Toy Model of an Autonomous Crop Weeder: Project Report

Matthew Watson
School of Computing and AI
Arizona State University
Tempe, United States
mjwats10@asu.edu

Chia-Cheng Kuo
School of Computing and AI
Arizona State University
Tempe, United States
ckuo37@asu.edu

Affan Bin Usman
School of Computing and AI
Arizona State University
Tempe, United States
ausman4@asu.edu

I. Background

In today's world, crop growers in developed companies must meet increasingly stringent consumer demands for high-quality produce at an affordable price, all while minimizing the use of synthetic pesticides and fertilizers. As a result, modern agricultural practices around integrated pest management (IPM) are quickly adapting to the market's demands to maintain the profitability of growing crops despite rising input costs.

Of particular importance to growers of organic vegetable row crops, such as spinach, arugula, carrots, and baby leaf lettuce, is meeting the challenge posed by unwanted plant species (weeds) growing in and among the valuable crop. As growers of organic produce are not permitted to use many of the more effective weed control chemistries available for conventional produce, it is often not possible to achieve weed control through chemical means. Therefore, organic produce growers must rely on manual labor to remove the weeds at regular intervals during the growing cycle. This process is expensive, and due to worsening labor shortages, is often unavailable entirely.

At the same time, recent innovations in computer vision, particularly with respect to the ability to perform real-time object detection and instance segmentation with relatively inexpensive consumer-grade hardware, have allowed the proliferation of automated mechanical solutions to this problem.

For our project, we use a LEGO Mindstorms EV3 Core Kit [1] to build a toy model of such an autonomous system (Fig. 1), replacing real plants in an outdoor field setting with various colors of LEGO blocks, with green blocks representing the valuable crop and other colors representing various species of weed.

II. System Overview

Fig. 2 shows an overview of how we implemented our system. In this project, we utilize YOLOv5 [3], an open-source image recognition model, to let us recognize the different colors of LEGO blocks. The primary detection and control program runs on a MacBook Air. It first takes the real-time video captured from an iPhone linked to the MacBook as a webcam using the Continuity Camera feature and then loads the video into our custom-trained YOLOv5 model to recognize

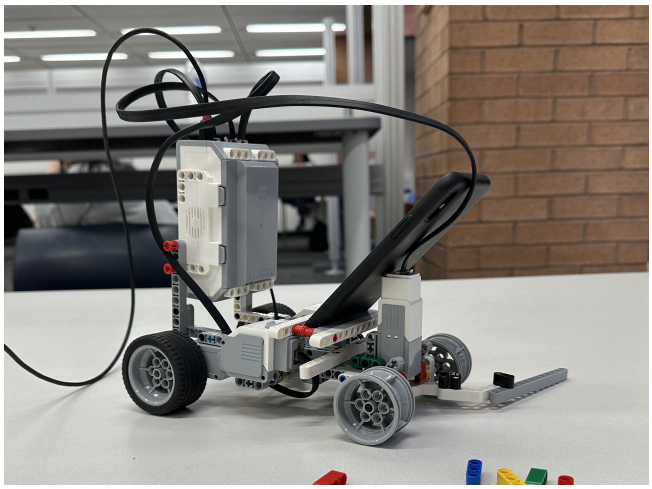

Fig. 1. The finished setup of our toy autonomous weeder

LEGO blocks. Our control logic proceeds by sending the controlling commands to the LEGO car based on the output of the YOLOv5 model, using the python ev3-dc library (which sends direct commands to the LEGO car without requiring to upload or install any program on the LEGO car) via USB cable.

