The International Society of Precision Agriculture presents the
16th International Conference on
Precision Agriculture
21–24 July 2024 | Manhattan, Kansas USA

# AI-based Pollinator using CoreXY Robot

**Hovannes Kulhandjian[1], Devin Rocha[1], Blake Bennett[1], and Michel Kulhandjian[2]**

[1]Department of Electrical and Computer Engineering, California State University, Fresno, CA, USA.

[2]Department of Electrical and Computer Engineering, Rice University, TX, USA.

Emails: hkulhandjian@csufresno.edu, devrocha01@mail.fresnostate.edu, vivalacrunch@mail.fresnostate.edu, michel.kulhandjian@rice.edu

**A paper from the Proceedings of the**
**16th International Conference on Precision Agriculture**
**21-24 July 2024**
**Manhattan, Kansas, United States**

***Abstract.***

*The declining populations of natural pollinators pose a significant ecological challenge, often attributed to the adverse effects of pesticides and intensive farming practices. To address the critical issue of pollination in the face of diminishing natural pollinators, we are pioneering an AI-based pollinator that utilizes a CoreXY pollination system. This solution aims to augment pollination efforts in agriculture, increasing yields and crop quality while mitigating the adverse impacts of pesticide usage and climate change on pollinator populations. Our approach harnesses deep learning technology and robotics to automate the pollination process, providing a sustainable alternative in regions where pollinators are dwindling. Our research focuses on integrating a flower detection and pollination system into our existing CoreXY weed detection and elimination system. Our methodology involves equipping the CoreXY system with a pollen sprayer, utilizing the integrated camera and trained YOLOv8 deep learning model to pinpoint fruit or vegetable flowers, and then pollinating them by moving the pollen sprayer overtop. This meticulous process is repeated across the entire field, prioritizing plant health and minimizing potential damage by avoiding physical contact. In summary, our AI-based Pollinator with a CoreXY pollination system offers a sustainable and technologically advanced solution to the challenges posed by declining natural pollinators.*

***Keywords.***
*Flower Detection, Flower Classification, Artificial Pollination, Robotic Pollinator, Artificial Intelligence (AI), Precision Agriculture, Farm Robotics, and Machine Learning.*

## I. Introduction

Over the past few years, precision agriculture has become increasingly vital to the farming industry, transforming traditional farming methods [1-4]. Leveraging advanced technologies such as GPS, IoT sensors, drones, robotic arms, and data analytics, precision agriculture allows farmers to optimize resource use, boost crop yields, and mitigate environmental impacts. These technologies facilitate real-time monitoring and management of crops, soil, and weather conditions, resulting in better decision-making and more efficient farm operations. Moreover, precision agriculture promotes sustainable farming practices by reducing waste, minimizing pesticide and fertilizer usage, and conserving water. As global food demand continues to grow, the adoption of precision agriculture is essential for meeting this demand while ensuring the sustainability of farming practices.

Amidst the growing need for artificial plant pollination, driven by both an increased demand for crop production and a decline in natural pollinator populations, researchers are exploring innovative solutions. In [5], Hiraguri *et al.* introduce an autonomous drone-based pollination system tailored for tomato cultivation. The system utilizes an artificial vibrator to disperse pollen, aiming for its transfer onto flower stigmas. However, the system is limited to self-pollination applications, necessitating further advancements to enable cross-pollination. Consequently, there is a pressing need for the development of efficient flower detection and pollination systems. This includes the integration of deep learning models for flower identification and the implementation of sophisticated systems for targeted artificial pollination. By synthesizing recent research and development endeavors, this paper seeks to elucidate the current state-of-the-art in artificial pollination technology and its potential implications for sustainable agriculture.

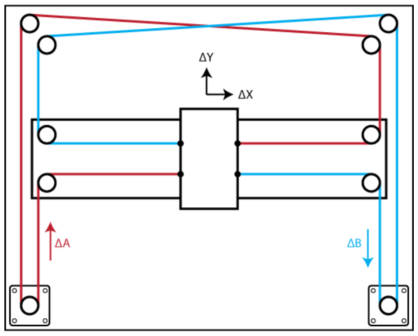
In this paper, we develop a flower detection and pollination system that is based on the current CoreXY weed detection and elimination system that we have developed [2]. The system is installed on our newly developed Smart Agricultural Robot bulldog (SARDOG) [4]. Utilizing the CoreXY architecture, favored in 3D printing and CNC devices for its precision and responsiveness, the system utilizes two Nema 17 stepper motors within a square aluminum frame, driven by DM542 microstep motor drivers. The Raspberry Pi 5 replaced the previous model for faster processing during flower detection while maintaining a quick response from the pollination system. The flower detection component utilizes the Ultralytics YOLOv8 computer vision deep learning model, coupled with the Raspberry Pi 5 and ELP Sony USB Camera. This hardware-software integration enables the detection of flowers in images, guiding the subsequent actions of the pollination system. The coordinate system program translates pixel coordinates from the flower detection model into stepper motor steps, facilitating precise positioning of the pollen sprayer over any detected flowers.

Training the flower detection computer vision model involved a trial-and-error approach, utilizing a combination of three crop flower datasets [6-8], which is modified to contain cucumbers, strawberries, and cherry tomatoes. Training sessions were conducted in a Google Colab environment for enhanced performance and reduced training time. The model effectively detects and classifies flowers, aligning with the primary objective of the project. Integration with hardware involved meticulous wiring and configuration of stepper drivers and motors, adhering to specifications outlined in the DM542 datasheet. After hardware and software were integrated, extensive testing and tuning were conducted, ensuring optimal performance of the system. The final phase involved the integration of the CoreXY system into the SARDOG robot, marking the successful culmination of the project.

Through a synthesis of recent research and development efforts, this paper elucidates the current state-of-the-art in artificial flower pollination system development, underscoring its potential implications for sustainable agriculture. The following sections provide detailed insights into the system's design, development, and performance, offering valuable contributions to the field of agricultural robotics and artificial pollination practices.

## II. CoreXY Pollination System Development

The original linear ball screw system design was upgraded to a sophisticated dual belt system capable of XY movements. The CoreXY configuration, favored in 3D printing and etching devices for its precision and responsiveness, was chosen. Unlike the singular Nema 23 stepper motor driving the ball screw actuator, the CoreXY system utilizes two smaller Nema 17 stepper motors. Constructed within a square aluminum frame, the CoreXY system includes idler pulleys, 30-tooth pulleys, GT2 timing belts, and a gantry system facilitating movement. The Nema 17 stepper motors are driven by DM542 motor drivers, which support a micro step of 400. Additionally, the Raspberry Pi 4 model B was replaced with the Raspberry Pi 5 for faster processing during flower detection while maintaining a quick response from the pollination system. The frame features an extrusion with a central opening accommodating a gantry capable of sliding from end to end. Perpendicular gantries at the extrusion ends enable forward/backward movement in the Y direction, while the gantry on the extrusion slides in the X direction. Gantry movement is governed by simple kinematic equations: equal rotation in opposite directions moves the middle gantry forward/backward, equal rotation in the same direction moves it left/right, and single motor rotation causes diagonal movement.



