Arab Academy for Science, Technology and Maritime Transport

College of Computing and Information Technology

Computer Science Department

B. Sc. Final Year Project

Autonomous Fruit Harvesting Robot Using Deep Learning

Presented By:

Ahmed Yasser Saad *Marwan Mohammed Esam El-Din*

Omar Alaa El-Din Suleiman *Yousef Adel Sabra*

Youssef Mamdouh Makram

Supervised By:

Prof. Dr. Nashwa Elbendary

JULY – 2023

DECLARATION

I hereby certify that this report, which I now submit for assessment on the programme of study leading to the award of Bachelor of Science in Computer Science, is all my own work and contains no Plagiarism. By submitting this report, I agree to the following terms:

> *Any text, diagrams or other material copied from other sources (including, but not limited to, books, journals, and the internet) have been clearly acknowledged and cited followed by the reference number used; either in the text or in a footnote/endnote. The details of the used references that are listed at the end of the report are confirming to the referencing style dictated by the final year project template and are, to my knowledge, accurate and complete.*

I have read the sections on referencing and plagiarism in the final year project template. I understand that plagiarism can lead to a reduced or fail grade, in serious cases, for the Graduation Project course.

Student Name: Ahmed Yasser
Registration Number: 19108846

Signed: _____

Date: – –

Student Name: Omar Alaa El-Din
Registration Number: 19108738

Signed: _____

Date: – –

Student Name: Marwan Mohammed Essam
Registration Number: 19108916

Signed: _____

Date: – –

Student Name: Yousef Adel
Registration Number: 19108685

Signed: _____

Date: – –

Student Name: Youssef Mamdouh
Registration Number: 19108541

Signed: _____

Date: – –

STUDENTS CONTRIBUTION

Autonomous Fruit Harvesting Robot Using Deep Learning

By:

Ahmed Yasser Saad
Marwan Mohammed Essam El-Din
Omar Alaa El-Din Suleiman
Yousef Adel Sabra
Youssef Mamdouh Makram

Chapter	Title	Contributors
1	Introduction	Omar Alaa El-Din Suleiman Yousef Adel Sabra
2	Literature Review	Ahmed Yasser Saad Marwan Mohammed Essam El-Din Omar Alaa El-Din Suleiman Yousef Adel Sabra Youssef Mamdouh Makram
3	Modeling	Ahmed Yasser Saad Youssef Mamdouh Makram
4	Results and Discussion	Ahmed Yasser Saad Marwan Mohammed Essam El-Din Omar Alaa El-Din Suleiman Yousef Adel Sabra Youssef Mamdouh Makram
5	Conclusions	Marwan Mohammed Essam El-Din Yousef Adel Sabra

ACKNOWLEDGMENT

We are grateful and appreciate everyone who has contributed to the development of the autonomous fruit harvesting robot.

We express our utmost gratitude towards our supervisor who has provided us with invaluable guidance and great care for the progression of the project throughout the entire duration of its development.

Additionally, we thank the community of researchers and engineers in the fields of robotics and A.I. as their work has been inspiring and aiding us in making the project possible.

We also thank the Information Technology Industry Development Agency (ITIDA) as this project was funded by a grant in the Information Technology Academia Collaboration (ITAC) program with the acceptance ID GP2023.R18.127.

Thank you all for your contribution and your support in this project.

ABSTRACT

Agricultural automation has been an interest for many companies due to the high benefits it provides. Our project aims to develop a robotic arm combined with a deep learning model that can automatically harvest fruits in order to address the challenges of manual harvesting, which is a tedious and laborious process that heavily relies on experienced workers. The loss of crops during harvest is a major issue in Egypt, particularly for tomatoes, which represent the highest rates of loss compared to other tomato-producing countries. Results acquired show a high average precision of 82% for the You Only Look Once version five model, an 80% for the You Only Look Once version seven model, and finally the Faster Region-based Convolutional Neural Network acquired an average precision of 70%. This technology could have a significant impact on food security and development in Egypt, as well as improving the quality and safety of agricultural production and exporting activities.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS/ABBREVIATIONS

AI	Artificial Intelligence
CNN	Convolutional Neural Network
CSPDarknet	Cross Stage Partial Dark Network
DOF	Degrees of Freedom
IoU	Intersection over Union
mAP	Mean Average Precision
PANet	Path Aggregation Network
RCNN	Region-based Convolutional Neural Network
ResNet-50	Residual Neural Network 50
RPN	Region Proposal Network
SSD	Single Shot Detector
SVM	Support Vector Machine
UR	Universal Robots
VGG-16	Visual Geometry Group 16
YOLO	You Only Look Once

Chapter One

INTRODUCTION

Traditionally, harvesting fruits has been a labour-intensive and time-consuming process across the globe. It heavily relies on a workforce that is physically fit, demanding significant time and financial resources to accomplish a successful harvest. Despite advancements in agricultural practices, many regions still face challenges in ensuring food security, preventing crop loss and combating hunger. Egypt, in particular, suffers with a moderate level of hunger, as highlighted by the 2019 Global Hunger Index, where it is ranked 61 out of 117 countries [1]. In addition to food security concerns, Egypt faces problems related to the affordability, quality, and safety of its food supply. The country heavily relies on global markets for more than half of its staple food items, exacerbating the challenges it faces in meeting the nutritional needs of its population.

One contributing factor to the growing problem of hunger in Egypt is the significant loss of crops during harvest seasons. Among the crops affected, tomatoes suffer the highest rates of loss compared to other tomato-producing countries, despite Egypt's notable position as the world's fifth-largest tomato producing country [2]. This paradox raises the need for innovative solutions to minimize losses and ensure a more sustainable and secure food supply.

Addressing the challenges associated with fruit harvesting is not just a matter of improving productivity; it is crucial for overcoming the food security and hunger-related issues Egypt faces. By reducing crop losses, farmers can increase their yields and contribute to the availability of nutritious food options for the Egyptian population. Moreover, enhancing the efficiency of the harvest process can lead to cost savings, making food more affordable and accessible for the average consumer.

