Robotic 3D Flower Pose Estimation for Small-Scale Urban Farms

Harsh Muriki[1], Hong Ray Teo[2], Ved Sengupta[1], and Ai-Ping Hu[3]

***Abstract*— The small scale of urban farms and the commercial availability of low-cost robots (such as the FarmBot) that automate simple tending tasks enable an accessible platform for plant phenotyping. We have used a FarmBot with a custom camera end-effector to estimate strawberry plant flower pose (for robotic pollination) from acquired 3D point cloud models. We describe a novel algorithm that translates individual occupancy grids along orthogonal axes of a point cloud to obtain 2D images corresponding to the six viewpoints. For each image, 2D object detection models for flowers are used to identify 2D bounding boxes which can be converted into the 3D space to extract flower point clouds. Pose estimation is performed by fitting three shapes (superellipsoids, paraboloids and planes) to the flower point clouds and compared with manually labeled ground truth. Our method successfully finds approximately 80% of flowers scanned using our customized FarmBot platform and has a mean flower pose error of 7.7 degrees, which is sufficient for robotic pollination and rivals previous results. All code will be made available at https://github.com/harshmuriki/flowerPose.git.**

I. Introduction

Urban farms [1] provide healthy food to local communities and can serve as platforms for education and sustainability. Unlike their rural counterparts, urban farms are usually small in scale and commercially available robotic systems such as the FarmBot [2] have been developed to help automate basic cultivation tasks such as seeding, weeding, and watering.

The FarmBot (Fig. 1) is a scalable XYZ gantry robot that traverses linear rails, covering an arable area of up to 18 m^2 (0.0044 acre). Such a platform makes it feasible to reach and potentially interact with each and every plant growing within its volumetric workspace. We have used the FarmBot with a custom two degrees-of-freedom (DOF) camera end-effector to explore phenotyping for small-scale urban farms (though our methodology and results are applicable to other forms of controlled environment agriculture, such as indoor farming). Specifically, this paper focuses on flower pose (position and orientation) estimation as a canonical phenotyping task, which is critical for robotic pollination.

Pollination is the transfer of pollen grains from the male stamen to the female pistil, either from one flower to another (cross-pollination) or from the same flower to itself (self-pollination). The pistil is located in the center of the flower's petals, extending outwards roughly orthogonal to their surface, and surrounded by multiple stamen (each comprised of a filament supporting a pollen grain). Knowledge of the pose

[1]Georgia Institute of Technology, Atlanta, GA USA (vmuriki3@gatech.edu, vsengupta7@gatech.edu).
[2]Cornell University, Ithaca, NY USA ht526@cornell.edu.
[3]Georgia Tech Research Institute, Atlanta, GA USA ai-ping.hu@gtri.gatech.edu.

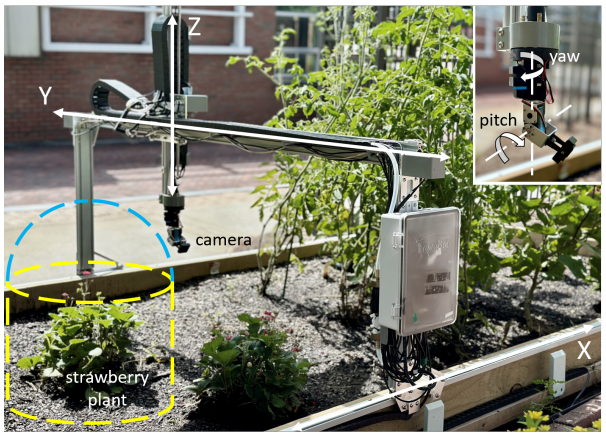

Fig. 1: FarmBot Genesis v1.7 tending a raised urban garden bed with strawberry plants in foreground. Inset: custom 2-DOF camera end-effector. The cylinder/dome plant scanning path for the camera is also schematically shown.

of the flower petals therefore enables access to the stamen and pistil. Robotic pollination can be used when natural pollinators such as bees are declining or unsuitable (as is the case for indoor farms). Yuan et al. [3] considered automated pollination for greenhouse tomatoes using a mobile robot arm platform and binocular vision for flower localization. Smith et al. [4], Strader et al. [5], Yang et al. [6], and Ohi et al. [7] have implemented mobile robot arm platforms (including a multi-arm robot) for pollination of self-pollinating bramble plants in a greenhouse environment. Flower detection is performed based on trained models (including YOLOv8 [8]) using RGB images. Three orientation classes (flower facing left, center and right) were learned in order to estimate flower pose for subsequent contact interaction between pollinator tool and flower center (containing the stamen and pistil). Yang et al. [9] developed a similar approach, detecting forsythia flower pistils directly based on stereo images, while Ahmad et al. [10] worked with watermelons and used nine flower orientation classes (center and eight radially outward-pointing directions). Finally, Hulens et al. [11] developed an autonomous drone for flower pollination that trains its deep learning models on a mixed dataset of real, artificial and computer-rendered flowers at 10-degree angular increments about a vertical axis.