**Equations of Motion:**

$$\Delta X = {}^{1}/_{2}\,(\Delta A + \Delta B), \quad \Delta Y = {}^{1}/_{2}\,(\Delta A - \Delta B)$$

$$\Delta A = \Delta X + \Delta Y, \quad \Delta B = \Delta X - \Delta Y$$

**Fig. 1.** CoreXY Belt Configuration and Kinematic Equations.

## III. Detection System Development

### A. Theoretical System Overview

The flower detection and pollination system design consists of the following hardware components:

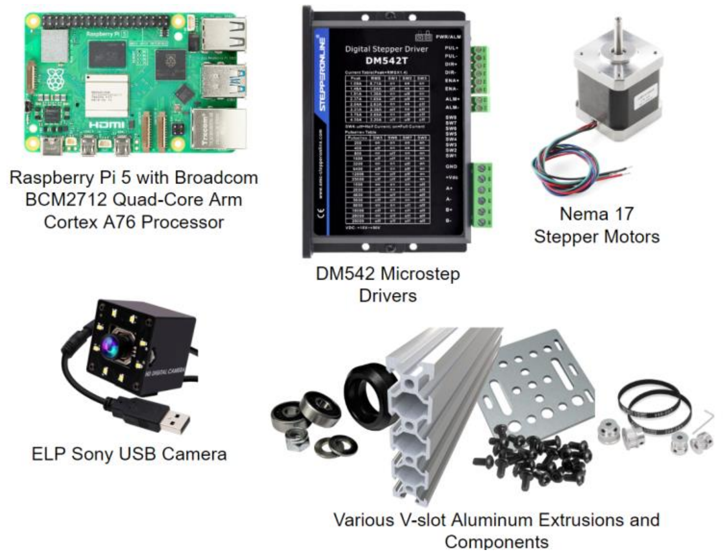

Raspberry Pi 5 with Broadcom BCM2712 Quad-Core Arm Cortex A76 Processor
Digital Stepper Driver DM542T
DM542 Microstep Drivers
Nema 17 Stepper Motors
ELP Sony USB Camera
Various V-slot Aluminum Extrusions and Components

**Fig. 2.** Detection Hardware.

In the flower detection and pollination system, key hardware components include the Raspberry Pi 5 computer and the ELP Sony USB Camera. Software crucial for flower detection comprises the Ultralytics YOLOv8 computer vision deep learning model, Ultralytics libraries, Python scripts, and a combination of three crop flower datasets [5-7].

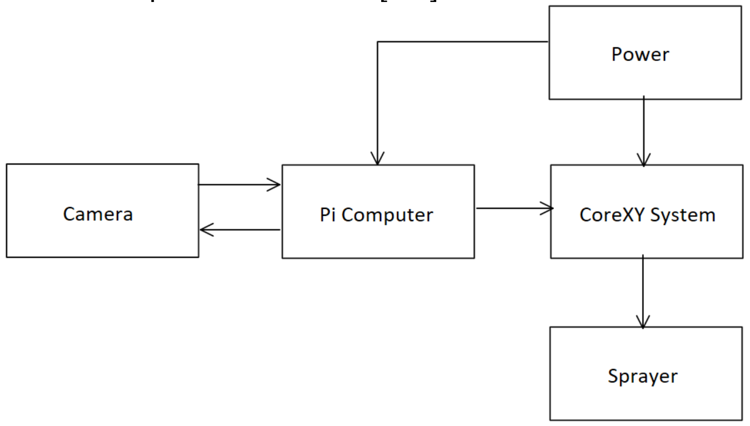


**Fig. 3.** Generic System Block Diagram.

The flower detection and pollination system design comprises several components outlined in the block diagram above. At the core is the Raspberry Pi 5 computer, serving as the central processing unit. The camera interfaces with the Raspberry Pi 5 to capture images of the ground below, processed by scripts on the Pi. The Pi computer communicates with the Stepper drivers in the CoreXY system, driving them while processing images. SARDOG accommodates two 24V batteries connected to a central 24V output, along with 12V and 5V step-down converters. The Raspberry Pi 5 utilizes the 5V output, while the stepper drivers are powered by the 24V central output. Once integrated, the Raspberry Pi 5 captures images of the ground below, processes them, and guides the CoreXY system to position the pollen sprayer over flowers for pollination.

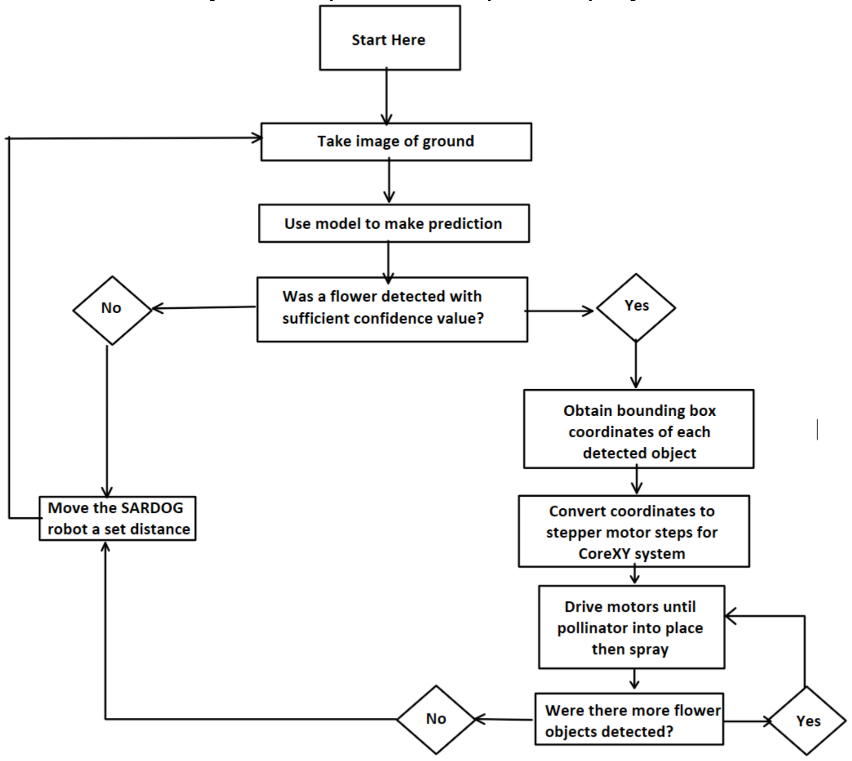

Start Here
Take image of ground
Use model to make prediction
Was a flower detected with sufficient confidence value?
No
Yes
Obtain bounding box coordinates of each detected object
Convert coordinates to stepper motor steps for CoreXY system
Drive motors until pollinator into place then spray
Were there more flower objects detected?
Move the SARDOG robot a set distance
No
Yes

**Fig. 4.** System Behavior Flowchart.

The flowchart above outlines the operation of the flower detection and pollination system. Upon initiation, it captures an image of the ground and utilizes the trained YOLOv8 computer vision model for flower detection. If no flowers are detected, it repeats the process of image capturing and processing. If flowers are detected, the system retrieves the bounding box coordinates for each flower instance. These coordinates are then translated into motor steps to position the pollen sprayer over the flower. After pollination, the system proceeds to the next detected flower until all are pollinated. Upon completion, the process begins again.

