factory (left) and scanning system (right).

We are developing a system that facilitates the phenotyping of the real cultivating environment. Within many growth indices, leaf area is important. Measuring the correct leaf area requires hard work. We cut the plant at the root, trim all leaves, spread, flatten, and scan them. This way of measurement is also destructive, and we cannot apply this method to all plants.

In this paper, our targets are tomato plants grown in a sunlight plant factory. There are several technical difficulties. First, plant shape is complex. Tomato plants have compound leaves whose shape is complex. Dense leaves cause a lot of occlusions. In tomato cultivation, side branches are pruned off, and there should be only one stem per plant, but it often happens that petioles are as thick as stems. Second, the plants are grown dense. Each plant is placed in the interval so that photosynthesis is not disturbed but as dense as possible for economic efficiency. The leaves of neighboring plants interfere and are hard to separate. This also limits viewpoints. It is usual that, on each cultivation bench, two

rows of tomatoes are planted. As a result, a plant can be observed only from one side. Third, leaf area measurement is hard and destructive, and only a few portions of ground truth are available.

As mentioned beforehand, tomato plants have a complex shape, and also similar shapes are repeated. This makes it hard to use the structure-from-motion approach that is used in many pieces of literatures. Instead, we used an RGB-D sensor for measuring the plants to model the plant shape as a point cloud. To analyze the plant structure, we applied the semantic segmentation method of the point cloud. After the stem location is estimated, we segment the leaves around the stem, and the number of leaf points around the stem should be correlated with the leaf area. With quite a limited amount of data, we validated our estimation method of the leaf area of a specific tomato plant in a plant factory with a relative error of 20 %.

The outline of our procedure is: first, to scan the tomato plants and generate point clouds (Sec. 2), to apply semantic segmentation (Sec. 3), then to extract the stems from the segmentation result (Sec. 4). In Sec. 5, we explain the process with the real data and finally evaluate the estimation of the leaf area.

2. Point Cloud Generation

2.1. Measurement

Our targets are tomato plants grown in a sunlight-type plant factory (Fig. 1). In such an environment, the plants are grown on raised-floor linear cultivation benches. Two rows of tomatoes are planted on a cultivation bench. Between two benches, there is a pathway on which a pair of heating pipes is laid.

As the RGB-D sensor, we used Intel Realsense D435i [10]. This sensor captures the depth image in addition to the RGB image. Depth is captured by binocular stereo with two infrared cameras attached with an RGB camera. This sensor can work in a bright environment, even in the sunlight, and in a dark environment with the help of a built-in infrared random-dot projector.

The sensor is attached to a slide that moves on a pole vertically driven by a geared motor (Fig. 1), and the pole is placed on a cart that slides on the heating pipes. To measure the target plant, we move the cart in front of it and start capturing and switching on the motor to scan the sensor vertically. The scanning direction is either upward or downward.

The sensor's vendor (Intel) provides GUI software (RealsenseViewer) [11] to control the sensor. The sequence captured by this software is stored as a bag file that is compatible with ROS. We decomposed this file using the librealsense SDK [11], and for each frame, we obtain an aligned pair of RGB and depth images that are considered an RGB-D frame. The intrinsic camera parameters are calibrated beforehand and fixed for all experiments.

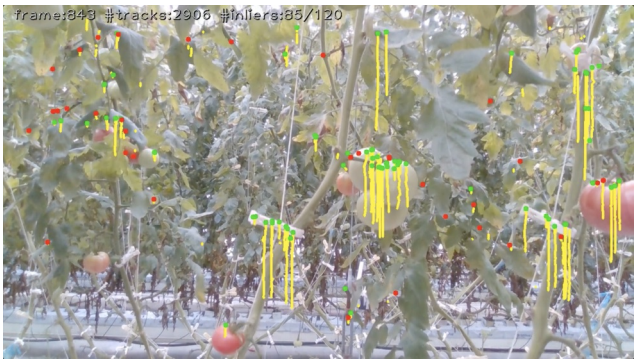

Figure 2. Tracking of the feature points. Green and red dots signify inliers and outliers, respectively. Tracked inlier paths are drawn in yellow.

2.2. Alignment

After scanning a plant, we obtain a sequence of about a thousand RGB-D frames, and we need to align them before merging them to build the total point cloud.

We establish the correspondence by tracking AKAZE feature points [1] on the RGB channels of RGB-D frames because they have more resolution and reliability than the depth channel. From the depth channel, we can determine the XYZ coordinates on each pixel using the intrinsic parameters. From the correspondence of feature points on the RGB channels, we can determine the correspondence of XYZ coordinates which will be used for alignment.

We align each frame to tracks. For each track, we store the feature point's XYZ coordinate values relative to the first frame and the feature vector. A new feature point initializes a new track by its XYZ coordinate values and feature vector. For the successive frames, each feature point corresponds to the track whose feature vector is the closest in the feature space. The 3D rigid transformation is determined by minimizing Cauchy loss of the corresponded XYZ coordinate values eliminating outliers with large alignment errors (> 5 mm). The inlier coordinate values are transformed by the estimated 3D rigid transformation, and each track's coordinate values are updated by the cumulative average of the transformed inlier coordinate values. The feature vector of the track is replaced by the last one matched to the track. This process is repeated for all frames, and for each frame, we can determine a 3D rigid transformation relative to the first frame. The sequence of 3D rigid transformations is optimized so that all transformed feature points of the same track coincide for all tracks.