Prior research efforts in robotic pollination use discrete 2D images of flowers to infer orientation, with none providing a quantitative evaluation of angle estimation accuracy. We leverage the capability of a customized FarmBot to reach arbitrary poses within a volumetric workspace enclosing a raised urban garden bed to autonomously generate 3D

models of flowering strawberry plants. A novel method is described to efficiently extract point clouds corresponding to the strawberry flowers for subsequent spatial pose estimation. Our results are evaluated with respect to known ground truth pose values and will be used in our own work on robotic pollination [12] that relies on precise knowledge of flower pose.

II. Method

A. Hardware Platform

We use a customized FarmBot Genesis v1.7 as our data acquisition platform. The XYZ gantry-type robot has been mounted onto a 5 ft $\times$ 10 ft raised garden bed (Fig. 1). A key feature of FarmBot is its universal tool mount (UTM), which acts as the robot's wrist that interfaces with custom tools via 12 pogo pins. The tools are held in place with neodymium magnets. The UTM can be positioned in the FarmBot's workspace using software commands issued through Farmbot's Python API [13].

The FarmBot's control system is composed of a Raspberry Pi [14] and a custom ATmega2560-based microcontroller (Farmduino). The Raspberry Pi runs FarmBot OS, which receives instructions from the FarmBot API via a cloud server. The Raspberry Pi in turn sends instructions to the Farmduino, which enables I/O pins to control different parts of the robot (twelve of these I/O pins connect with the UTM pogo pins).

A custom 2-DOF camera end-effector has been designed to be mounted onto the UTM, enabling yaw and pitch rotations actuated via two high-torque 35 kg waterproof servo motors that are controlled using digital output pins D5 and D6 on the Farmduino. The camera is an Arducam 4K 8MP IMX219 USB Autofocus Camera with 1280 $\times$ 720 video resolution. To interface with the Arducam (via USB) and store recorded images/videos, we have added a separate Raspberry Pi 5 to the hardware set-up.

B. Data Acquisition

Our approach to estimating flower pose is based on using autonomously generated 3D models of the supporting plant. We have chosen flowering strawberry plants as the specimen of interest, with artificial white flowers used to augment our sample size. Based on images taken of the sample plant from multiple camera poses, we use photogrammetry to stitch them together into a spatial model.

We have programmed the FarmBot's camera end-effector path to revolve around the plant in two parts: a cylinder to capture circumferential details topped by a hemisphere to capture top-down details (refer to the schematic in Fig. 1). The camera, set at a fixed manual exposure, continuously records a video while traversing this scanning path. For the cylinder portion, the 2-DOF end-effector is actuated so that the camera points towards the cylinder's center-line, while for the half-dome it points towards the spherical center aligned with radial lines. In order to take advantage of the auto-focus capability of the Arducam, the robot pauses at regular intervals along its scanning path to help ensure sharp and well-lit images of the flowering plant.

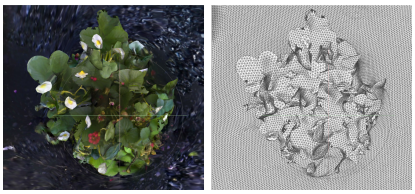

Fig. 2: Rendered (left) and wireframe (right) 3D mesh model of example flowering strawberry plant outputted from Polycam based on photogrammetry.

Once a video sequence of the flowering strawberry plant has been recorded (at 30 frames per second), images are extracted frame-by-frame. As an example: for a plant with diameter 15 inches and height 12 inches, our scanning path takes approximately 20 minutes to execute. This yields approximately 36,000 image frames. The images are separated into N equally sized sequential bins, where N is the number of images to be used for photogrammetry stitching. Each bin corresponds to a segment of the scanning path, which ensures coverage of the entire plant. For each bin, Laplacian filtering [15] is performed on the central area of each image to obtain an average sharpness value, giving us a quality score to help choose the best image in the bin.

C. 3D Model Generation

We have used a commercially available off-the-shelf software tool called Polycam [16] to generate 3D photogrammetry models of the scanned flowering strawberry plants using the N selected images. Through empirical testing, we noticed that 200 images achieved dense point clouds with an average of 62,500 vertices, commensurate with the scale and detail of strawberry plants.

Additional images did not result in better quality point clouds. We used *RAW* mode and enabled the *Sequential* option (since the images we captured were spatially ordered). To further increase the detail of the point cloud, we used Polycam's *Remesh* feature, maximized the *Polygon* count as well as *Texture* resolution, and set the topology to *Uniform*. Fig. 2 shows an example 3D model output from Polycam, which includes the strawberry plant and the underlying soil surface. We note that the flowers in our raised urban garden bed have white petals and yellow pistils.