## B. Initial Hardware and Software Testing

The initial stage of the project was primarily setting up and researching the various components and software that were going to be used for the flower detection and pollination system. A large portion of the early setup and testing was becoming familiar with the Raspberry Pi computer, setting it up with the YOLOv8 software, testing the GPIO, and testing the stepper motors and drivers.

The first task was setting up the Raspberry Pi with YOLOv8 software and libraries. Once they were installed, basic programs were created to simultaneously test the software and become more familiar with it. For the purposes of early testing, a pre-trained YOLOv8 model (trained on the COCO dataset) was utilized to make testing the libraries possible while the actual flower detection model was being trained.

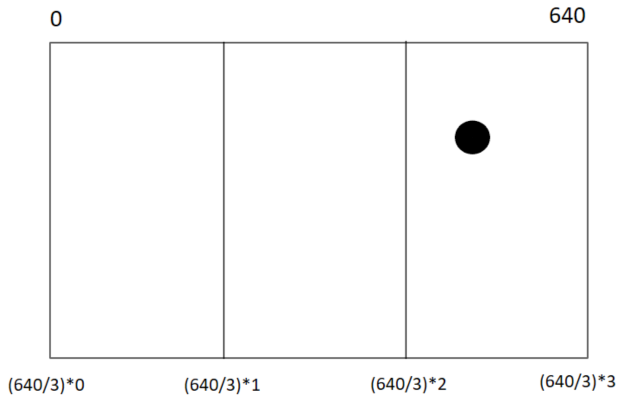


**Fig. 5.** Preliminary Coordinate System Choice.

When first experimenting with the YOLOv8 pre-trained model and Raspberry Pi GPIO, a very basic version of the coordinate system was created so that a better understanding of the software could be obtained. This also enabled basic testing to be performed on the stepper motors. A basic program was written that activated an LED on the left, middle, or right side of a breadboard depending on where a smartphone object was detected in the coordinate system. Using a smartphone was selected arbitrarily to allow for ease of testing while waiting for the actual flower detection model to be trained. This was the first step towards creating the much more advanced version of the CoreXY driver coordinate system that was implemented in the final product. This LED test was eventually modified into a stepper motor test, which allowed for the creation of a rudimentary detection and response system. This program allowed for testing of the stepper motors and drivers with the Pi 5 GPIO, as well as developed a better understanding of how they would work with the final detection model.

## C. Flower Dataset Setup and Formatting

During machine learning and computer vision training, ample data is essential for model training and testing. Many datasets were evaluated before selecting the three datasets that would be combined to serve as a fit for this project [5-7]. This dataset comprises 1584 images featuring three different types of crop flowers.

| Class | Species | Type |
|---|---|---|
| 1 | Cucumber Flower | Crop |
| 2 | Strawberry Flower | Crop |
| 3 | Cherry Tomato Flower | Crop |

**Table 1.** Dataset Configuration.

Table 1 displays the class numbers assigned to each crop flower category that is used in the configuration file for training the flower detection model. The data was split into training and validation sets with a ratio of 82/18 to maximize training data retention. Approximately 1300 images were allocated to the training folder, and 284 images were used for model validation.

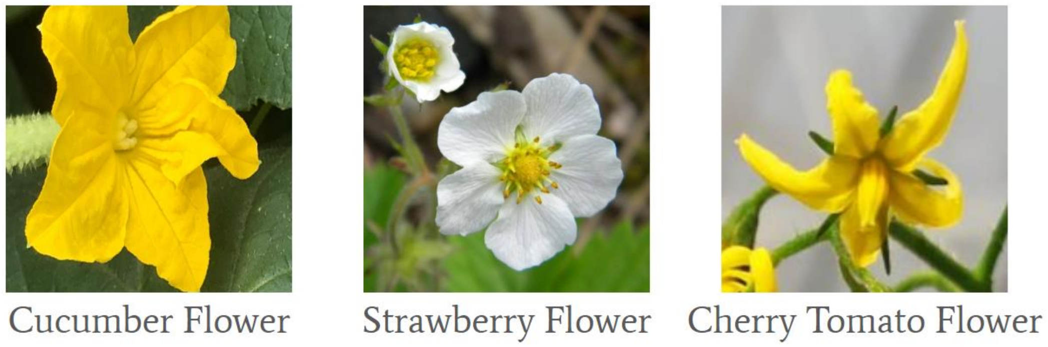


**Fig. 6.** Dataset Flower Examples.

Figure 6 above showcases a sample image from each class in the dataset, each category containing a few hundred images of that flower type. The images feature the three main classes Cucumber Flowers, Strawberry Flowers, and Cherry Tomato Flowers detailed in Table 1. Each image in the dataset is accompanied by a corresponding .txt file containing multilabel annotations for every flower. These annotations include bounding box coordinates and flower class. These datasets were already available in YOLOv8 format, so minimal augmentation was necessary to combine them into one dataset for the flower detection model.

A MATLAB script was created to process the labels and images from each dataset and convert them into one dataset, with class 0 being assigned to cucumbers, class 1 being assigned to strawberries, and class 2 being assigned to cherry tomatoes. Once the labels were augmented to reflect the correct class number, the dataset was compressed into a zip file, and uploaded to Google Drive. The data was now prepared for training using the YOLOv8 computer vision model.

## D. YOLOv8 Computer Vision Training

Training attempts were conducted in a Google Colab environment for enhanced performance and reduced training duration. Training trials in Google Colab utilized the dataset uploaded to Google Drive, with the class configuration set to 3 unique flower classifications. The training was conducted using an NVIDIA Tesla A100 PCIe, with parameters adjusted to 300 epochs, an image size of 640x480, and a patience value of 25 epochs. The main training session was terminated at 197 epochs due to a lack of performance improvement, with the best performance being selected from epoch 172. This early termination allows for the trained model to avoid overfitting to the flower training data.