III. Hardware

- LEGO EV3 Education Kit
  - ARM9 Controller
  - 300 MHz
  - RAM 64 MB
- Camera / Phone

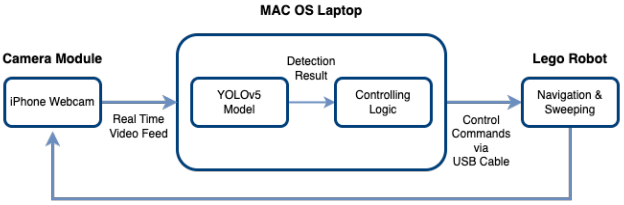

Fig. 2. Control Logic

  - iPhone Xs*
- Processing Hub
  - Laptop (MacOS)*

*Note: *In order to utilize Continuity Camera, it is essential to have a MacOS-based laptop and an iOS-based phone.*

IV. Software

A. Training Yolov5s model

To train a machine learning model to recognize different colors of LEGO blocks, we started by collecting an image dataset. We manually took *550* pictures of LEGO blocks in various environments including placing the LEGO blocks on surfaces such as carpets, tiles, and tables. We then use Label Studio [4], an open-source data labeling tool, to manually label the LEGO blocks in the images with bounding boxes and class labels, specifying the location of the LEGO block in the image, as well as the color of the block. At this point, we obtained the complete image dataset, ready to be used for training the YOLOv5s model.

YOLOv5 stands for "You Only Look Once" version 5. It is a deep learning model used for image recognition developed by Ultralytics, to identify objects and the location of objects in images. We implemented the training of the YOLOv5s model on Google Colab utilizing Python. To train the YOLOv5s model, we simply install the YOLOv5 library from the official GitHub repository, specify the dataset path, and call the 'train.py' program from the library. We divided the dataset into training and validation subsets using an 80/20 train/validation split and trained YOLOv5s from scratch for 300 epochs with a batch size of 16 and an input image size of *416x416*. After the training, we obtained a customized YOLOv5 model which was able to detect the LEGO blocks, their color, and their location in images. We then created a Python program to load the customized YOLOv5 model using PyTorch and used OpenCV to feed the real-time video captured by the iPhone into the YOLOv5 model. The model would then output what objects (in this case, the different colors of LEGO blocks) are detected, along with the location of the object in the image (*xmax, xmin, ymax, ymin*), and the confidence level of its prediction.

B. Control Policy

We used the ev3-dc Python library [2] and a hand-crafted control policy to send movement and sweeper-actuation commands from the laptop to the EV3 controller through a USB interface. The control policy can be broken down into three components: 1) an exploration mode that is active as long as no LEGO bricks are detected, 2) a navigation mode that guides the vehicle to a brick once it is detected, and 3) a function to actuate the sweeper arm to remove bricks representing weeds.

Upon entering exploration mode (Fig. 3), the vehicle proceeds to execute a random walk. All the while, it is analyzing the video stream from the phone looking for a LEGO brick. We implemented the exploration strategy by allowing the model to pick a *MoveForward* action with a 70% probability and *RandomTurnDirection* action with a 30% probability. The

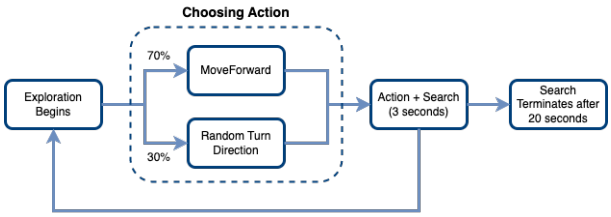

Fig. 3. Exploration strategy

values of these probabilities were found by repeated trials. In case the robot is unable to locate any weed block for 20 seconds, the robot stops.

Once the object detector returns a detection with confidence greater than the minimum threshold, the vehicle switches to navigation mode. In navigation mode, the policy finds the brick closest to the vehicle, determines the horizontal offset between the center of the brick and the center of the camera image, and calculates steering and longitudinal commands which are sent to the EV3 controller (Fig. 4). The controller, in turn, sends commands to the two motors which drive the rear wheels. Finally, when the brick, the vehicle is navigating towards, comes within range of the sweeper, the policy stops the vehicle and then sends the EV3 controller a command to actuate the sweeper arm if the brick represents a weed (any non-green block) (Fig. 5). The arm then sweeps from left to right across the front of the vehicle and back to its original position, sweeping the brick out of the way.