D. Translating Occupancy Grid Method

There has been extensive research on extracting 3D objects from point clouds, which can be broadly categorized into two main approaches: utilizing the points (or voxels) directly or projecting the points onto 2D planes. 3D deep learning models based on neural networks or transformer model architectures have been employed to work directly with points, like those discussed in Wu et al. [17] , Kolodiazhny et al. [18] and Qi et al. [19]. However, one major limitation of this method is that their accuracy strongly depends on the dataset used during training. If a new class of objects is introduced, the model often requires re-training. This process

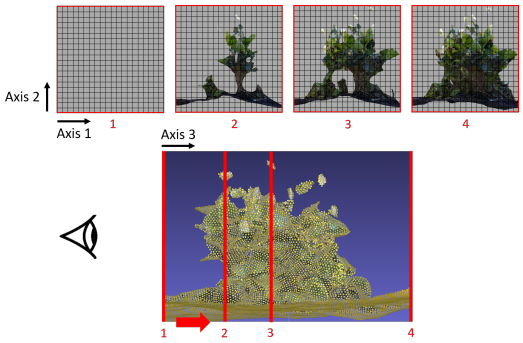

Fig. 3: Schematic of the translating occupancy grid that maps 3D points onto a 2D image, shown for one orthogonal viewpoint direction.

is computationally intensive and can be costly, particularly when dealing with large datasets or complex models. Additionally, they require a very large amount of runtime because of the amount of data they work with.

Algorithm 1 Pseudocode for Translating Occupancy Grid Method

```
TranslatingOccupancyGridMethod()
Data: Pointcloud
Result: Array of 3D coordinates of detected strawberry flowers
orthogonalSides = initialization(Pointcloud)
Pointcloud = threshold(Pointcloud) {Remove ground points using bounding boxes.}
for each side in orthogonalSides do
    2Dimage, 2Dgrid, 2Ddata = get2dImagefrom3dPointcloud(Pointcloud, side)
    boundingBoxes = objectDetection(2Dimage)
    allData = []
    for each boundingBox in boundingBoxes do
        3Ddata = get3DPointcloudfrom2dImage(boundingBox, 2Dgrid, 2Ddata)
        allData.append(3Ddata)
    end
    newPcd = New pointcloud with allData points
end
newPcd ← apply statistical outlier removal and radius outlier removal
boundingBoxes, segments = DBSCAN(newPcd)
return segments
```

Utilizing a projection-based system where point clouds are projected directly onto 2D planes addresses some of these limitations. For example, Boulch et al. [20] employed manually captured 2D snapshots of different views of the point cloud to segment objects. In contrast, Lahoud et al. [21] use a single RGB-D image to estimate the location of an object. Another approach by Yang et al. [22] uses a mini T-net model to extract features from 3D point clouds and project them onto 2D planes. Lyu et al. [23] uses Delaunary triangulation to convert 3D point clouds to 2D images. However, these approaches perform best when the detected object constitutes a large fraction of the point cloud with relatively little external noise. Furthermore, some of these processes are either very laborious to execute [20] or have low accuracy. Moreover, the downstream task of pose estimation necessitates 3D positional information, which cannot be derived solely from 2D images and detections; hence, generating and processing a point cloud is essential.

Algorithm 2 Pseudocode for 3D to 2D Conversion

```
get2dImagefrom3dPointcloud()
Data: Pointcloud, side
Result: Array of image, grid and data
resolution = 10 {Determined via trial & error.}
width, height = 2D_Image.shape
N, M = width + 2 × resolution, height + 2 × resolution {Adding the extra 2 × resolution to allow for processing of edge pixels.}
checkGrid = Array of zeros of shape (N,M) {Indicates whether a particular grid index is filled or empty.}
grid = Array of zeros of shape (N,M)
image = A blank grayscale RGB image
data = {empty dictionary} {Contains key-value pairs of unique identifiers, where each key is associated with the color of a voxel and its 3D location.}
axis1, axis2 = axes {For a given side, coordinate axes along which we perform the operations.}
points = sort(points, axis3)
colors = sort(colors, axis3) {Colors of the corresponding points}
for each (idx,coords) in points do
    coords_norm = normalize coords to [-1,1] along (axis1, axis2) {Map to 2D image.}
    index1 = (coords_norm.axis1 * width + (width / 2)) + resolution
    index2 = (coords_norm.axis2 * height + (height / 2)) + resolution
    if not checkGrid[index1,index2] then
        checkGrid[index1,index2] = 1
        data[idx] = [colors[idx], coords]
        grid[index1,index2] = idx
        image[index1,index2] = colors[idx]
        Toggle all 2D points around [index1, index2] with a radius of ± resolution to 1 in checkGrid
    end
end
return image, grid, data
```