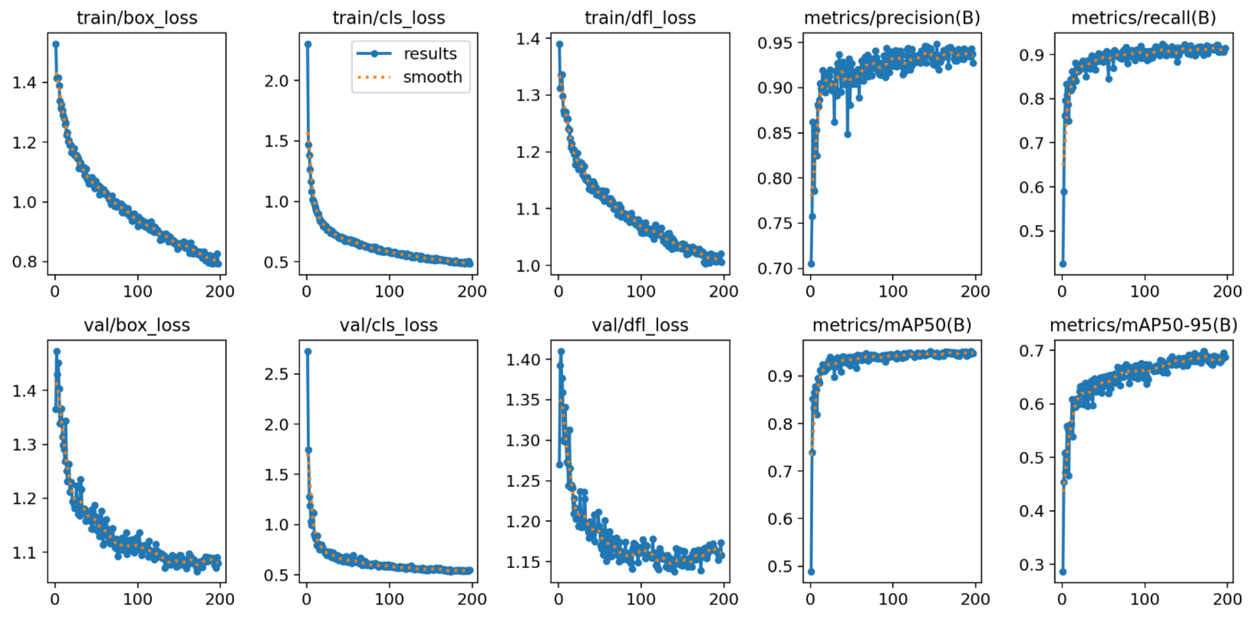


**Fig. 9.** Training and Validation Results.

The presented results illustrate the performance metrics of the trained deep learning model over the course of its training. At 197 epochs (the end of each graph), the model demonstrates convergence into a stable state for the provided flower data. The training algorithm determined that the model was most stable at epoch 172. This model performs well by all metrics, achieving a mAP50 accuracy rating of approximately 94%. This level of performance is ideal for the flower pollination system, allowing for accurate and precise detection and pollination of flowers.

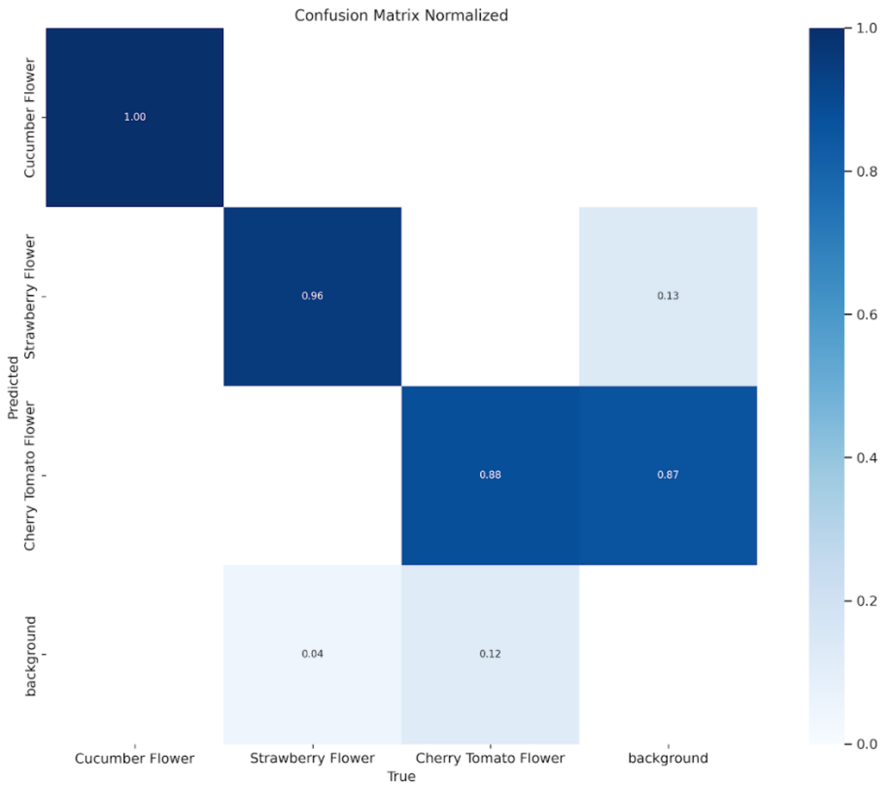


**Fig. 10.** Normalized Confusion Matrix.

The normalized confusion matrix in Figure 10 above visually represents the deep learning model's classification predictions of the flowers compared to the actual classifications of the flowers. The diagonal of the matrix represents the correct predictions, so it can be seen from the data above that the model is performing very well, with minimal predictions outside the diagonal.

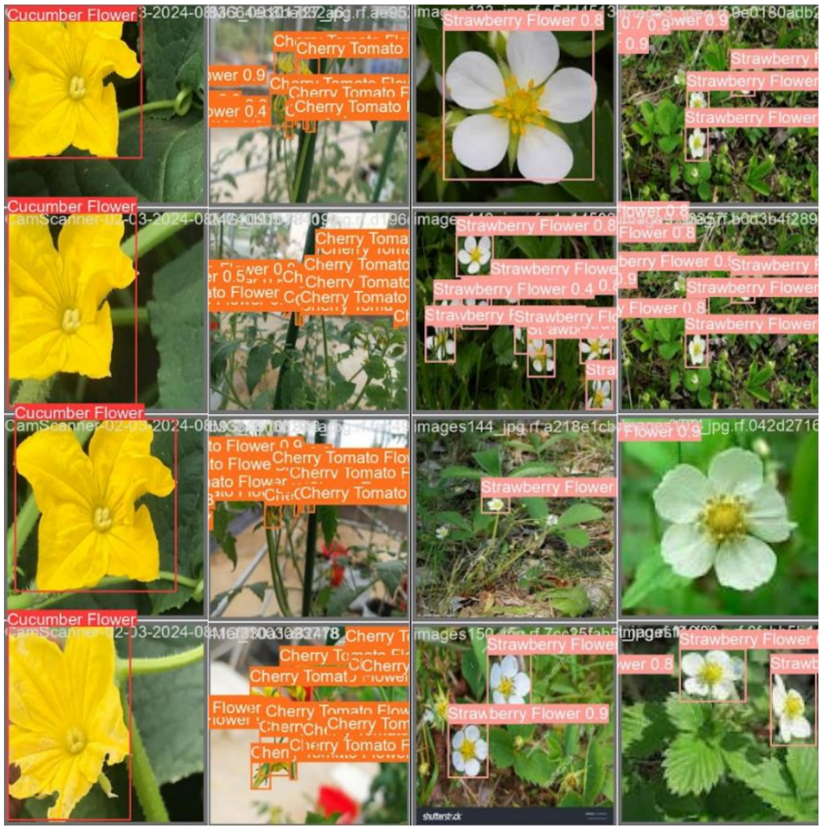


**Fig. 11.** Validation Batch Example.

The validation process is a crucial training step where the model refines its weights based on validation accuracy. The provided example validation batch illustrates multiple predictions made by the trained model. It predicts the location of flowers in the image, draws bounding boxes around them, creates a flower segment, and assigns a confidence value indicating its certainty about the flower classification. This detection capability is invaluable for guiding the pollen sprayer, and the bounding box coordinates play a pivotal role in guiding the hardware. A detailed discussion of how this is managed will follow in the subsequent sections.