V. Obstacles

When deploying the trained YOLOv5 model on the LEGO car, we encountered a problem in which the model constantly detect the sweeper (or broom) which is a longer gray LEGO block we used to swipe out the LEGO blocks, as a blue LEGO block. The issue would make our LEGO car keep swiping its broom even if there were no actual LEGO blocks. To solve the issue, we simply collected more images of the gray LEGO block without labeling it. After updating the dataset with the long gray LEGO block images added, the issue was solved

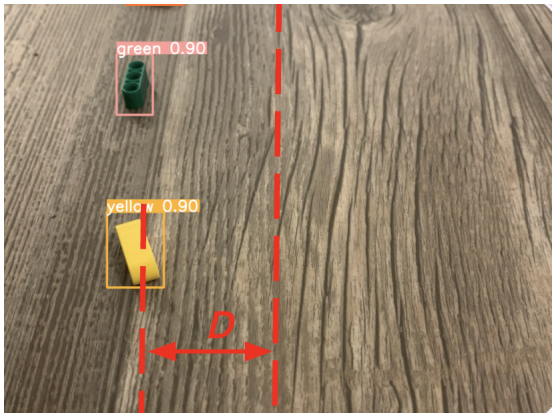

Fig. 4. Calculating brick offset D for steering.

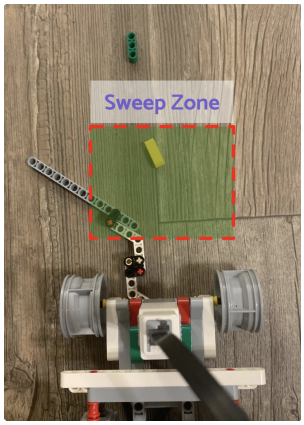

Fig. 5. Sweeper is actuated when a ”weed” is within range of the arm.

after training the YOLOv5 model again. To summarize, further data collection and retraining solved the problem.

VI. Results

We tested our robot by creating lines of LEGO blocks of various colors, mimicking a crop row in a field environment, and then started our weeder robot at one end of the row and let it navigate down the row, swatting weeds out of the line as it went. While we did not do any quantitative assessment of the robot's performance, in this setting it performed well, with rare mistakes caused by false positive or false negative detections. To ”stress test” the steering policy, we modified the previous setup by re-arranging the line of blocks into a curved ”snaking” pattern. In this condition, performance was still good but the vehicle would occasionally run over a block with one of the front tires, representing possible damage to the valuable crop. Finally, we evaluated the performance of the exploration policy by adding large gaps between bricks in the linear row arrangement. This meant that after passing or sweeping the last brick before a gap, the weeder would no longer have any bricks in view, and thus became reliant on the exploration policy to find the next brick in the line. We quickly realized that the best results were achieved using an exploration policy that assigned a much greater probability mass to exploring forward rather than exploring laterally, which reflects the fact that the bricks, like crops in the field, are arranged in rows. However, if the gaps between bricks were made large enough, the machine would reliably fail to find the next brick in the row and end up on a trajectory that resulted in exploring an empty portion of the ”field” area.

VII. Conclusion

Our toy model of an autonomous row crop weeder successfully mimics the basic functionality of the real thing, albeit in a more controlled environment that does not contain obstacles such as people, irrigation infrastructure, or other vehicles. Future work could extend its functionality by implementing a policy to weed multiple parallel crop lines, a better representation of the layout of an actual field. Additionally, an obstacle avoidance module could be added to allow it to safely work in the proximity of stationary hazards. To cater to the factor of scalability, a ”cooperative” module combining localization and mapping could be added to allow two identical weeder robots to potentially divide up the ”field” consisting of multiple crop lines without either machine repeating areas already weeded by the other.

References

[1] LEGO mindstorm ev3 https://education.lego.com/en-us/start/mindstorms-ev3#Setting-Up
[2] ev3-dc Python library https://pypi.org/project/ev3-dc/
[3] YOLOv5 https://github.com/ultralytics/yolov5
[4] Label Studio https://labelstud.io/