We have developed a novel method for computing 3D bounding boxes around flowers in the point cloud of the strawberry plant generated in the previous section, summarized in Algorithm 1. Six orthogonal snapshots were captured, each representing different perspectives of the plant, using open-source tools such as Open3d [24], OpenCV [25] and Numpy [26]. This allowed us to capture all the visible flowers from every side of the plant. Using these six orthogonal sides, we had six directions along three orthogonal axes: X, $-X$, Y, $-Y$, Z, and $-Z$. We then used a 2D occupancy grid as seen in Fig. 3 that moves along each of these 6 directions. Each square element within the grid was assigned based on the first 3D point it encountered, as illustrated in the progression from location 1 to 4 in Fig. 3. The occupancy grid from location 4 (upon exiting the point cloud) is the resulting 2D image. We note that each detected element in the occupancy grid retains an associated "depth" along the viewpoint direction.

This process is detailed in Algorithm 2. It is used to

generate six 2D color images, illustrated in Fig. 4 as the sides of a cube (we used square occupancy grids). Each grid element is essentially a pixel. An important detail to note is the resolution parameter in Algorithm 2. This parameter defines the radius, in pixels, around each detected (occupied) pixel. All pixels within this radius are colored the same color as the detected pixel. The resolution can be adjusted to modify the 2D image's granularity. Our empirical evidence demonstrates that capturing high-resolution snapshots (e.g., 7000 $\times$ 7000 pixels) of point clouds containing numerous objects, such as plants, produces highly detailed images with minimal pixelation.

Algorithm 3 Pseudocode for 2D to 3D Conversion

```
get3DPointcloudfrom2dImage()
Data: boundingBox, grid, data
Result: Array of 3D points and colors of only flowers
x_min, x_max, y_min, y_max = boundingBox
colors, points = []
for each x in range(x_min, x_max, 1) do
    for each y in range(y_min, y_max, 1) do
        (colors, points) = (colors, points) ∪
                (colors, 3D coordinate at (x,y))
    end
end
return (points, colors)
```

Multiple object detection algorithms such as pre-trained CNNs, color-based thresholding, etc. can be employed due to the modular nature of the process. We decided to leverage pre-trained object detection models like YOLOv10 [27] and Roboflow 3.0 Object Detection [28], fine-tuning them to detect white strawberry flowers. As Algorithm 1 illustrates, each image generates multiple 2D bounding boxes since there are numerous detected flowers. Since Algorithm 2 mapped each pixel to its corresponding 3D coordinate, we can use these 2D bounding boxes to extract the 3D points associated with the detected flowers, as described in Algorithm 3. We then merge all of these 3D points into a unified 3D point cloud and apply density-based spatial clustering of applications with noise (DBSCAN) [29] clustering based on spatial location to segment and filter the flowers from the noise. Through domain knowledge and iterative experimentation, we set the parameters *eps* and *min_points* to 0.01 and 20 respectively. Smaller *eps* leads to fragmented clusters while higher *min_points* reduces sensitivity to noise and excludes some of the smaller clusters. This resulted in the creation of 3D bounding cuboids around the strawberry flowers, as shown in Fig. 5 (bottom left). This method is computationally faster and more flexible than aforementioned work.

E. Flower Pose Estimation

1) Post-Processing: Using the bounding cuboids outputted from the Translating Occupancy Grid Method, we obtain the point clouds for each of the flowers found in the strawberry plant. For each flower, we seek to extract the points in the point cloud corresponding to the petals and the pistil separately, since flower poses are intuited from how the corolla (the petals collectively) opens up and from the position of the pistil. Since the strawberry flowers used have

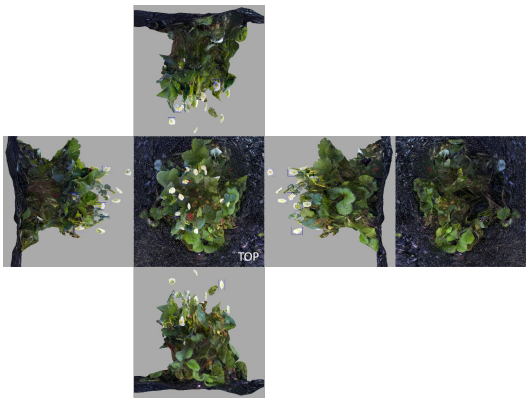

Fig. 4: Images resulting from the translating occupancy grid method applied along six orthogonal views.