The following section, being the background and literature review, will include a review of research conducted concerning the topic of the project or is heavily related to the idea behind this project. The third section regarding the mathematical model illustrates the implementation approach, including the dataset preparation, detection and classification, and robot arm control. The results section presents the results and performance metrics and outcomes of the system, such

as the accuracy of the detection algorithm. The final section, the conclusion, will provide a summary of the findings, highlighting the system's effectiveness and benefits, as well as discussing the limitations and future research directions.

Chapter Two

BACKGROUND AND LITERATURE REVIEW

There has been a growing interest in the use of robotics and AI in agriculture to address these challenges. The use of robotic systems for harvesting fruits and vegetables has been widely studied and several solutions have been proposed. These include robotic arms and grippers that can be used to pluck fruits and vegetables, as well as cameras and sensors that can be used to detect and classify the fruits.

A project in Spain regarding the field was worked on by Delia et al, [3]. The objective of this project was to develop a more robust system for harvesting aubergines using a dual arm robot. The complexity in the environment they were working in required a system that could adapt and coordinate efficiently, yet the success rate reached as high as 91.67%. The methodology employed the use of two cameras for higher depth accuracy, the usage of the SVM Cubic algorithm, a dual arm robot and over 1700 images of aubergines for the dataset, all assumed self-captured.

Yuki et al, [4] is a group with a similar interest that developed a project for harvesting Fuji apples, a popular type of apple in Japan. Unlike the previously stated project, the group used only one stereo camera for their robot. The Single-Shot Detector (SSD) algorithm was the used object detection model with an accuracy of over 90% along with a Universal Robots UR3 robotic arm to grasp the fruits. The engineers stated that their project would also work on pears, since the fruit has a similar shape to the Fuji apples.

Another group with Xiong et al, [5] set their goals to build a robot that can gently pluck strawberries with minimal damage to the berry. Strawberries are fragile and can easily be damaged, making them less likely to be accepted in the local markets due to the standards. They proposed a system using Faster Region-based Convolutional Neural Networks (RCNN), and a uniquely designed robot to pluck strawberries. The robot has a cone-shaped end-effector to move between the berries without damaging them, the cone then opens to grab a berry or several berries and cuts the stem with built-in scissors inside the cone. The entire system was able to reach success rates

between 50% to 97.1% on the first attempt of plucking and 75% to 100% success rate on the second attempt.

Olarewaju, an engineer in Shanxi Agricultural University decided to look into improving the You Only Look Once (YOLO) algorithm's accuracy on his own [6]. He modified the original YOLOv4 architecture to suit his own environmental conditions on the Muskmelon fruit, thus naming the model YOLOMuskmelon. YOLOMuskmelon is developed to detect muskmelons offering faster detection than other YOLO models. The model is 56.1% faster than the YOLOv4 model. The used dataset was self-captured in a greenhouse in China which contained 410 images under natural daylight.

More people were interested in improving the detection technologies and automating harvesting processes. Toon et al, focus on tomatoes in the research, [7] the algorithm was based on Convolutional Neural Networks (CNN) for classification with custom layers. Using a dataset of 800 images, the group was able to get an average of 76% accuracy in detecting whether the object is a tomato or not, and an accuracy of 98% in classifying whether a tomato is ripe or unripe.

Chapter Three

MATHEMATICAL MODEL

3.1 PROJECT MODULES

The study aimed to develop an automated system for tomato harvesting by combining three modules: Fruit Image Capturing module, Deep Learning Module and Robot Movement module. The project methodology focuses on utilizing advanced technologies such as deep learning and robotics to streamline the tomato harvesting process, ultimately improving productivity and reducing the labor-intensive harvesting tasks.

The following figure represents the block diagram for the entire project, starting from the first module of capturing the fruit images till the final module of grasping the fruit.

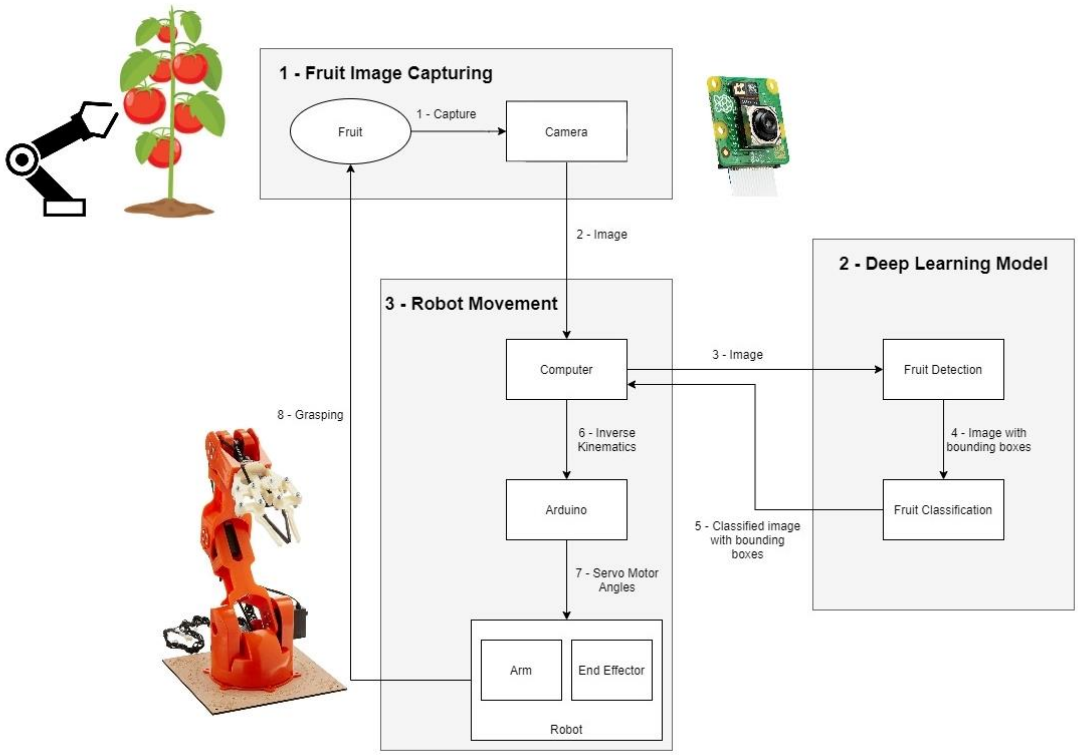

Figure 3-1 Block diagram for the full project

3.1.1 Fruit image capturing module

The Fruit image capturing module is a vital component of the system. This module involves a camera connected to a Raspberry Pi, which serves as the image capture device. The camera captures high-resolution images of tomato clusters on bushes with varying lighting conditions, leaf occlusions, and different ripeness stages. The Raspberry Pi, acting as an intermediary, transfers the captured images to a connected PC for further processing and analysis.