The sensor applies the automatic gain control. To compensate for this, we estimate the relative gain change to the

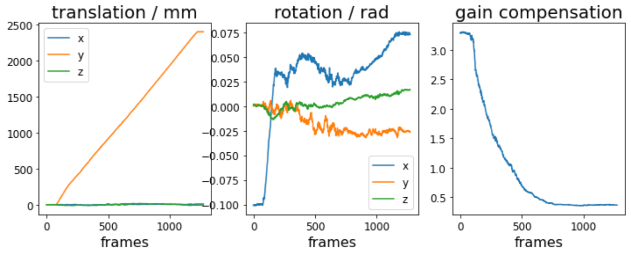

Figure 3. Tracked motion parameters and gain compensation factors.

next frame using the intensities of inlier image point pairs. We linearize the intensity response beforehand (Sec. 5.3). The whole-sequence gain compensation factors are determined by accumulated product from the first frame.

Finally, the point cloud is transformed in the global coordinates: the z-axis is in the hight direction, the y-axis is in the depth direction from the sensor to the plant, and the origin is at the bottom of the sensor position. As a result, for the i-th frame, we obtain a global 3D rigid transformation (camera extrinsic parameters) $\boldsymbol{T}_i$, and a global gain compensation factor g_i.

For acceleration, we split the sequence into batches of 55 frames with 5-frame overlap for parallel processing and connect the batches by seaming the overlapped tracks together between batches.

The sensor (D435i) is equipped with an IMU of a 3D accelerometer and a gyroscope. We don't use the motion information for alignment because the plants are not completely static and the alignment cannot be achieved just by the sensor's IMU. The accelerometer's output is used to align the z-axis orientation to the gravity orientation.

Figure 3 shows an example of the results of alignment. The tracked motion parameters relative to the first frame are plotted in the translation (mm) and rotation (radians) parts of the twist representation [17]. In the sensor's coordinate system, the y-axis is pointing down to the ground. In this plot, almost constantly increasing y-axis translation is dominant, which means that the sensor was going down in nearly constant velocity. The plot of gain compensation factors shows that the sensor's gain was changed greatly during the sequence.

2.3. Merging

Once the RGB-D sequence is aligned, we merge them to construct the total point cloud. Our objective is to measure the leaf area, and we chose to build a nearly uniformly sampled point cloud by voxels.

Assume that the i-th RGB-D frame is converted to a point cloud of global XYZ-RGB values $[\boldsymbol{X}, \boldsymbol{C}] = [\boldsymbol{T}_i \boldsymbol{x}, g_i \boldsymbol{c}]$ using the intrinsic parameters, the global rigid transformation $\boldsymbol{T}_i$ and global gain compensation factor g_i. This point cloud is quantized in voxels whose size is δ according to the global XYZ values $\boldsymbol{X}$. For each voxel V, we determine the signed distance field (SDF) representation [15, 6, 18], which is a set of $[\bar{\boldsymbol{X}}_V, \boldsymbol{N}_V, \bar{\boldsymbol{C}}_V]$, where $\bar{\boldsymbol{X}}_V$ and $\bar{\boldsymbol{C}}_V$ are the averages of $\boldsymbol{X}_V = \{\boldsymbol{X} | \boldsymbol{X} \in V\}$ and $\boldsymbol{C}_V = \{\boldsymbol{C} | \boldsymbol{C} \in V\}$, respectively. The normal $\boldsymbol{N}_V$ is determined by the vector pointing to the voxel center from the average of k-closest points ($k = 5$) of the whole points $\boldsymbol{X}$ from it. This is done very quickly with the k-d tree algorithm [3]. The normal vector's length is normalized, and its orientation is determined so that it is on the viewpoint side. If the angle between the normal angle and the vector to the viewpoint is greater than a threshold ($45°$), the voxel is considered unreliable and eliminated.

The aligned SDFs in the same voxel are integrated to form the global SDF. This process is achieved by incremental averaging of each element of the SDF. For each global SDF element $[\bar{\boldsymbol{X}}_V, \boldsymbol{N}_V, \bar{\boldsymbol{C}}_V]$, we store the number of points, the number of frames and the last frame index integrated into the voxel in addition.

We set the forgetting factor ($= 0.9 \leq 1$) in integrating each SDF that lightens the weight for the preceding SDF. This is introduced to reduce the effect of non-rigid motion of plants that cannot be aligned by a rigid transform. For example, when the forgetting factor is 0.9, the weight for the 7-th frame before is about a half ($0.5 \approx 0.9^7$).

Merging is also parallelized by splitting the frames into batches. The merging process is dependent on the order, and we need to keep the frame order throughout the process.

2.4. Pruning

The merged SDF contains many spurious points, which we need to prune out to generate a clearer point cloud. The following steps are applied in sequence.

The parts observed only by a few frames are unreliable. First, we select only the SDF element whose number of frames integrated into the voxel is greater than the threshold (=5).