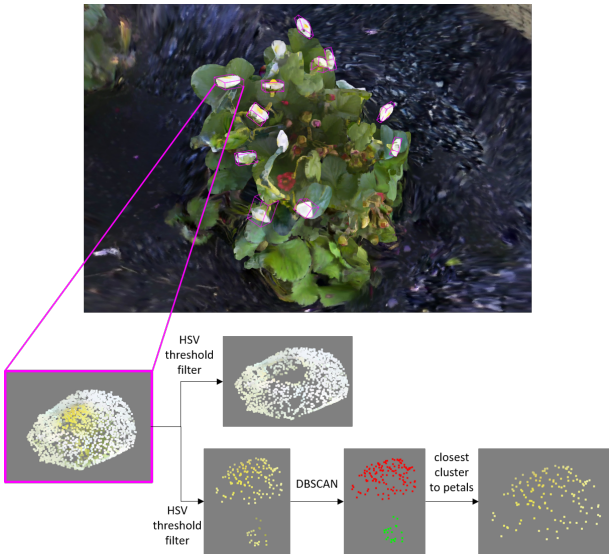

Fig. 5: Illustration of proceeding from bounding cuboids from translating occupancy grid method to flower point cloud (minus pistils, etc.).

distinct white petals and yellow pistils, we accomplish this by converting the point colors into the hue, saturation, value (HSV) color space and applying separate threshold filters for the petals and the pistil. This method successfully segments out the petals. However, due to the yellowish-green color of the flower stems, the threshold filter for the pistil includes some spurious points from the base of the flower.

To address this issue, we perform DBSCAN clustering on the result of the pistil threshold filter, with each of the clusters generated being potential candidates for the pistil. DBSCAN groups together points that are close to each other based on two parameters: a distance metric and the minimum number of points contained in each cluster. These parameters have been set keeping real world flowers in mind. Based on the assumption that the flower pistil should be located in the middle of the petals, we select the cluster whose centroid has the shortest Euclidean distance to the centroid of the petal point cloud obtained in the previous step. Fig. 5 shows the entire process of extracting the petals and pistil for each of the flowers.

2) Shape Fitting: The extracted petal points will be used to determine flower pose. After centering the petal point cloud at its centroid (origin), we proceed to fit quadric surfaces. Three candidate surfaces were investigated: superellipsoid (Eq. 1), paraboloid ($z = (x/a)^2 + (y/b)^2$) and plane ($ax + by + cz = d$).

$$\left[\left(\frac{x}{a}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{b}\right)^{\frac{2}{\epsilon_2}}\right]^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{c}\right)^{\frac{2}{\epsilon_1}} = 1 \tag{1}$$

The motivation for fitting a *superellipsoid* comes from the observation that the petal point cloud often traces out part of a rounded and closed surface which is similar to that of an ellipsoid. Superellipsoids are the generalization of an ellipsoid that can be additionally deformed using two introduced parameters, adding greater shape-fitting flexibility (e.g., see [30]–[32]). A superellipsoid is fit onto the petal point cloud by performing constrained nonlinear least-squares optimization on parameters a, b, c, ϵ_1, and ϵ_2 as well as on Euler angles ϕ, θ and ψ that define the spatial orientation of the superellipsoid. The optimization is performed by the trust region reflective algorithm [33] which solves a system of equations motivated by a first-order optimality condition. The bounds on the parameters are: $0 \leq a, b, c \leq 0.1$ and $0.9 \leq \epsilon_1, \epsilon_2 \leq 1.1$. Parameters a, b and c are bounded to be smaller than 0.1 m to reflect the size of real-life flowers, while ϵ_1 and ϵ_2 are bounded so that the shape of the superellipsoid is similar to one that empirically yields the best result compared to having no bounds or to directly fitting an ellipsoid.

The next step is to determine which of the six directions in the coordinate frame of the superellipsoid (two for each axis) is the pose estimate for the flower. We achieve this by finding the minimum of a, b and c (which, similar to an ellipsoid, describe how much the superellipsoid is stretched in each axis) to find the shortest axis spanned by the superellipsoid. This axis would be parallel to the pose estimate. Finally, to determine the correct direction along that axis, we use the pistil point cloud extracted in the previous step. Since the pistil should always be on the side of the flower where the corolla opens up (opposite the stem), we define a vector originating from the centroid of the entire flower point cloud to the centroid of the pistil point cloud. The direction whose dot product with the aforementioned vector is positive is chosen as the estimated flower pose.

The motivation for fitting a *paraboloid* is the observation that the petals also had a tendency to curve upward along the sides of the flower. The advantage of using a paraboloid over a superellipsoid is the fact that it is directional, making it a trivial task to determine the pose estimate once the fitting is performed. Similar to the superellipsoid, we fit a paraboloid onto the petal point cloud by performing unconstrained nonlinear least-squares optimization on parameters a and b, as well as Euler angles ϕ, θ and ψ. The Levenberg-Marquardt nonlinear least-squares algorithm [34] is used for optimization. Since the paraboloid is directional along the z-axis of its local coordinate frame, the pose estimate is simply the direction of the positive z-axis transformed into the global coordinate frame. This offers an advantage over the superellipsoid where determining which of the six directions to use as the pose estimate is a non-trivial task.