## E. Coordinate System Program

Having trained the flower detection computer vision model, the next significant task was integrating it with both software and hardware. Initially, the programs were developed for the Raspberry Pi 4 Model B. However, with the availability of the Raspberry Pi 5, which offers significantly faster performance, the code was later adapted for this platform. Due to differences in GPIO hardware mappings between the two Raspberry Pi models, the Python libraries were incompatible, necessitating adaptation of the coordinate system program to utilize the gpiozero library. The primary objective of the coordinate system was to synchronize the hardware, capable of moving a sprayer anywhere in an XY plane, with the flower detection model, which provides bounding box coordinates for detected flower objects. This was achieved by breaking down the image into its pixels and the pollen sprayer XY plane into stepper motor steps.

$$Spp = \frac{S_{\max}}{P_{max}}$$

$$\begin{gathered}\textit{Where } Spp \textit{ is the Steps per pixel ratio}\\ \textit{Where } S_{\max} \textit{ is the maximum steps}\\ \textit{Where } P_{max} \textit{ is the maximum pixels}\end{gathered}$$

**Fig. 12.** Derived Steps Per Pixel Equation.

The "Steps Per Pixel" ratio (SPP) illustrated above is determined by dividing the maximum number of stepper motor steps the pollen sprayer can travel in one direction by the total number of pixels in the image. Since the image size is 640x480, and the maximum number of steps the pollen sprayer can travel varies for the X and Y directions of the CoreXY, a unique SPP ratio is calculated for each direction, X and Y.

$$d = |P_T - P_c|$$

$$\begin{gathered}\text{Where } d \text{ is pixel distance}\\ \text{Where } P_T \text{ is target pixel}\\ \text{Where } P_c \text{ is current pixel}\end{gathered}$$

**Fig. 13.** Derived Pixel Distance Equation.

Given that the pollen sprayer begins at a known location within the XY plane, the number of steps required to move the pollen sprayer to the desired location in the image can be calculated by determining the absolute difference between the current pollen sprayer pixel location and the target flower pixel location.

$$S = Spp \cdot d$$

$$\begin{gathered}\textit{Where } S \textit{ is steps}\\ \textit{Where } Spp \textit{ is Steps per pixel ratio}\\ \textit{Where } d \textit{ is pixel distance}\end{gathered}$$

**Fig. 14.** Derived Steps Equation.

The coordinate system program determines the required stepper motor steps to position the pollen sprayer at the target location by multiplying the calculated pixel distance value with its corresponding SPP ratio. These equations played a crucial role in the development of the coordinate system program. The program initiates by importing the YOLOv8 libraries and loading the trained flower detection model. It then iterates through an infinite loop, processing the results of the flower detection model. The X and Y pixel coordinates of the bounding boxes are stored in variables. Subsequently, the coordinate system calculates the pixel distance from the pollen sprayer to the target flower in both the X and Y directions. Utilizing the previously discussed equations, the coordinate system converts the pixel distance to steps for the X and Y directions. These step values are then transmitted to the Stepper Driver program, managing the actual movement of the pollen sprayer. Once the sprayer is in position, the Sprayer Driver program is invoked to control the pump. The coordinate system repeats this process for each detected flower, ensuring to return of the sprayer to its initial location at the conclusion of the process. This cycle continues indefinitely until the flower detection and pollination system is deactivated.

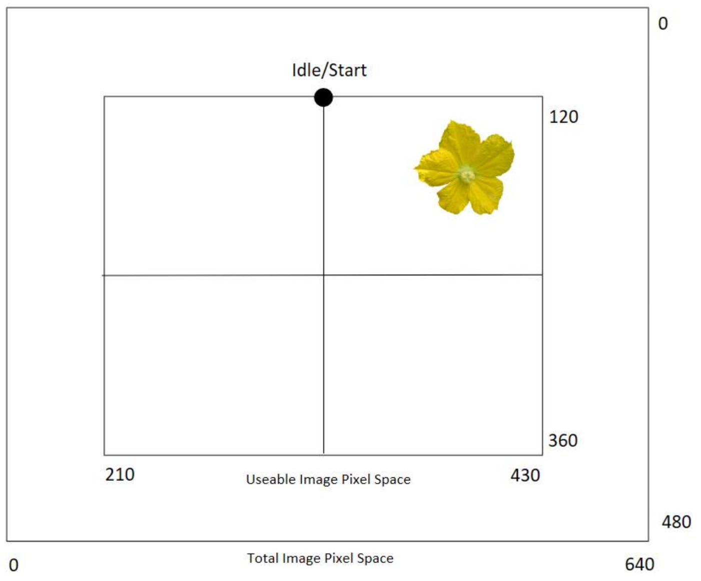


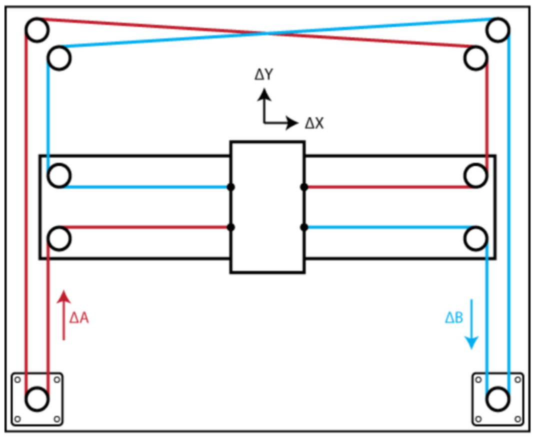
**Fig. 15.** Coordinate System Visualization.

The depicted image illustrates the functional aspect of the coordinate system. The idle/start location denotes the initial position of the pollinator relative to the entire image. Due to the camera's slight elevation above the CoreXY pollination system, there exists unusable space at the image border, inaccessible to the pollinator. The coordinate system compensates for this, preventing the motors from directing the pollinator beyond its operational range. This adjustment enhances pollinator precision by refining the accuracy of the steps per pixel ratio.

In essence, the coordinate system program serves as the primary control mechanism, orchestrating the operation of the flower detection model, computing the steps between the pollen sprayer and the flower, and executing all requisite function calls.

## F. Stepper Driver Program

The stepper driver program plays a pivotal role as a subprogram within the coordinate system. Following the completion of image processing and step calculations by the coordinate system, it invokes the stepper driver program. This program is furnished with the precise number of steps required to maneuver the pollen sprayer in both the X and Y directions. Direct interaction with the Raspberry Pi GPIO facilitates the control of the stepper drivers.



**Equations of Motion:**

$$\Delta X = {}^{1}/_{2}\,(\Delta A + \Delta B), \quad \Delta Y = {}^{1}/_{2}\,(\Delta A - \Delta B)$$

$$\Delta A = \Delta X + \Delta Y, \quad \Delta B = \Delta X - \Delta Y$$

**Fig. 16.** CoreXY Architecture and Kinematic Equations.

The functionality of the stepper driver is intricate, particularly due to the CoreXY architecture of the pollination system. Upon configuring GPIO pins for interaction with the stepper drivers' pulse and direction inputs, the function undertakes steps in X and Y directions, adjusting them based on the camera's orientation. Motor A's direction and step count are determined by summing the X and Y steps to ascertain its delta, while Motor B's direction and step count involve subtracting Y steps from X steps to find its delta. This conversion is imperative as CoreXY employs both motors collaboratively to maneuver the gantry carriage, unlike conventional X and Y motion systems utilizing individual motors for each direction. To ensure operational feasibility, negative step values are converted to their absolute values, with the direction pin set counterclockwise as necessary. Subsequently, the stepper driver program initiates a loop, pulsing each motor until it reaches its designated destination, employing CoreXY kinematics principles.