The objects that move non-rigidly cannot be aligned by a rigid transformation and cause blur. Like the non-maximum suppression used in Canny edge detection [4], we apply the non-latest suppression. For each voxel, the voxel is preserved if it has the largest last frame index (or, if it is the latest) compared to the voxels in the normal direction, which is quantized in the 26-neighborhood directions.

The SDFs that are far from the surface are not necessary. For each voxel, if the sign of the distance is different from the ones of the neighboring voxels (if the voxel is crossing the surface), the voxel is preserved.

Finally, the isolated points are detected by the DBSCAN clustering algorithm [7] on the SDF points $\bar{\boldsymbol{X}}_V$. The DBSCAN algorithm is a clustering algorithm that finds the cluster connected to the specified number (=5) of 'core' points within the specified neighborhood radius ε (=1.5δ).

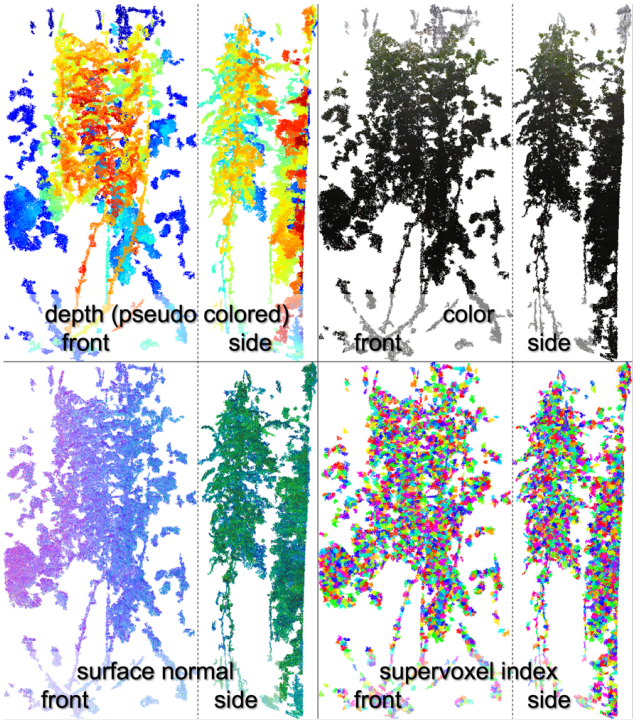

Figure 4. Ortho plot of attributes of a reconstructed point cloud in front and side views.

The points that don't have connections to the core points are detected as the noise points, and in addition, the clusters whose number of points are less than the threshold (=50) are removed.

Figure 4 shows an example of the merged point cloud after pruning in the orthographic projection. The front and side views are the projection on the global x-z and y-z planes. The global y-coordinates represent the depth value which is colored in red for close and blue for far points. 2.2, For this dataset, the point cloud contained 287579 points. The supervoxel will be explained in the next section, and the scale parameter for supervoxel clustering was 20 mm.

3. Segmentation

3.1. Annotation

For segmenting the plant point cloud, we need to give annotation labels. It is too tedious to put labels on all image frames, even with image annotation tools, and also, it is hard to use video annotation tools because tracking occluded objects in a highly cluttered scene tends to fail. Instead, we build our own GUI tool for point cloud annotation (Fig. 5).

It is also impossible to assign labels on all points of point clouds manually. We split the merged point cloud into nearly equal-sized clusters by the mean shift clustering algorithm [5] like "supervoxels", and we put an annotation label to each supervoxel. In the GUI tool, the point cloud

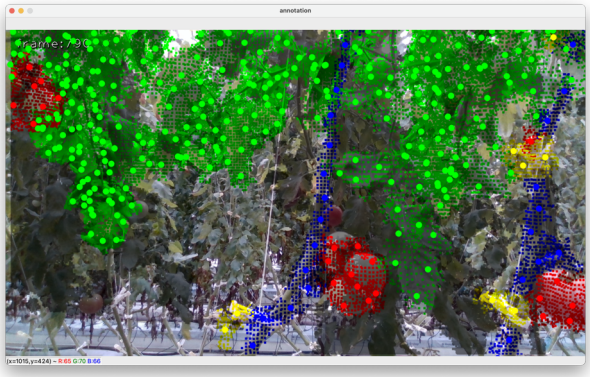

Figure 5. Window capture of the annotation tool. The point cloud is overlaid on the RGB image. The point cloud is segmented in supervoxels, whose center points are drawn as circular dots, and we put labels for each supervoxel by the keyboard input while the mouse cursor is on the dot. Stem, Leaf, Fruit, and Other classes are colored in blue, green, red, and yellow, respectively.

is overlaid on the RGB image, and the center points of supervoxels are shown as circular dots on the image as the control points. These 3-D points are projected on the RGB image using the camera's intrinsic and extrinsic parameters. While the mouse cursor is on a circular dot, the label of the supervoxel is assigned by the keyboard input.

We put the four labels: Stem, Leaf, Fruit, and Other. A tomato plant should have only one stem per plant because unnecessary branches are pruned off in cultivation. Flower clusters and their remain are included in Stem. A tomato leaf is composed of several parts: petiole, rachis, and leaflets, and they are all labeled as Leaf. All fruits are labeled as Fruit regardless of color. Other things than the plant are labeled as Other, which includes clips, ropes, cords, pipes, hooks, chains, boxes, and measurement artifacts.