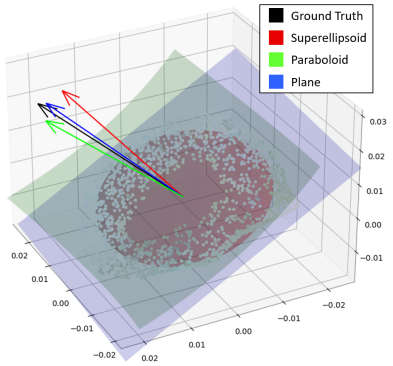

Fig. 6: Example of ground truth pose as well as superellipsoid, paraboloid and plane pose estimates on a flower.

Finally, we experimented with fitting a *plane* because we noticed that the superellipsoid and paraboloid can occasionally be skewed by outliers in the point cloud. A plane is a simpler mathematical model that would be less prone to over-fitting. We perform the fitting by applying principal component analysis to the petal point cloud and selecting the eigenvector corresponding to the smallest eigenvalue. This eigenvector represents the direction with the smallest variation in the point cloud data and thus it would be parallel to the normal of the best-fit plane. Then, just as for the superellipsoid estimate, the pistil of the flower is used to choose the correct direction between the two possible normals (180 degrees apart) to the plane.

For illustration, Fig. 6 shows an example result of fitting all three quadrics onto a petal point cloud. Colored arrows represent the respective three pose estimates. The black arrow denotes ground truth pose.

III. Results

We used the customized FarmBot platform to autonomously acquire seven scans of flowering strawberry plants. This section outlines the process of deriving ground truth and estimated flower poses from scans from our methodology.

A. Ground Truth Specification of Flower Pose

We annotate by hand the "ground truth" pose of each flower in the scans using AWS SageMaker Ground Truth [35] software. This is done by adding oriented bounding boxes with arrows on discerned flowers, as seen in Fig. 7. Cases which cannot be labeled include concealed flowers or ones with heavily distorted corollas, which resulted from lower quality point clouds due to the Arducam's low resolution (read section III-B for improvements). We estimate 5° of human error in ground truth pose values. The number of flowers labeled with the ground truth pose for each plant scan is shown in the second column of TABLE I.

B. Flower Pose Estimation Results

We estimate the pose for a flower using the three quadric surfaces described in section II.E. only if it has been detected

Fig. 7: Example of ground truth flower pose labeling using AWS SageMaker Ground Truth.

using the Translating Occupancy Grid Method and has been assigned a ground truth pose value. The results appear in TABLE I. Overall, 80.3% of ground truth flowers were detected using our method. An edge case occurred with Plant ID 6, where only 50% of the flowers were detected. This is because, during the object detection phase, the model failed to detect some of the flowers due to merging with leaves or being occluded.

We find that the plane performs the best with a mean error of 7.7 degrees, which is a satisfactory result given that the ground truth itself could be off by 5 degrees due to human error. There are two main sources of error, both of which are caused by poor point clouds. Firstly, the petals can curve downward and fuse with the stem, resulting in a mismatch with the mathematical model of a flat plane. Secondly, the white flower petals can fuse with the pistil and take on a yellow color, causing it to be filtered out during petal extraction.

The superellipsoid (the next best method) suffers from the same problems as the plane. This shape also occasionally does not fit properly to the petal point cloud, resulting in significant pose errors. We hypothesize that this is because the algorithm gets stuck in a local minima since the objective function in the least-squares optimization is non-convex.

The paraboloid has the worst performance because it is based on the assumption that the flower petals curve upward. However (as mentioned previously) a number of flowers have downward curving petals, resulting in a pose estimation that can be 180° off from the ground truth pose. The error in these cases is very large and explains the large disparity between the mean and the median.

There are a few cases where a flower is detected by the Translating Occupancy Grid Method but has not been assigned a ground truth pose, labeled as 'No. Extra' in TABLE I. This is because these particular poses are difficult to manually determine due to a deformation in the point cloud or occlusion from foliage, but is detected by the algorithm since there remains a resemblance to a strawberry flower. The cases where a detected "flower" is actually a leaf or part of another flower that has already been detected are

Plant ID	GT	Flowers Found (%)	No. Extra	FP	Super-ellipsoid error	Para-boloid error	Plane error
1	7	6 (85.7)	1	0	14.7°	34.7°	5.4°
2	10	10 (100)	2	1	12.1°	60.5°	9.5°
3	11	9 (81.8)	3	2	26.7°	10.2°	8.0°
4	7	6 (85.7)	5	0	19.5°	30.2°	9.6°
5	9	7 (77.8)	2	0	42.0°	120.3°	7.4°
6	10	5 (50)	4	1	8.0°	172.0°	7.5°
7	7	6 (85.7)	6	0	7.5°	7.9°	4.6°
Total	61	49 (80.3)	23	4	-	-	-
Mean	-	-	-	-	19.3°	57.9°	7.7°
Med	-	-	-	-	8.9°	11.8°	5.9°
Std Dev	-	-	-	-	33.6°	73.3°	5.8°

TABLE I: Flower pose estimation errors for each plant scan captured using the FarmBot platform. GT: ground truth. FP: false positives.

labeled as 'FP' (false positives) in TABLE I.