## G. CoreXY Integration

After training the flower detection model and finalizing the stepper driver and coordinating system programs, integrating with the hardware was essential for comprehensive testing and fine-tuning of the system. The wiring of stepper drivers and motors followed the specifications outlined in the DM542 datasheet.

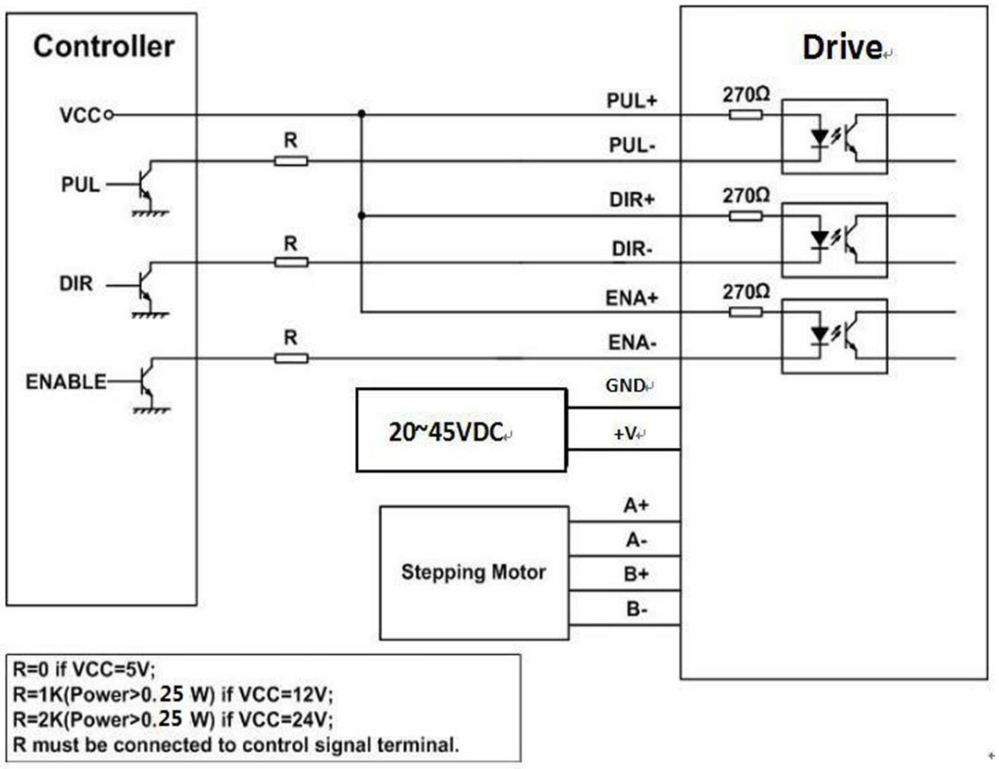


**Fig. 17.** DM542 Datasheet Connection.

The A and B coils of the stepper motors were linked to the respective stepper drivers. Additional wires were connected to the stepper drivers' +V and GND terminals to supply power. Throughout the testing phase, a DC bench power supply furnished 24V to the stepper drivers. Settings were adjusted to microstep at 400 steps per revolution, maintain a 50% current during idle, and limit each motor's current to 1 amp. Idle motor consumption was approximately 0.15A, rising to around 0.3A during operation. With motors and drivers installed on the CoreXY frame, control signals were then linked to the Raspberry Pi 5 GPIO using GPIO cables.

```
#Pin setup
PUL_A = 4  #board 7
#PUL_A- = board 9
PUL_B = 11  #board 23
#PUL_B- = board 25
DIR_A = 18  #board 12
#PUL_A- = board 14
DIR_B = 24  #board 18
#PUL_B- = board 20
```

**Fig. 18.** Stepper Driver Pins Code Snippet.

The connections for the stepper driver pulse and direction were configured as indicated in the provided code snippet. Each negative pulse and direction connection was linked to an individual ground terminal on the Raspberry Pi GPIO. While the specific pin values were somewhat arbitrarily chosen, the primary aim was to have the positive pulse and direction connections adjacent to available ground pins.

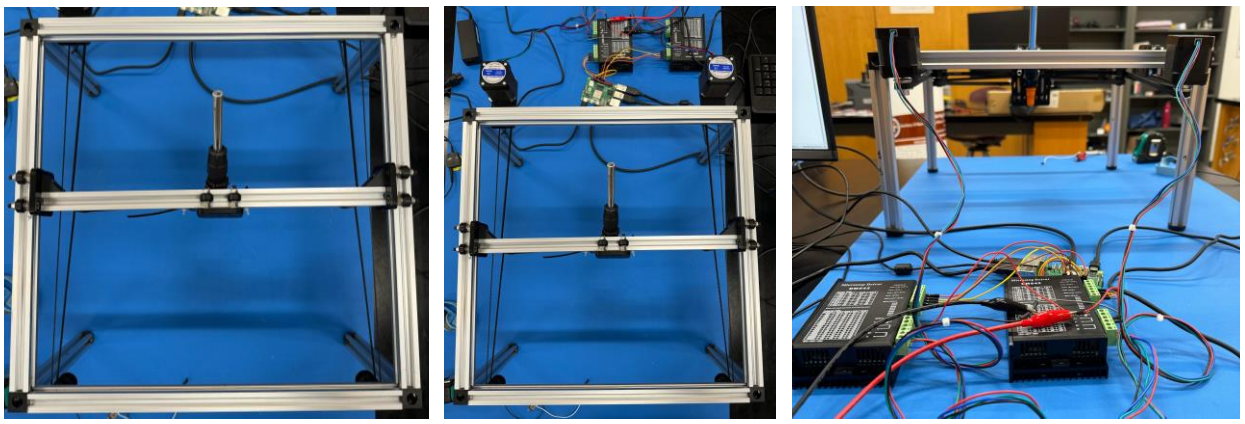


**Fig. 19.** Perspectives of Integrated Detection and CoreXY.

With hardware and software now integrated, the testing and tuning phase of the project began. A variant of the CoreXY driver program, known as CoreXYTest, was developed for manual testing of the CoreXY pollination system, allowing users to provide X and Y steps manually. During this phase, everything performed better than expected. Users could move the carriage with high precision using only the software, requiring minimal tuning for optimal hardware and software performance. The stepper motor pulse width was adjusted from 1ms down to 0.1ms in increments of 0.1ms, with an experimentally determined optimal pulse width of 0.7ms providing a balance of speed and system stability. Additionally, the test program helped determine the maximum number of steps the pollen sprayer could travel in the Y and X directions—around 1800 steps and 1200 steps respectively—before reaching the edge of the plane. This information was crucial for accurate tuning of the CoreXY driver program before integration into the SARDOG robot.