It is usual in many plant factories that there are two rows of plants on a cultivation bench. Scan from the pathway contains two layers of point clouds: foreground and background. We segment these layers by applying the spectral clustering algorithm [21] with two classes on the supervoxel centers whose y-coordinates are augmented by multiplying 2. We use only the foreground super voxels for annotation.

3.2. Semantic Segmentation

We apply semantic segmentation on the annotated point clouds. We follow a PyTorch implementation [23] of Pointnet++ [20]. In this implementation, there are four abstraction layers for feature extraction, four feature propagation layers, and two convolution layers (Fig. 6). In an abstraction layer, the points in the neighborhood are sampled and grouped to form a feature. The radius of neighborhood r increases twice in the next abstraction layer. The feature is

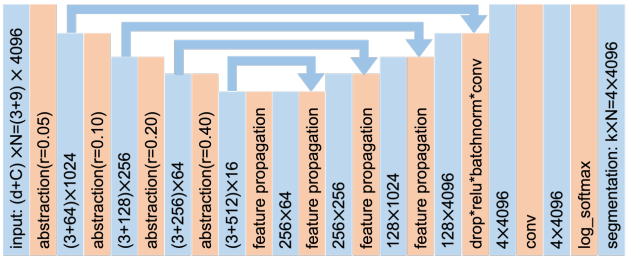

Figure 6. Overview of the architecture. The abstraction layers are composed of sampling and grouping points in the neighborhoods. The feature propagation layers interpolate the previous layer outputs and concatenate with the skipped layer outputs.

interpolated and concatenated in a feature propagation layer with features derived from a skip connection. These layers are made four-fold, and the output features are converted to segmentation by applying two-fold convolution layers.

There are several differences to the base implementation [23]. For semantic segmentation learning, we cut out points in the cubic block whose size is D. We choose a point randomly from the whole scene, then add random shifts in every XYZ axis by the random value in the range of $[-D/2, D/2]$, which becomes the center of the block. Within the block, N (=4096) points are randomly sampled. We applied random rotation in random orientation with the angle at most 15° for augmentation.

We prepare the datasets for learning so that the XYZ coordinate values are in meters to make their values are around 1.0. The RGB values are already normalized between 0 to 1. The input data for learning has 9 channels, which is a concatenation of three components: normalized XYZ coordinates, RGB values, original XYZ coordinates. The normalized XYZ coordinates have the origin at the center of the block. The original XYZ coordinates help classify location-dependent objects, for example, hooks at the top and bare stems at the bottom.

The loss function is the negative log likelihood loss. The weights are determined by $w_c = \mathrm{pow}((\max_{c \in C} h_c)/h_c, 1/3)$, where h_c is the occurrence of the class c among the whole classes [23].

For testing, the block whose size is D is swept with at least $D/2$ overlap to cover the whole point cloud, and the class is determined by voting multiple segmentation results.

4. Stem Extraction

As the result of semantic segmentation, we get the point cloud segmented in four classes. The stem points are not complete, and we need to interpolate. At first, we remove isolated noise points from the stem points by applying the DBSCAN clustering algorithm[7]. Usually, multiple stems are in a point set, and we built a simple GUI for users to specify the required stem (Fig. 7), which is different to the annotation GUI tool. We use $\tilde{\boldsymbol{s}}_l = (\tilde{s}_x, \tilde{s}_y, s_z)_l$ as the control points where l signifies the z-level and $0 \leq l < L$. The user samples a few stem points (usually 1-4 points are enough) by mouse clicks on the x-z projection of the stem points, and the initial values of the x-y coordinates of the control points $(\tilde{s}_x, \tilde{s}_y)_l$ are given by the polynomial interpolation of these points at $(s_z)_l$. From the initial state, we optimize the x-y coordinates of the control points $(\tilde{s}_x, \tilde{s}_y)_l$ by minimizing the error function:

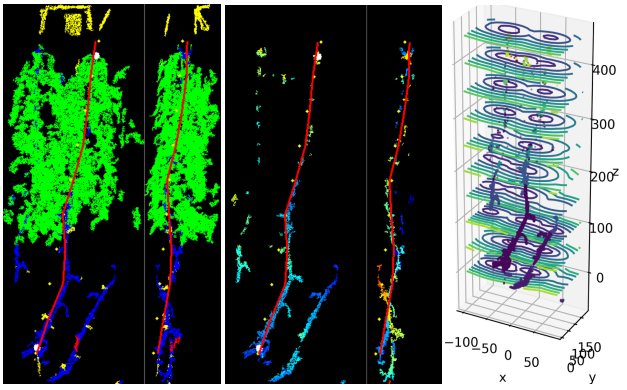

Figure 7. The stem extraction tool. We give the initial state by clicking a few points by mouse, and it is optimized to fit the stem as shown by the red curve overlaid on the segmentation result (left) and on the segmented stem points (middle). The error function is based on the distance to the stem points (right, ticks are in voxels).

$$E(\{(\tilde{s}_x, \tilde{s}_y)_l | 0 \leq l < L\}) = \sum_{0 \leq l < L} \min_{\boldsymbol{s} \in S} d(\tilde{\boldsymbol{s}}_l, \boldsymbol{s}) + w_s \sum_{1 \leq l < L-1} \left\| \tilde{\boldsymbol{s}}_{l-1} - 2\tilde{\boldsymbol{s}}_l + \tilde{\boldsymbol{s}}_{l+1} \right\|^2,$$

where the first term is the sum over the minimum distance from each control point $\tilde{\boldsymbol{s}}_l$ to the stem points $\boldsymbol{s} \in S$, and the second term controls the stiffness of the stem curve. Basically, the stem is estimated to trace the local minimum of the distance function to the stem points (Fig. 7 right). The weight w_s is set as 0.01.