Cameras aren't perfect. Since many of our errors arise from the relatively poor quality of scans obtained using the Arducam, we manually captured an eighth scan using a smartphone with a higher resolution rear camera that produced a 3D model with 125,000 vertices. This is about twice the resolution of the autonomous FarmBot scans. The improved results for this scan are presented in TABLE II. Note the significantly higher number of point cloud flowers that can be labeled with a ground truth pose, due to better 3D mesh models that facilitate manual pose labeling.

Plant ID	GT	Flowers Found (%)	No. Extra	FP	Super-ellipsoid error	Para-boloid error	Plane error
8	19	15 (78.9)	0	2	17.5	41.6	6.4

TABLE II: Mean flower pose estimation errors for the plant scan captured using a smartphone.

IV. Conclusions and Future Work

Through automated data acquisition using a customized FarmBot, point cloud reconstruction by Polycam, and application of our novel method of extracting flowers from point clouds, we have successfully developed a pipeline to estimate flower pose that finds approximately 80% of flowers scanned using the gantry-style robotic platform. The mean flower pose error for pollination is 7.7 degrees, which is sufficient for robotic pollination and rivals previous results such as Ci et al. [36], Sun et al. [37] and Luo et al. [38].

Using a gantry robot places limits on plant height and farm area, making our system most suitable to smaller urban farms. The parameters for the Translating Occupancy Grid Method are also specific to strawberry flowers and would have to be tuned for other kinds of flowers.

As for future work, we should be able to increase the percentage of flowers found by using semantic segmentation instead of object detection. This is because object detection yields a 2D rectangular bounding box, which includes unwanted noise artifacts whereas semantic segmentation does not. Furthermore, using an RGB-D instead of an RGB camera would require fewer images to generate a good 3D model of the plant, speeding up the data acquisition step.

REFERENCES

[1] United States Department of Agriculture. Urban Agriculture. http://www.nal.usda.gov/farms-and-agricultural-production-systems/urban-agriculture

[2] FarmBot. http://farm.bot

[3] Yuan, T., Zhang, S., Sheng, X., Wang, D., Gong, Y. and Li, W., 2016, November. An autonomous pollination robot for hormone treatment of tomato flower in greenhouse. In 2016 3rd international conference on systems and informatics (ICSAI) (pp. 108-113). IEEE.

[4] Smith, T., Rijal, M., Tatsch, C., Butts, R.M., Beard, J., Cook, R.T., Chu, A., Gross, J. and Gu, Y., 2024. Design of Stickbug: a Six-Armed Precision Pollination Robot. arXiv preprint arXiv:2404.03489.

[5] Strader, J., Nguyen, J., Tatsch, C., Du, Y., Lassak, K., Buzzo, B., Watson, R., Cerbone, H., Ohi, N., Yang, C. and Gu, Y., 2019, November. Flower interaction subsystem for a precision pollination robot. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 5534-5541). IEEE.

[6] Yang, C., Watson, R.M., Gross, J.N. and Gu, Y., 2019, September. Localization algorithm design and evaluation for an autonomous pollination robot. In International Meeting of The Satellite Division of the Institute of Navigation (pp. 2702-2710).

[7] Ohi, N., Lassak, K., Watson, R., Strader, J., Du, Y., Yang, C., Hedrick, G., Nguyen, J., Harper, S., Reynolds, D. and Kilic, C., 2018, October. Design of an autonomous precision pollination robot. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 7711-7718). IEEE.

[8] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO," Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[9] Yang, M., Lyu, H., Zhao, Y., Sun, Y., Pan, H., Sun, Q., Chen, J., Qiang, B. and Yang, H., 2023. Delivery of pollen to forsythia flower pistils autonomously and precisely using a robot arm. Computers and Electronics in Agriculture, 214, p.108274.

[10] Ahmad, K., Park, J.E., Ilyas, T., Lee, J.H., Lee, J.H., Kim, S. and Kim, H., 2024. Accurate and robust pollinations for watermelons using intelligence guided visual servoing. Computers and Electronics in Agriculture, 219, p.108753.

[11] Hulens, D., Van Ranst, W., Cao, Y. and Goedemé, T., 2022. Autonomous visual navigation for a flower pollination drone. Machines, 10(5), p.364.