3.1.2 Deep learning model module

In this module, 3 deep learning models were chosen as test cases for the system. The models are stored in the computer and start the processing phase once the camera takes a picture. The model creates bounding boxes around all tomatoes shown in the image and starts classifying the tomatoes as ripened, half ripened and unripened. Once the classification phase is over, the image, which now has bounding boxes around each tomato as well as a classification tag for each, is sent to the main computer to be used in the Robot Movement module, explained in the next sub-sub section.

The deep learning models tested for the proposed system are as follows:

a) YOLOv5 model [8]

b) YOLOv7 model [9]

c) Faster RCNN model [10]

a) The YOLOv5 model is a successor of the previous YOLOv4 model, it uses a different architecture called EfficientDet, which is based on the EfficientNet architecture. Due to the complexity of its architecture, it ensures higher accuracies and better generalization to more object categories compared to its predecessors. Selected for its accuracy and efficiency, it held a significant position in detecting and classifying objects in real-time.

The following figure represents the YOLOv5 architecture, consisting of three main parts:

- Cross Stage Partial Dark Network (CSPDarknet) as the backbone

- Path Aggregation Network (PANet) as the neck
- The YOLO layer as the head

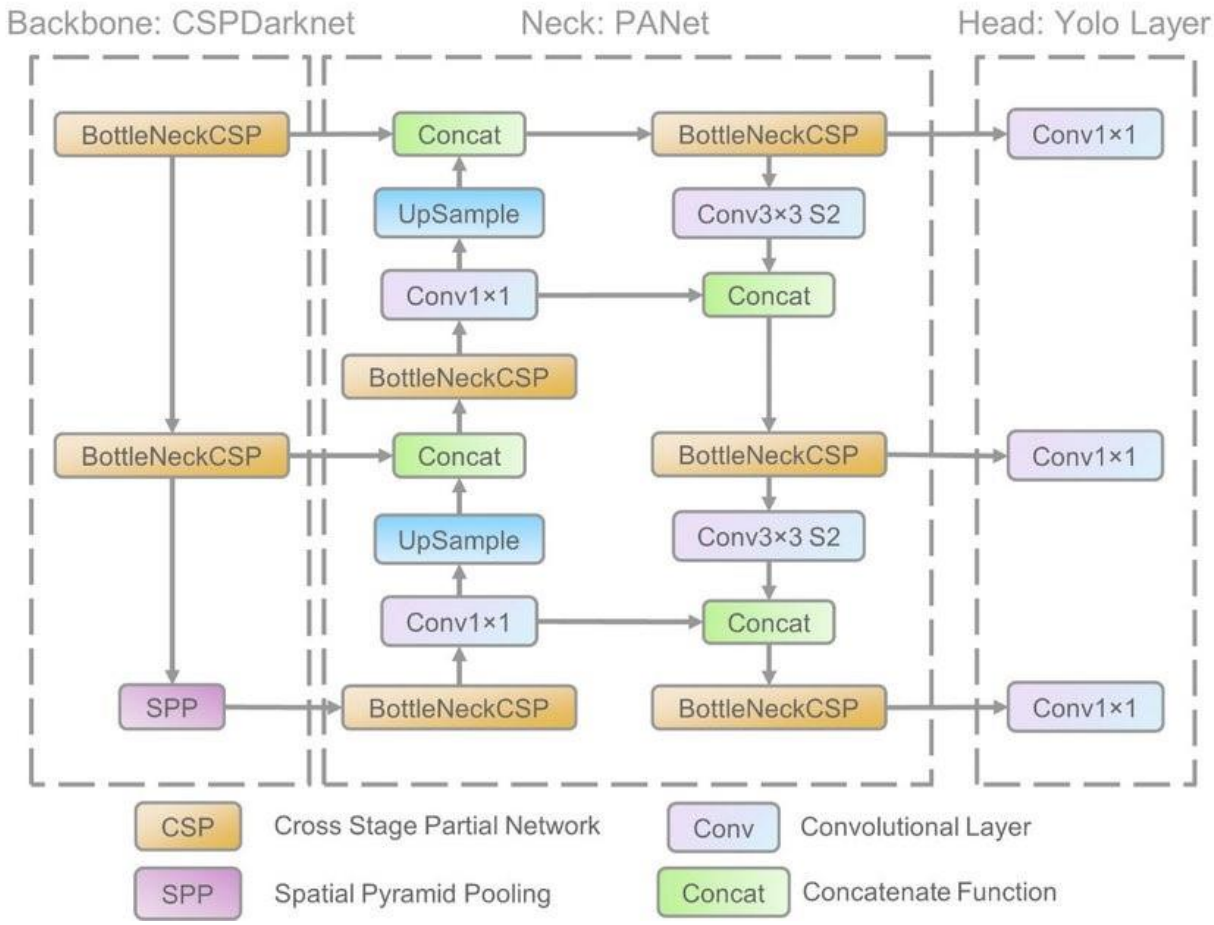

Figure 3-2 YOLOv5 architecture diagram [11]

b) The YOLOv7 model is two generations newer than YOLOv5 and it introduces several new features and improvements, one of which is using anchor boxes for better detection of objects of varying shapes, this will greatly reduce the number of false positives acquired from the testing.

The architecture of the model has similarities to YOLOv5 as it is a straight upgrade. Below is a figure representing the architecture of the model.