An example of the results is shown in Fig. 7 (left and middle). Figure 7 (right) shows the distance function from the stem points, where $L = 10$ with the interval of $50 \times \delta$. The ticks of the plot are in voxel size $\delta = 5\mathrm{mm}$, and the height range is from 0 m to 2.25 m.

5. Experiment

5.1. Sensing

In the experiments, the sensor settings were: for the stereo module, 1280×720 pixels at 30 fps in 16-bit depth values; for the RGB camera, 1280×720 pixels at 30 fps in 8-bit intensity values (Fig. 8). It took 40-50 seconds for each scan. This sensor's stereo module didn't produce all frames as RGB images, and each scan contains 500-1500 RGB-D frames. In the plant factory (Fig. 1), we captured

Figure 8. The depth and RGB channels of an RGB-D frame.

two rows of cultivation benches. There were 160 plants in total, and we captured 162 sequences.

5.2. Point Cloud Reconstruction

Each sequence of input RGB-D frames (Fig. 8) were aligned (Sec. 2.2) and merged (Sec. 2.3) to form a point cloud. We selected the voxel size as $\delta = 5$ mm, and the average minimum point-to-point distance is about 5 mm. We cut off the points farther than 1200 mm from the sensor for alignment. On a notebook PC with an 8-core CPU, alignment takes about 5 minutes, and merging takes about 20 minutes with parallel processing.

5.3. Gamma Correction

For proper gain compensation, we need a linear intensity response. The sensor has an option parameter named "gamma," but its property is not documented. We did a simple calibration to linearize the intensity response. We printed a very small black-and-white random dot pattern with liner density gradation on a sheet of paper. The dot size was made smaller than the pixel resolution, and each pixel should have an average intensity within the pixel. We captured the pattern for various gamma settings g ($100 \leq g \leq 500$), and assuming that the gamma is a linear function like $\gamma(g) = \gamma_0 + g\gamma_k$, we estimate the parameters so that the recovered intensities $\mathrm{pow}(I, 1/\gamma(g))$ become linear by the least square regression, where I signifies the original image intensity ($0 \leq I \leq 1$). The optimized result was concisely approximated by $\mathrm{pow}(I, 1/(2.5 - g/300))$. The "gamma" option parameter was set as $g = 300$ in this experiment, and we could linearize the intensity using $\gamma(300) = 1.5$.

5.4. Annotation

Using our custom-made GUI tool (Sec. 3.1), we assigned one of the labels of four classes (Stem, Leaf, Fruit, Other) on each supervoxel. Among the whole plants, there were 24 plants whose corresponding true leaf areas are known. We treat the datasets of these plants as the test data. There were multiple plants within a scanned dataset, and we avoid using for training the 72 datasets that have overlap with the test data and 13 bad datasets. The remaining 53 datasets were used for training. We used annotations of 77 datasets in total. It takes 10–20 minutes per dataset for annotation.

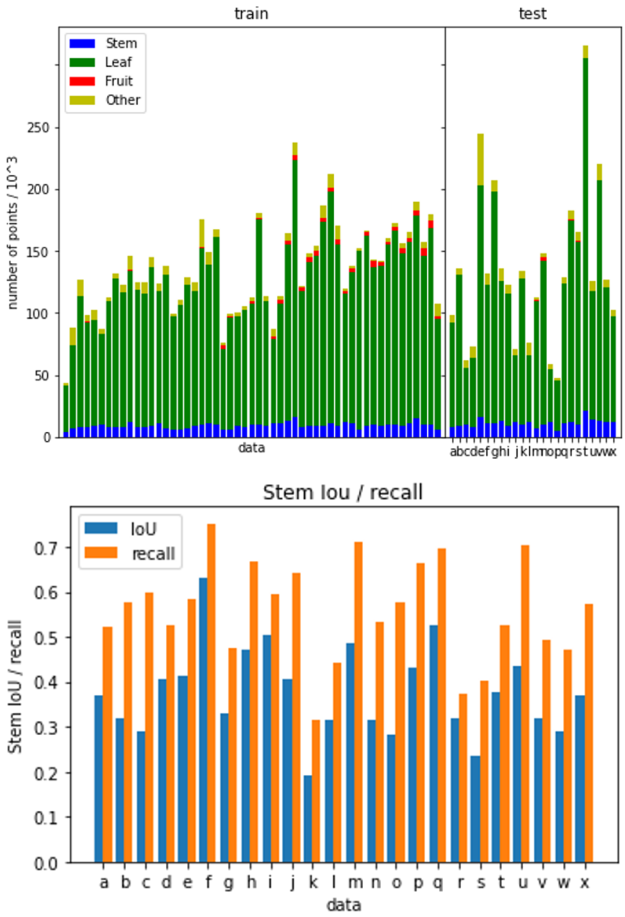

Figure 9. Number of points used for learning (top), stem IoU and recall of each test datasets (bottom).