## IV. SARDOG Integration and Testing

Integrating the CoreXY flower pollination system into the SARDOG robot involved a step-by-step tuning process. Initially, the robot was widened to accommodate the CoreXY frame, which was then secured in place using zip ties around each corner. Since the robot undergoes frequent remodeling and relocation, a permanent mounting method wasn't suitable as it would hinder movement through narrow spaces. The zip ties provided flexibility, allowing the pollination system frame to slide in and out of the robot when needed. Next, the stepper drivers and Raspberry Pi 5 were secured beside the frame on a small wooden board mounted in one corner of the robot. The stepper drivers were powered by the robot's 24V output, while the camera was fixed to the front end, providing an elevated view of the underside. The electric pollinator gun, depicted in Figure 20, is initially filled with artificial pollen specific to the target flower. Once the system detects the flower stigma, the Raspberry Pi 5 sends a signal to the trigger to activate the pollination process. With all hardware integrated, the final phase involved software testing and calibration.

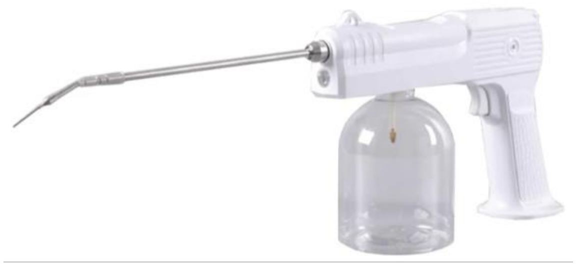


**Fig. 20.** Electric Pollinator Gun.

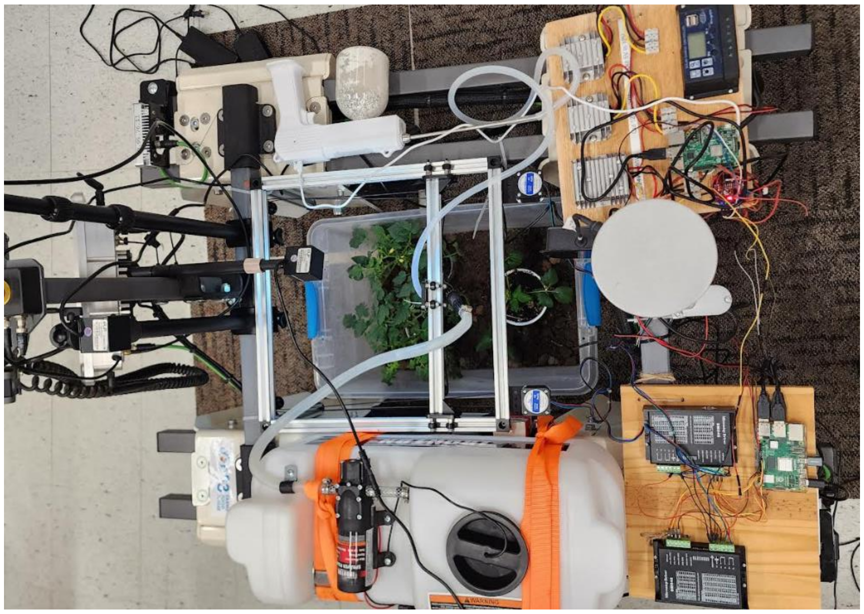


**Fig. 21.** Entire System Integrated on SARDOG.

In Figure 21, the CoreXY pollination frame is centrally mounted on the robot. At the bottom right, the two stepper drivers and the Raspberry Pi 5 computer are securely fastened to the wooden board. The camera is positioned to look straight down beneath the robot, providing a clear view of the flowers below. Additionally, a box containing soil and flowers is placed directly under the pollination system to test its performance on real flowers.

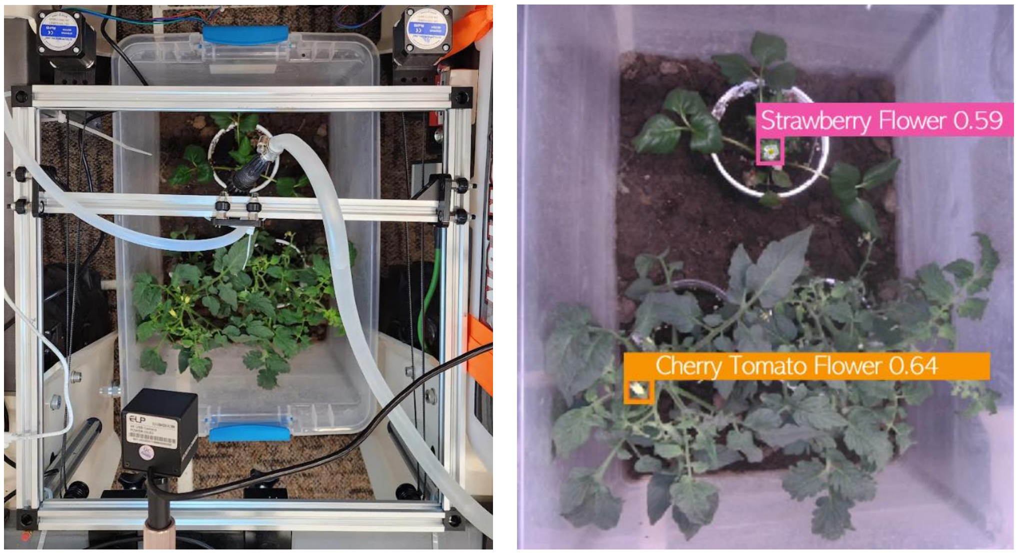
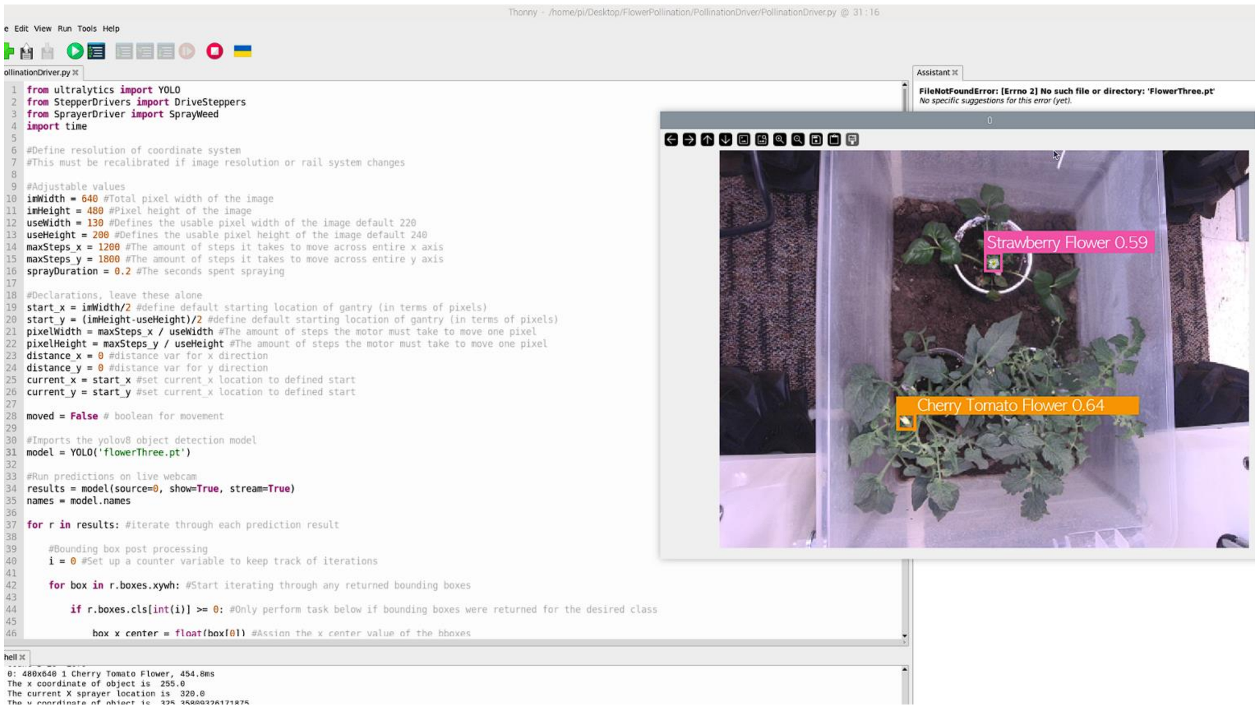