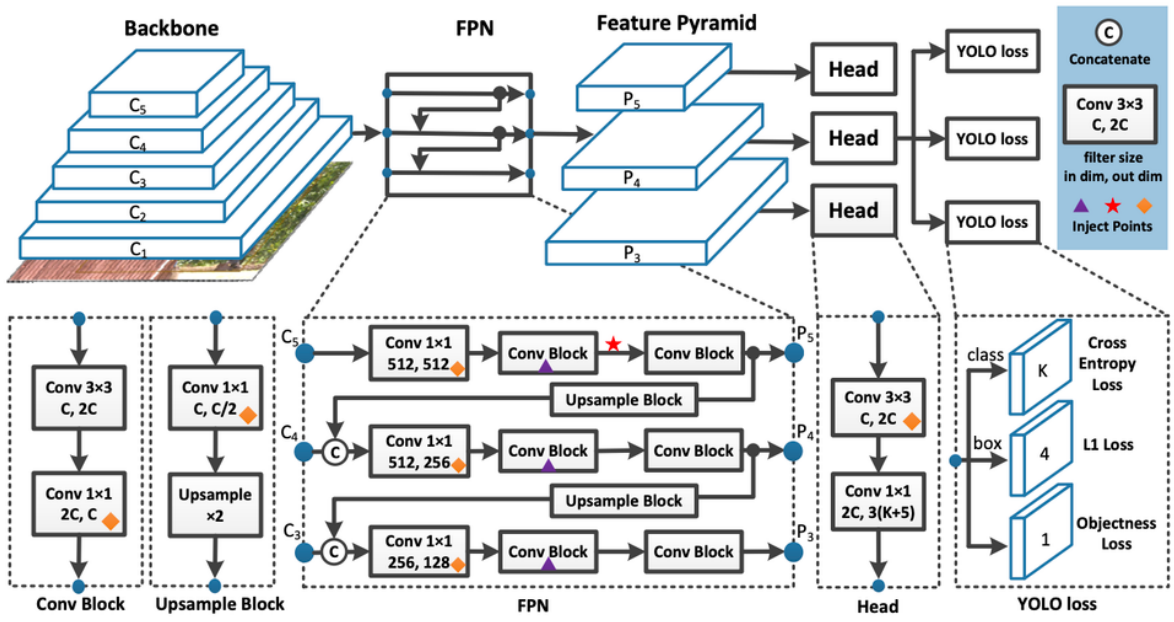

Figure 3-3 YOLOv7 architecture diagram [12]

c) The Faster RCNN model is also widely used in computer vision, it is composed of two main modules, a Region Proposal Network (RPN) which is a deep fully connected network, and a CNN, in this case, the Fast RCNN. Below is an image representing the Faster RCNN model.

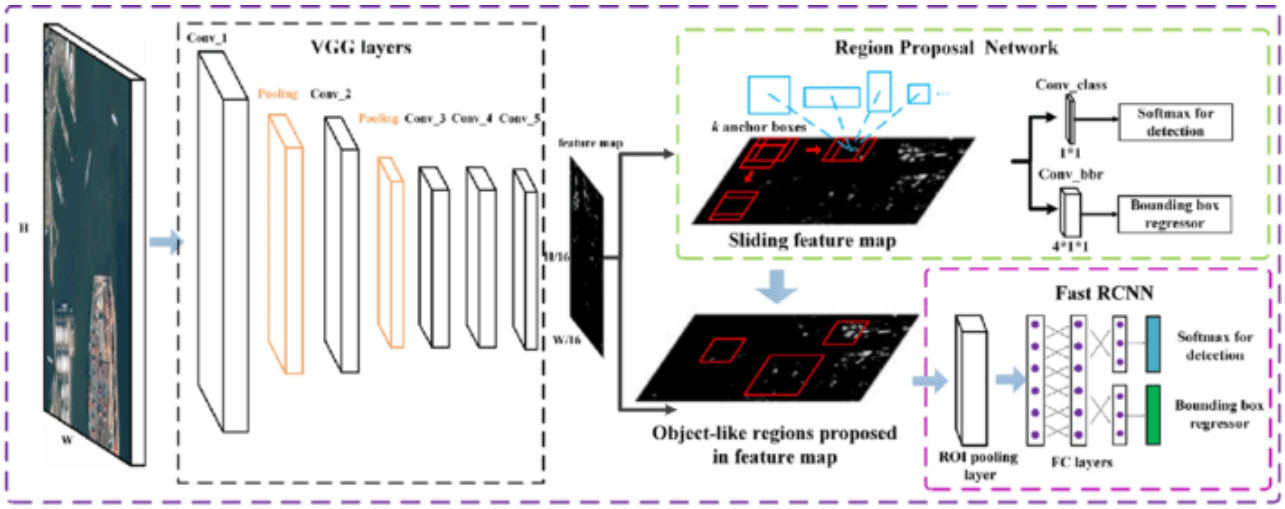

Figure 3-4 Faster RCNN architecture diagram [13]

3.1.3 Robot movement module

The robotic arm module is a crucial component as it is responsible for physically handling the identified tomatoes. The Tinkerkit Braccio, from Arduino, has a simple design which possesses six degrees of freedom (DOF). The input image from the deep learning model is used to extract the exact coordinates of each tomato with respect to the camera. The coordinates are used to calculate the inverse kinematics utilizing a library created by cgxeiji [14] to get the angles each joint in the robot needs to arrive at the specified coordinates. The arm will then grasp the tomato and deposit it into a designated container. Afterwards, the arm will return to a position viewing its harvesting area to gather any additional tomatoes that need harvesting. This process can be repeated seamlessly and efficiently.

3.1.4 Full system setup

In the following figure, the image represents the full system after setup and in working condition.