Estimated \ Truth	Stem	Leaf	Fruit	Other
Stem	0.552	0.39	0.005	0.052
Leaf	0.032	0.962	0.000	0.005
Fruit	0.459	0.242	0.259	0.039
Other	0.247	0.166	0.010	0.577

Table 1. Confusion Matrix. Each row is normalized so that the sum is equal to one.

5.5. Semantic Segmentation

For semantic segmentation (Sec. 3.2), we set the block size $D = 0.5$m, the neighborhood radius of the first abstraction layer $r = 5$cm, number of points $N = 4096$, and the batch size was 16. These parameters were determined empirically. The number of points in the datasets used for learning is shown in Fig. 9 (top). Leaf class is the majority, and in addition to the weights defined in (Sec. 3.2), we doubled the weight for Stem for improving Stem class classification performance, and the weights for each class were [4.72, 1.00, 4.12, 2.75] for four classes respectively.

We applied Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$,

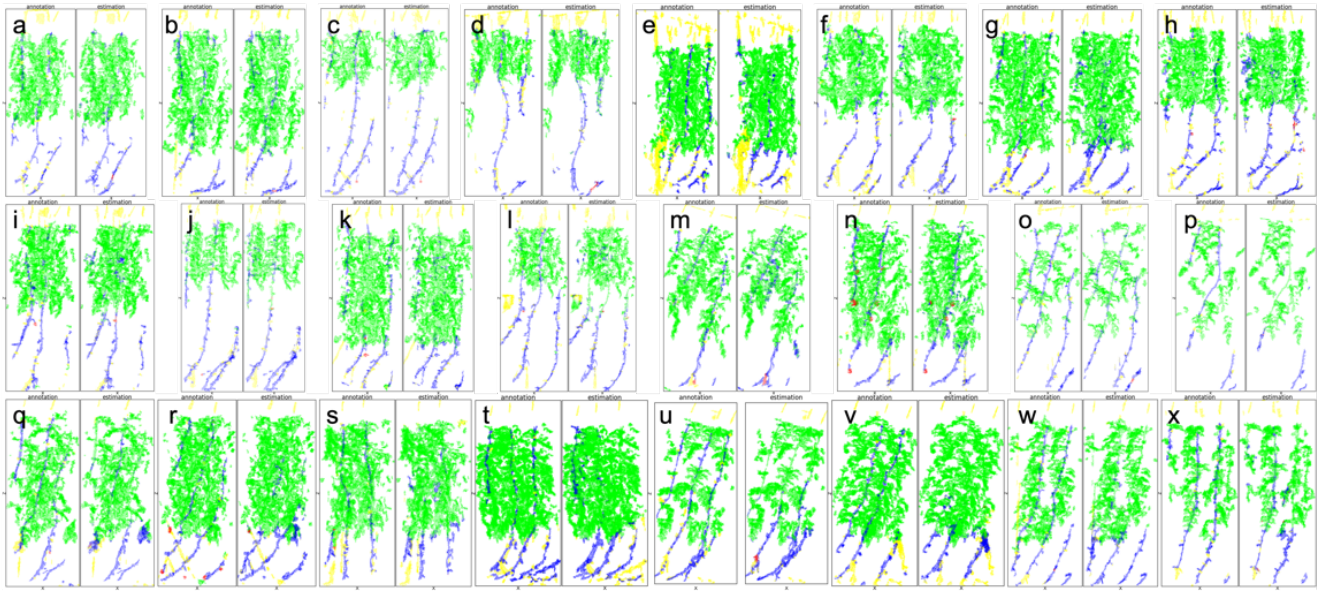

Figure 10. Semantic segmentation results. For each dataset, a pair of plots of annotation (left) and estimation (right) classes are shown.

$\epsilon = 10^{-8}$, $\eta = 10^{-3}$, and the weight decay for the L2 regularization is 10^{-4}. We iterated up to 500 epochs, and this seemed enough for convergence. The IoU and recall of Stem class in each test data are shown in Fig. 9 (bottom), and the total confusion matrix of the test data is shown in Tab. 1. The recall of Stem was about 55%. We use only Stem and Leaf points in the next step, and the performance of Fruit is not important. The cause of the low fruit detection rate can be considered due to the sensor property. The sensor doesn't use the projected random dot pattern in a bright environment, and the passive binocular stereo tends to fail to capture fruits' spherical shapes, which have nearly textureless and glossy surface (Fig. 8).

In Fig. 10, we show the semantic segmentation results with the corresponding annotation. In each segmentation result, the top and bottom parts are segmented well. We could find stems hidden in the leaves in the annotation stage, but they could not be fully detected by semantic segmentation. Especially, the datasets (p), (s), (u), (v), and (w) lost the stem severely.

5.6. Stem Extraction

Figure 11 shows the results of stem extraction. As explained in Sec. 4, the stem was extracted by fitting a curve with a human-giving initial state, and the result depends on it. But even for the dataset whose stem points were lost by the semantic segmentation, the stem is just extrapolated by the stiffness term and pulled by the points of other stems. The evidently failed cases can be found for the datasets (p), (s), (t), (u), (v), and (w) and treated as outliers.