**Fig. 22.** Working System Close Up.


Thonny - /home/pi/Desktop/FlowerPollination/PollinationDriver/PollinationDriver.py @ 31 : 16
e Edit View Run Tools Help
ollinationDriver.py
Assistant
Strawberry Flower 0.59
Cherry Tomato Flower 0.64
hell

```
from ultralytics import YOLO
from StepperDrivers import DriveSteppers
from SprayerDriver import SprayWeed
import time

#Define resolution of coordinate system
#This must be recalibrated if image resolution or rail system changes

#Adjustable values
imWidth = 640 #Total pixel width of the image
imHeight = 480 #Pixel height of the image
useWidth = 130 #Defines the usable pixel width of the image default 220
useHeight = 200 #Defines the usable pixel height of the image default 240
maxSteps_x = 1200 #The amount of steps it takes to move across entire x axis
maxSteps_y = 1800 #The amount of steps it takes to move across entire y axis
sprayDuration = 0.2 #The seconds spent spraying

#Declarations, leave these alone
start_x = imWidth/2 #define default starting location of gantry (in terms of pixels)
start_y = (imHeight-useHeight)/2 #define default starting location of gantry (in terms of pixels)
pixelWidth = maxSteps_x / useWidth #The amount of steps the motor must take to move one pixel
pixelHeight = maxSteps_y / useHeight #The amount of steps the motor must take to move one pixel
distance_x = 0 #distance var for x direction
distance_y = 0 #distance var for y direction
current_x = start_x #set current_x location to defined start
current_y = start_y #set current_x location to defined start

moved = False # boolean for movement

#Imports the yolov8 object detection model
model = YOLO('flowerThree.pt')

#Run predictions on live webcam
results = model(source=0, show=True, stream=True)
names = model.names

for r in results: #iterate through each prediction result

    #Bounding box post processing
    i = 0 #Set up a counter variable to keep track of iterations

    for box in r.boxes.xywh: #Start iterating through any returned bounding boxes

        if r.boxes.cls[int(i)] >= 0: #Only perform task below if bounding boxes were returned for the desired class

            box_x_center = float(box[0]) #Assign the x center value of the bboxes
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'FlowerThree.pt'
No specific suggestions for this error (yet).
```

```
0: 480x640 1 Cherry Tomato Flower, 454.8ms
The x coordinate of object is  255.0
The current X sprayer location is  320.0
The y coordinate of object is  325.35809326171875
```

**Fig. 23.** CoreXY Driver Program Performing Predictions.

In Figure 22, the camera offers an optimal view of the ground beneath the robot. While the CoreXY driver program is operational, the flower detection model conducts inferences on images every few hundred milliseconds. Upon detecting flowers, as shown in the image on the right, the CoreXY driver program retrieves the bounding box coordinates, positions the pollen sprayer nozzle over each flower, activates the electric pollinator sprayer, and then returns it to the starting location. Calibrating the camera to ensure alignment with the flower required some adjustments initially, but after fine-tuning, the system operated smoothly. This marks the successful development and implementation of a fully functional flower detection and pollination system.

## Conclusion

We've created an autonomous flower detection and pollination system for SARDOG. Utilizing a dual belt gantry system in a CoreXY configuration, we've enhanced precision and responsiveness while reducing maintenance needs in field settings. The system employs a trained YOLOv8 deep learning model for flower identification and classification, coupled with a complex coordinate system program to trigger the pollination process. Extensive research and design efforts were invested in perfecting the detection programs and deep learning model. The resulting system addresses the growing concerns related to declining pollinator populations while also enhancing crop yield and quality through consistent and timely pollination. SARDOG's compact size enables it to perform various functions efficiently, making it a valuable asset for flower detection and pollination.

### Acknowledgments

We want to give a special thanks to DPS Telecom for generously providing funding for the project as well as to the Economic Development Administration (EDA) Project Grant 077907908, Fresno-Merced Future of Food Innovation (F3) Coalition.

## References

[1] H. Kulhandjian, B. Irineo, J. Sales, M. Kulhandjian, “Low-Cost Tree Health Categorization and Localization Using Drones and Machine Learning,” in Proc. of IEEE International Conference on Computing, Networking and Communications (ICNC), Big Island, Hawaii, Feb. 2024.

[2] H. Kulhandjian, D. Rocha, B. Bennett, N. Amely, M. Kulhandjian, “AI-based Precision Weed Detection and Elimination,” in Proc. of 16th International Conference on Precision Agriculture, Manhattan, Kansas, July. 2024.

[3] H. Kulhandjian, N. Amely, M. Kulhandjian, “AI-based Pollinator using XY-Core Robot,” in Proc. of 16th International Conference on Precision Agriculture, Manhattan, Kansas, July. 2024.

[4] H. Kulhandjian, Y. Yang and N. Amely, "Design and Implementation of a Smart Agricultural Robot bullDOG (SARDOG)," in Proc. of IEEE International Conference on Computing, Networking and Communications (ICNC), Big Island, Hawaii, Feb. 2024.

[5] T. Hiraguri et al., "Autonomous Drone-Based Pollination System Using AI Classifier to Replace Bees for Greenhouse Tomato Cultivation," in IEEE Access, vol. 11, pp. 99352-99364, 2023, doi: 10.1109/ACCESS.2023.3312151.

[6] “Cucumber Object Detection Dataset and Pre-Trained Model by Creeds Workspace,” Roboflow. https://universe.roboflow.com/creeds-workspace/cucumber-6wrv8.

[7] “Strawberry Flowers Detection Object Detection Dataset by tru projects,” Roboflow. https://universe.roboflow.com/tru-projects-7dbwp/strawberry-flowers-detection.

[8] “Sugar-Cherry Object Detection Dataset (v1, 2023-08-25 3:38pm) by Phng m,” Roboflow. https://universe.roboflow.com/phng-m/sugar-cherry-8rrnf/dataset/1.

[9] D. Steininger, A. Troll, G. Croonen, J. Simon, and V. Widhalm, “The crop and weed dataset: A multi-modal learning approach for efficient crop and weed manipulation,” 2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), Jan. 2022. doi:10.1109/wacv56688.2023.00372