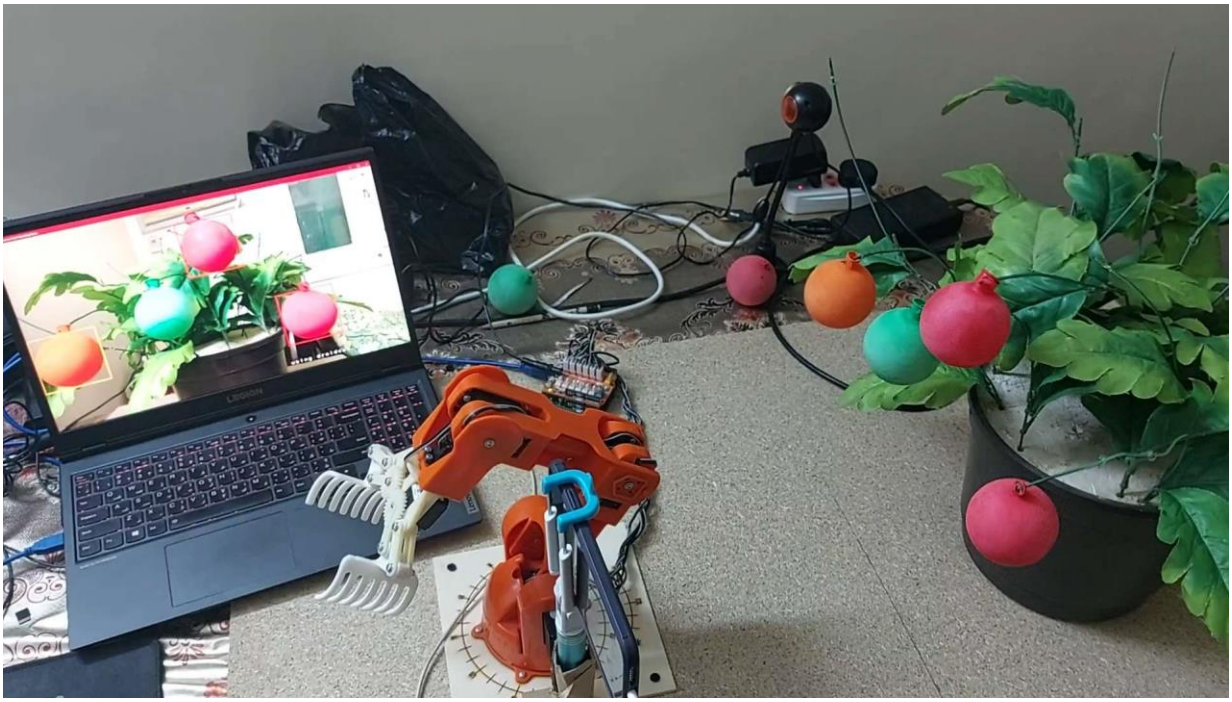

Figure 3-5 Full setup of robot, bush and deep learning model

3.2 DATASET DETAILS

To train and evaluate the deep learning model, the project uses the Laboro Tomato Dataset, which is collected from GitHub. [15] The dataset consists of 643 images including cherry and normal-sized tomato clusters growing on bushes with varying lighting conditions, leaf occlusions, and multiple ripeness stages. The following figure represents six example images from the dataset.

Figure 3-6 Example images of dataset

To ensure accurate classification, each image is carefully processed by cropping objects and extracting their ripeness stages individually. This step enables precise classification of each tomato and facilitates the training process of the deep learning model. The dataset is augmented as it is an essential step to enhance the training process and improve the robustness of the deep learning model. In this project, several augmentation methods are applied to the original dataset:

- Horizontal flipping
- Vertical flipping
- 180° rotation

These augmentation techniques allowed the dataset to be expanded to 2572 images. This augmented dataset provides a diverse range of variations in tomato appearance, lighting conditions, and orientations, enabling the model to generalize better and improve its performance in different scenarios.

Chapter Four

RESULTS AND DISCUSSION

This section presents the initial results of our research on the application of the deep learning models YOLOv5, YOLOv7 and Faster RCNN for automating tomato harvesting. We trained these models using the mentioned dataset of images of labelled tomatoes and classified into fully ripened, half-ripened, and unripened. The primary aim of this module is to determine the performance of the models in detecting tomatoes and classifying between the ripeness stages.

We will provide an overview of the results concerning the precision, recall, F-measure and Mean Average Precision (mAP) of each model. The first three metrics are most commonly used in machine learning and information retrieval tasks, whereas the mAP is commonly used to analyse the performance of object detection and segmentation systems. The equations used in the system are as follows:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (1)$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (2)$$

$$F - Measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (3)$$

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \quad (4)$$

Precision is the ratio of true positives to the total number of positively predicted objects, visible in Equation (1). The recall is the ratio between the true positives predicted by the model and the total number of actual positives in the dataset, shown in Equation (2). The F-Score or F-Measure is a weighted mean of precision and recall, combining both metrics as shown in Equation (3). Finally, the mAP is the average precision for varying recall values. It utilizes the Intersection over Union (IoU) to produce accepted results as the IoU is the intersection area of the predicted bounding box to the actual bounding box. By default, most models are trained with having an IoU greater than 50% and less than 95%. The mAP is measured using Equation (4).

4.1 YOLOV5 RESULTS

The YOLOv5 model yielded the highest metrics for the proposed system, it was able to reach a precision value of 95% and a recall value of 83%. This is used in Eq. (3) to calculate the F-measure of the model. The resulted score of 89% is significantly higher than the rest of the used models. In addition, the model scored an mAP of 82% across the range of confidence thresholds from 0.5 and 0.95 highlighted the model's robustness in detecting tomatoes with varying degrees of certainty. Below is a figure of some example results from the model.