5.7. Leaf Area Estimation

We have only 24 ground truth datasets of leaf area, and we estimate the extent of the target plant statistically. Assuming that the leaves spread homogeneously from the stem, we count the leaf points within the horizontal radius from the stem. For each quantized radius, we plotted the relationship between the ground truth and the number of leaf points and evaluated its linearity by the R^2 factor. It becomes the peak when the radius is equal to 20 cm (Fig. 12). The failed stem extraction in Sec. 5.6 are treated as outliers and not included in this computation. It can be observed that the number of leaf points is linear to the ground truth leaf area, and we can estimate the leaf area from the number of leaf points to some extent.

Finally, using the number of leaf points within 20 cm from the stem, we applied the leave-one-out estimation and compared it to the ground truth. The standard deviation was 0.16 m^2, and the relative error to the average of ground truth was 0.21.

6. Conclusion

In this paper, we presented an attempt to estimate the leaf area of a tomato plant in the sunlight type plant factory. We scanned the plant with an RGB-D sensor, integrated the scanned data as a uniformly sampled point cloud, applied semantic segmentation with Pointnet++, extracted the stem and leaf points, and estimated the leaf area from the number of leaf points around the stem. The final relative error was about 20%, which is comparable to the non-destructive simplified leaf area estimation methods (e.g., estimation method from the leaf width and length) usually used in real

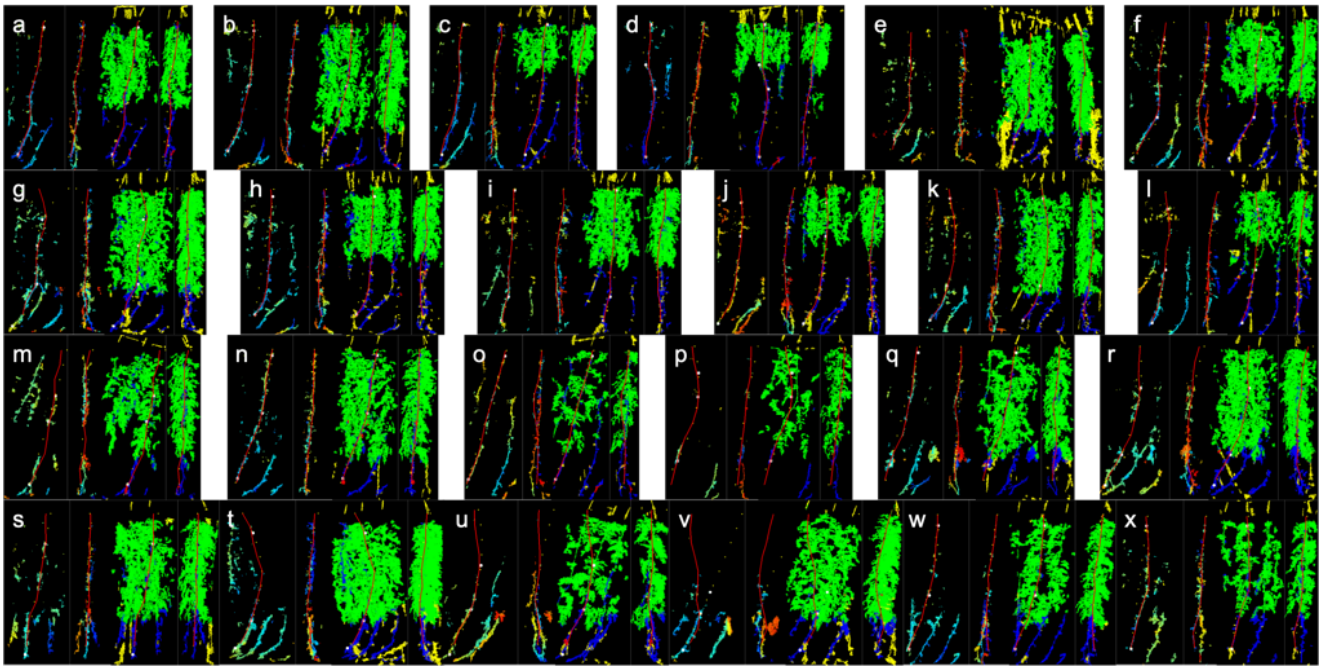

Figure 11. The stem extraction results. Red curves signify the extracted stems. For each dataset, the stem-only front and side view, segmentation front, and view images are shown from left to right.

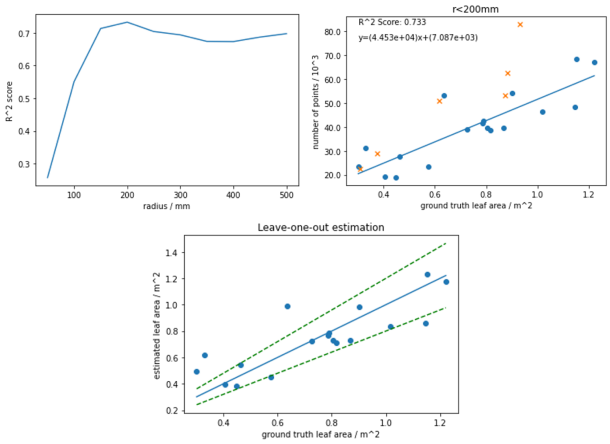

Figure 12. The plot of the R^2-score of linear regression of the number of leaf points in accordance with the radius from the stem (top left), the number of leaf points within 200mm from the stem compared with the ground truth (top right). The outliers (Sec. 5.6) are marked by 'x', and the plot of the leave-one-out estimation with the ground truth (bottom). The green dotted lines signify $\pm 20\%$ error bound.

cultivation sites.