[12] Kong, C., Qiu, A., Wibowo, I., Ren, M., Dhori, A., Ling, K.-S., Hu, A.-P. and Kousik, S., 2024. Towards closing the loop in robotic pollination for indoor farming via autonomous microscopic inspection. *In preparation.*

[13] FarmBot Developer API. http://developer.farm.bot/v15/docs/web-app/rest-api.html

[14] Raspberry Pi, 2024. Raspberry Pi Foundation. https://www.raspberrypi.com/

[15] OpenCV: Laplace operator. http://docs.opencv.org/4.x/d5/db5/tutorial_laplace_operator.html

[16] Polycam. http://poly.cam

[17] Wu, X., Jiang, L., Wang, P., Liu, Z., Liu, X., Qiao, Y., Ouyang, W., He, T., and Zhao, H., 2024. Point Transformer V3: Simpler, Faster, Stronger. arXiv preprint arXiv:2312.10035. https://arxiv.org/abs/2312.10035.

[18] Kolodiazhnyi, M., Vorontsova, A., Konushin, A., and Rukhovich, D., 2023. OneFormer3D: One Transformer for Unified Point Cloud Segmentation. arXiv preprint arXiv:2311.14405. https://arxiv.org/abs/2311.14405.

[19] Qi, C. R., Yi, L., Su, H., and Guibas, L. J., 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. arXiv preprint arXiv:1706.02413. https://arxiv.org/abs/1706.02413.

[20] Boulch, A., Guerry, J., Le Saux, B., and Audebert, N., 2018. SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks. Computers / Graphics, 71, pp. 189-198. https://doi.org/10.1016/j.cag.2017.11.010.

[21] Lahoud, J., and Ghanem, B., 2017. 2D-Driven 3D Object Detection in RGB-D Images. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia.

[22] Yang, J., Lee, C., Ahn, P., Lee, H., Yi, E., and Kim, J., 2020. PBP-Net: Point Projection and Back-Projection Network for 3D Point Cloud Segmentation. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 8469-8475). IEEE. https://doi.org/10.1109/IROS45743.2020.9341776.

[23] Lyu, Y., Huang, X., and Zhang, Z., 2020. Learning to Segment 3D Point Clouds in 2D Image Space. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Worcester Polytechnic Institute.

[24] Zhou, Q. Y., Park, J., and Koltun, V., 2018. Open3D: A Modern Library for 3D Data Processing. arXiv preprint arXiv:1801.09847. https://arxiv.org/abs/1801.09847.

[25] OpenCV, 2023. Open Source Computer Vision Library. https://opencv.org.

[26] NumPy, 2024. NumPy — Official Website. Available at: https://numpy.org/.

[27] VOLOv10. http://github.com/THU-MIG/yolov10

[28] Roboflow. http://universe.roboflow.com/instituto-politcnico-nacional-ee9gw/polinizador

[29] Ester, M., Kriegel, H.P., Sander, J. and Xu, X., 1996, August. A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd (Vol. 96, No. 34, pp. 226-231).

[30] Lehnert, C., Sa, I., McCool, C., Upcroft, B. Perez, T., 2016. Sweet pepper pose detection and grasping for automated crop harvesting. In 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, pp. 2428-2434. DOI: 10.1109/ICRA.2016.7487394.

[31] Marangoz, S., Zaenker, T., Menon, R. Bennewitz, M., 2022. Fruit mapping with shape completion for autonomous crop monitoring. In 2022 IEEE 18th International Conference on Automation Science and Engineering (CASE), Mexico City, Mexico, pp. 471-476. DOI: 10.1109/CASE49997.2022.9926466.

[32] Giefer, L.A., Lütjen, M., Rohde, A.-K. Freitag, M., 2019. Determination of the optimal state of dough fermentation in bread production by using optical sensors and deep learning. In Applied Sciences, 9(20), p. 4266. DOI: 10.3390/app9204266.

[33] Branch, M., Coleman, T. Li, Y., 1999. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. SIAM Journal on Scientific Computing, 21, pp. 1-23. DOI: 10.1137/S1064827595289108.

[34] Gavin, H. P., 2013. The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems. Available at: https://api.semanticscholar.org/CorpusID:5708656.

[35] AWS SageMaker Ground Truth. http://aws.amazon.com/sagemaker/groundtruth

[36] Ci, J. et al. (2024) '3D pose estimation of tomato peduncle nodes using deep keypoint detection and point cloud', Biosystems Engineering, 243, pp. 57–69. doi:10.1016/j.biosystemseng.2024.04.017

[37] Sun, Q., Zhong, M., Chai, X., Zeng, Z., Yin, H., Zhou, G. and Sun, T. (2023). Citrus pose estimation from an RGB image for automated harvesting. Computers and Electronics in Agriculture, [online] 211, p.108022. doi:https://doi.org/10.1016/j.compag.2023.108022

[38] Luo, L., Yin, W., Ning, Z., Wang, J., Wei, H., Chen, W. and Lu, Q. (2022). In-field pose estimation of grape clusters with combined point cloud segmentation and geometric analysis. Computers and electronics in agriculture, 200, pp.107197–107197. doi:https://doi.org/10.1016/j.compag.2022.107197