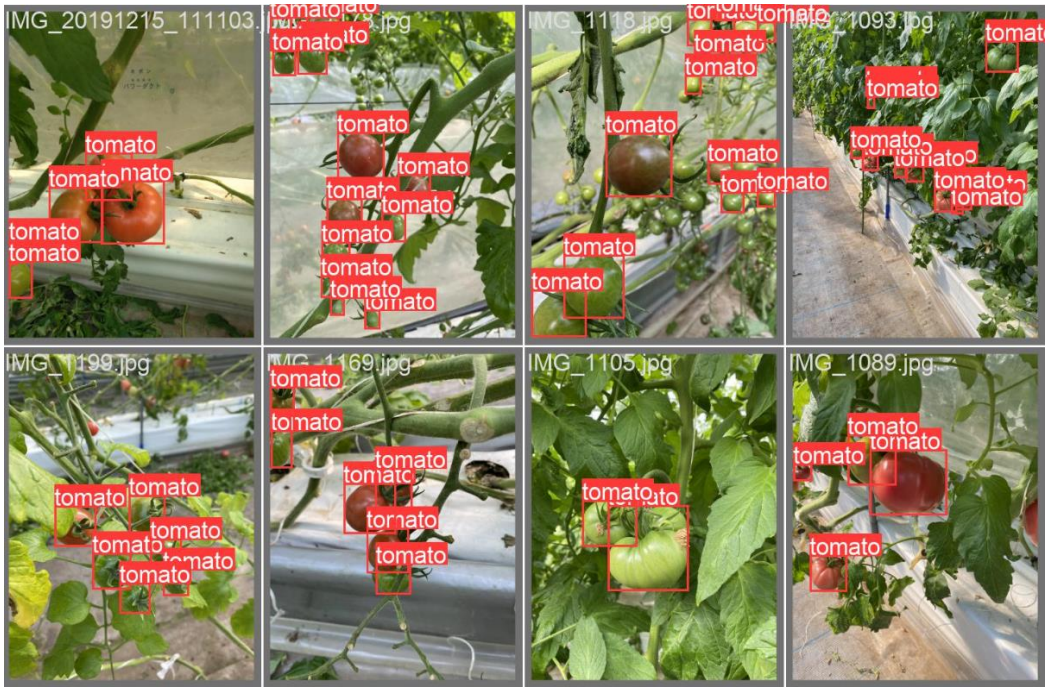

Figure 4-1 Example results from YOLOv5

The model's confusion matrix is shown in the figure below.

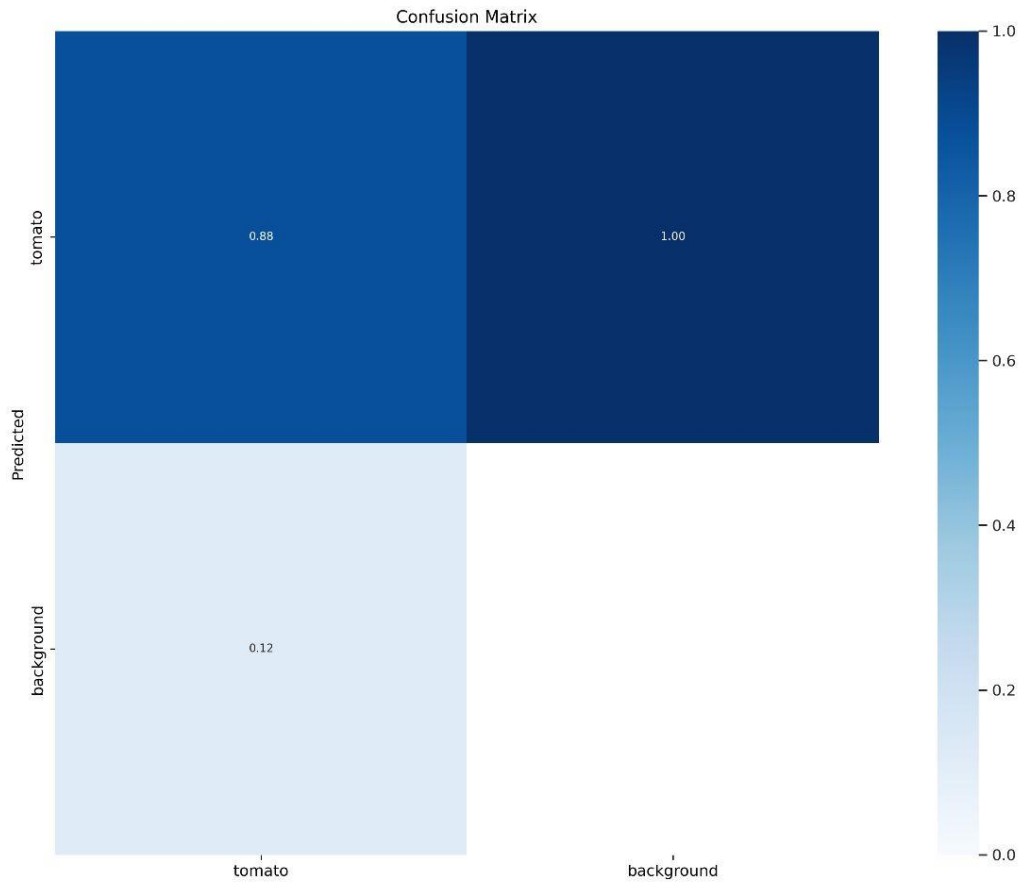

Figure 4-2 Confusion matrix for YOLOv5

4.2 YOLOV7 RESULTS

The YOLOv7 model had impressive, yet lower metrics compared to the previous model. With a precision of 88% and recall of 91%, the concluded F-measure from the precision and recall is 89%, and an mAP of 80%. These values were gathered through 15 epochs and with a batch size of 16, as well as scaling the images to a resolution of 640 x 640 pixels. The following image represents an example result acquired from testing the model. It is obvious that the model is less suitable to the used dataset compared to YOLOv5. The figure below represents some of the model's results.

Figure 4-3 Example results from YOLOv7

4.3 FASTER RCNN RESULTS

The model was considered the slowest in the training phase and unexpectedly resulted in lower metrics than anticipated. The model had resulted with a precision of 89% and a recall of 75%, leading to an F-score of 81%. In addition, the mAP resulted was 70%. This model was not in its most ideal state to be used in the project as both YOLO models had shown significantly higher results than it. The figure below represents some example results.

Figure 4-4 Example results from Faster RCNN

4.4 GENERAL STATISTICS

Generally, all three models were a success and had impressive performance. The following figure shows a comparison between the metrics for all three models.