In this research, the amount of available data was quite limited. The ground truth was available only for 24 plants. We scanned 160 plants, but to avoid the overlap between training and test datasets, we used 53 datasets for learning. This amount of data is not enough for end-to-end learning or the intrinsic segmentation for segmenting individual plants. We need more data for better results and reliable evaluation.

We used voxel sampled point clouds for processing. The voxel size was 5 mm, which is not precise, but considering the sensor's resolution and accuracy, this is near the lower bound. There are several possible next steps: for example, we can replace the sensor with newer ones like compact LiDARs [12, 16].

In this paper, we rely on only the point clouds, but images have much resolution than the 3D data. We might be able to improve the performance with multimodal methods using point clouds and images. Our annotation on the point clouds can still be used for image annotation.

The limitation of this work is that we needed to detect the stem to estimate the extent of the plant, and the problem is the stem tends to be highly occluded. For the cases where the stem is completely covered, we need to find another way to identify the plant extent. In the future, we would like to make the system much simpler, smarter and reliable for efficient farming management.

Acknowledgments

This work was financially supported in part by a grant from commissioned project study on "the research project for the future agricultural production utilizing artificial intelligence," Ministry of Agriculture, Forestry and Fisheries of Japan. We thank the National Agriculture and Food Research Organization staff for providing us with opportunities for experiments and the leaf area data.

References

[1] P. F. Alcantarilla, J. Nuevo, and A. Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. In *British Machine Vision Conference*, Bristol, UK, 2013. 2

[2] Pedro Andrade-Sanchez, Michael A. Gore, John T. Heun, Kelly R. Thorp, A. Elizabete Carmo-Silva, Andrew N. French, Michael E. Salvucci, and Jeffrey W. White. Development and evaluation of a field-based high-throughput phenotyping platform. *Functional Plant Biology*, 2014. 1

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 3

[4] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986. 3

[5] D. Comaniciu and P Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002. 4

[6] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. pages 303–312, 1996. 3

[7] Martin Ester, Hans-Peter Kriegel, Jiirg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD'96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, August 1996. 3, 5

[8] Alexander Feldman, Haozhou Wang, Yuya Fukano, Yoichiro Kato, Seishi Ninomiya, and Wei Guo. EasyDCP: An affordable, high-throughput tool to measure plant phenotypic traits in 3d. *Methods in Ecology and Evolution*, 2021. 1

[9] Miki Fujita, Takanari Tanabata, Kaoru Urano, Saya Kikuchi, and Kazuo Shinozaki. RIPPS: A plant phenotyping system for quantitative evaluation of growth under controlled environmental stress conditions. *Plant Cell Physiol.*, 59(10):2030–2038, July 2018. 1

[10] Intel. Intel Realsense Depth Camera D435i. URL: https://www.intelrealsense.com/depth-camera-d435i/. 2

[11] Intel. IntelRealsense/librealsense: Intel RealSense SDK. URL: https://github.com/IntelRealSense/librealsense. 2

[12] Intel. Intel® RealSense™ LiDAR Camera L515. URL: https://www.intelrealsense.com/lidar-camera-l515/. 8

[13] Dawei Li, Lihong Xu, Chengxiang Tan, Erik D. Goodman, Daichang Fu, and Longjiao Xin. Digitization and visualization of greenhouse tomato plants in indoor environments. *Sensors*, 15:4019–4051, 2015. 1

[14] Zhenbo Li, Ruohao Guo, Meng Li, Yaru Chen, and Guangyao Li. A review of computer vision technologies for plant phenotyping. *Computers and Electronics in Agriculture*, 2020. 1

[15] Takeshi Masuda. Registration and integration of multiple range images by matching signed distance fields for object shape modeling. *Computer Vision and Image Understanding*, 87(1-3):51–65, July 2002. 3

[16] Microsoft. Azure kinect dk. URL: https://azure.microsoft.com/en-us/services/kinect-dk/. 8

[17] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994. 3

[18] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE ISMAR*. IEEE, October 2011. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=155378. 3

[19] Fumio Okura, Takahiro Isokane, Ayaka Ide, Yasuyuki Matsushita, and Yasushi Yagi. *Intelligent Image Analysis for Plant Phenotyping*, chapter Image-based structural phenotyping of stems and branches, pages 143–154. CRC Press, October 2020. 1

[20] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, 2017. 4

[21] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. 4

[22] Ping Tan, Gang Zeng, Jingdong Wang, Sing Bing Kang, and Long Quan. Image-based tree modeling. *ACM Transactions on Graphics*, 26(3), July 2007. 1

[23] Xu Yan. Pointnet/Pointnet++ Pytorch. URL: https://github.com/yanx27/Pointnet_Pointnet2_pytorch. 4, 5