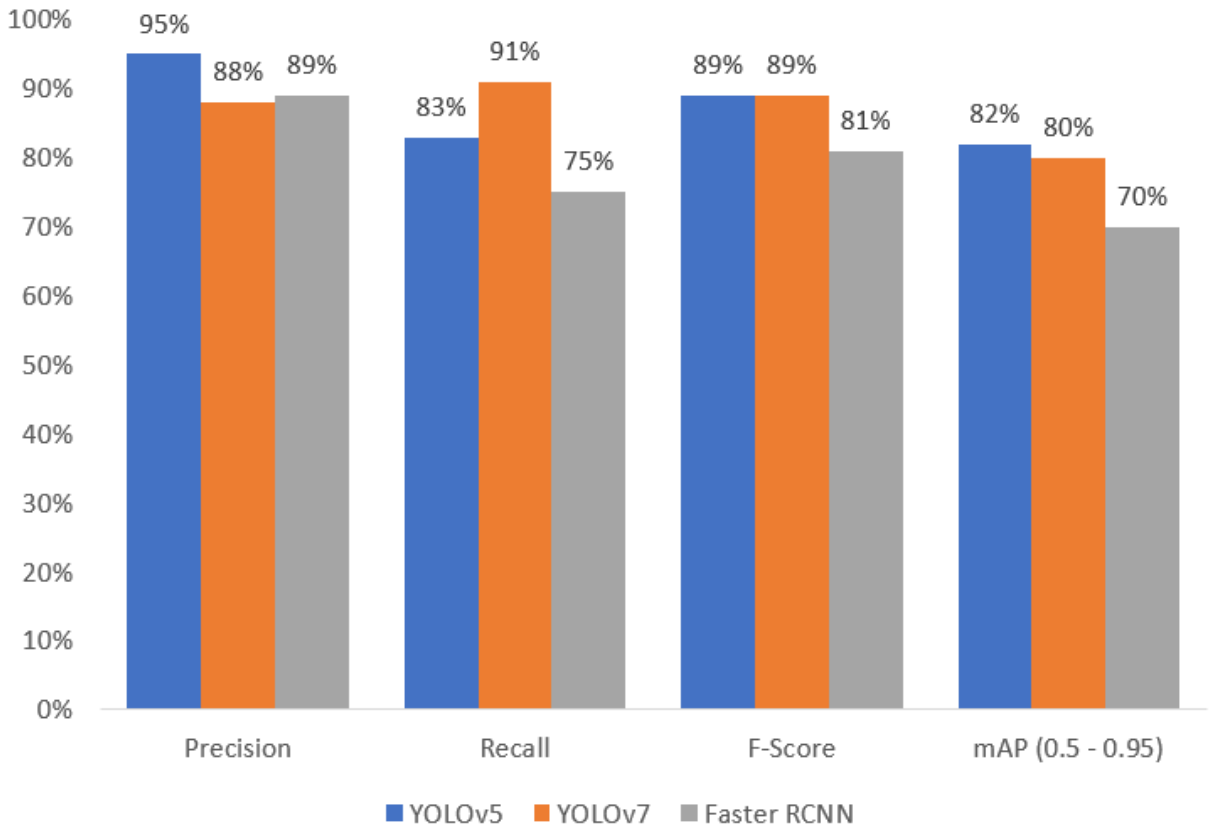

Figure 4-5 Statistical graph for precision, recall, F-score and mAP for all models

In addition to the detection, following figure shows a comparison between the accuracies of the classification models ResNet-50 and VGG-16

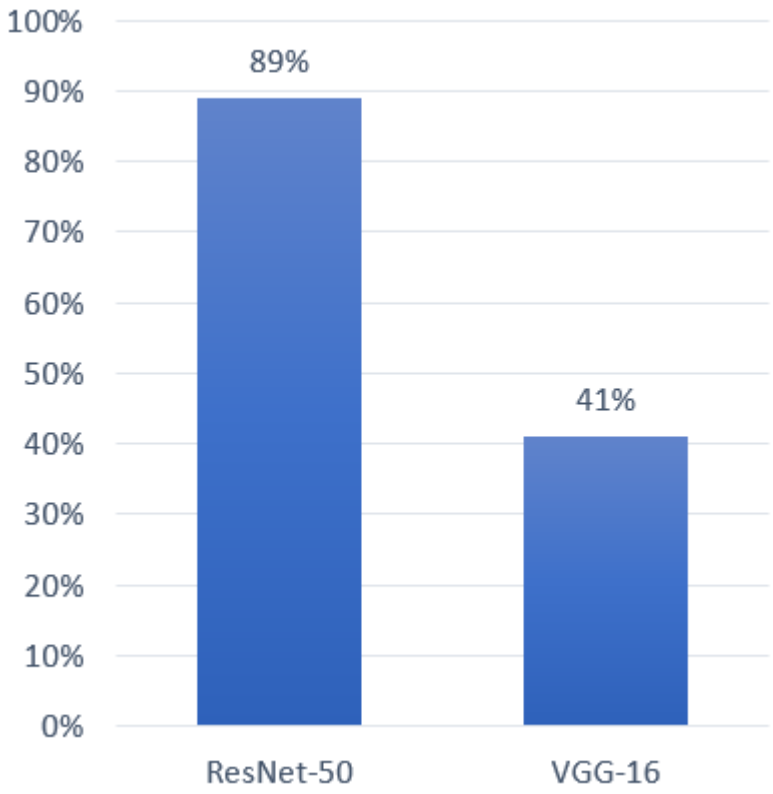

Figure 4-6 Statistical graph for accuracies of ResNet-50 and VGG-16

Chapter Five

CONCLUSION

The models employed in this project demonstrated a remarkable ability to accurately detect tomatoes. However, like any deep learning model, they are not without limitations and necessitate ongoing refinement to ensure optimal performance across diverse environments and achieve even higher levels of accuracy. The YOLOv5 model achieved higher metrics than the YOLOv7 and Faster RCNN with the high mAP of 87% and F-Score of 97%. Crucially, the models need to be robust in handling tomatoes under varying lighting conditions, angles, and sizes. To address these challenges, future work will involve the incorporation of additional data into the dataset, enabling the models to learn from a broader range of tomato variations. Moreover, efforts will be directed towards the development of a user-friendly mobile application that logs the system's actions comprehensively. This application will enhance transparency for consumers by providing insights into the success of harvests, along with statistical information such as tomatoes harvested count and harvest time. The entire project can be a great step to solving the agricultural issues of crop loss and tediousness of manual harvesting.

REFERENCES

[1] World Food Programme "Egypt", Available: https://www.wfp.org/countries/egypt Accessed January 15th 2023

[2] Egypt Today "Egypt ranks 5th worldwide in tomato production: FAO", Available: https://www.egypttoday.com/Article/3/116739/Egypt-ranks-5th-worldwide-in-tomato-production-FAO Accessed January 17th 2023

[3] D. SepúLveda, R. Fernández, E. Navas, M. Armada and P. González-De-Santos, "Robotic Aubergine Harvesting Using Dual-Arm Manipulation", in IEEE Access, vol. 8, pp. 121889-121904, 2020, Available: https://ieeexplore.ieee.org/abstract/document/9133102

[4] Y. Onishi, T. Yoshida, H. Kurita, T. Fukao, H. Arihara and A. Iwai, "An automated fruit harvesting robot by using deep learning", in Robomech J 6, 13, 2019. Available: https://doi.org/10.1186/s40648-019-0141-2

[5] Y. Xiong, Y. Ge, L. Grimstad and P. From, "An autonomous strawberry-harvesting robot: Design, development, integration, and field evaluation", in Journal of Field Robotics. 37, 2019, Available: https://www.researchgate.net/publication/335036773_An_autonomous_strawberry-harvesting_robot_Design_development_integration_and_field_evaluation

[6] O. Lawal, "YOLOMuskmelon: Quest for Fruit Detection Speed and Accuracy Using Deep Learning", in IEEE Access, vol. 9, pp. 15221-15227, 2021, Available: https://ieeexplore.ieee.org/document/9328755

[7] O. Toon, M. Zakaria, A. Ab. Nasir, A. Majeed, C, Tan and L. Ng, "Autonomous Tomato Harvesting Robotic System in Greenhouses: Deep Learning Classification", in MEKATRONIKA, 2019, Available: https://journal.ump.edu.my/mekatronika/article/view/1148

[8] duran67, "YOLOv5 Algorithm", Available: https://github.com/ultralytics/yolov5 Accessed January 20th 2023

[9] Taran Marley, "YOLOv7 Algorithm", Available: https://www.kaggle.com/code/taranmarley/yolo-v7-object-detection/input Accessed April 25th 2023

[10] Ben Manor, "Faster RCNN Algorithm", Available: https://www.kaggle.com/code/benmanor/crater-object-detection-using-faster-rcnn Accessed June 29th 2023

[11] R. Xu, H. Lin, K. Lu and L. Cao, "A Forest Fire Detection System Based on Ensemble Learning", in Forests, 12, 2021, Available: https://www.researchgate.net/publication/349299852_A_Forest_Fire_Detection_System_Based_on_Ensemble_Learning

[12] Roboflow "What is YOLOv7? A Complete Guide", Available: https://blog.roboflow.com/yolov7-breakdown Accessed July 5th 2023

[13] Z. Deng, H. Sun, S. Zhou and J. Zhao, "Multi-scale object detection in remote sensing imagery with convolutional neural networks", in ISPRS Journal of Photogrammetry and Remote Sensing, 145, 2018

[14] cgxeiji, "CGx-InverseK", Available: https://github.com/cgxeiji/CGx-InverseK/tree/master Accessed July 4th 2023

[15] Etoye, "Laboro Tomato Dataset", Available: https://github.com/laboroai/LaboroTomato Accessed January 17th 2023

APPENDICES

1- Arduino code

```
#include <Arduino.h>
#include <Braccio.h>
#include <Servo.h>
#include <InverseK.h>
#include <string.h>

Servo base;
Servo shoulder;
Servo elbow;
Servo wrist_ver;
Servo wrist_rot;
Servo gripper;

unsigned int _baseAngle = 90;
unsigned int _shoulderAngle = 90;
unsigned int _elbowAngle = 90;
unsigned int _wrist_verAngle = 90;
unsigned int _wrist_rotAngle = 90;
unsigned int _gripperAngle =12; //closed

int x = -150;
int y = 0;
int z = 100;
int sign = 0;

int x_old = -150;
int y_old = 0;
int z_old = 100;

int movementSpeed = 23;
int grippingSpeed = 10;

int incoming[7];

void setup()
{
  Serial.begin(9600);
  Serial.setTimeout(1);

  Braccio.begin();

  Link base, upperarm, forearm, hand;
  base.init(71, b2a(0.0), b2a(180.0));
  upperarm.init(125, b2a(15.0), b2a(165.0));
  forearm.init(125, b2a(0.0), b2a(180.0));
  hand.init(192, b2a(0.0), b2a(180.0));

  // Attach the links to the inverse kinematic model
```

```
  InverseK.attach(base, upperarm, forearm, hand);

  float a0, a1, a2, a3;

  // InverseK.solve() return true if it could find a solution and false if not.

  // Calculates the angles without considering a specific approach angle
  // InverseK.solve(x, y, z, a0, a1, a2, a3)
  if(InverseK.solve(-150, 0, 100, a0, a1, a2, a3)) {
    // Serial.print(a2b(a0)); Serial.print(',');
    // Serial.print(a2b(a1)); Serial.print(',');
    // Serial.print(a2b(a2)); Serial.print(',');
    // Serial.println(a2b(a3));
    _baseAngle = a2b(a0);
    _shoulderAngle = a2b(a1);
    _elbowAngle = a2b(a2);
    _wrist_verAngle = a2b(a3);
  } else {
    Serial.println("No solution found!");
  }

Braccio.ServoMovement(movementSpeed,_baseAngle,_shoulderAngle,_elbowAngle,_wrist_verA
ngle,_wrist_rotAngle,_gripperAngle);
}

bool goTo(int x, int y, int z) {
  float a0, a1, a2, a3;

  // InverseK.solve() return true if it could find a solution and false if not.

  // Calculates the angles without considering a specific approach angle
  // InverseK.solve(x, y, z, a0, a1, a2, a3)
  if(InverseK.solve(x, y, z, a0, a1, a2, a3)) {
    _baseAngle = a2b(a0);
    _shoulderAngle = a2b(a1);
    _elbowAngle = a2b(a2);
    _wrist_verAngle = a2b(a3);
    return true;
  } else {
    Serial.println("No solution found!");
    return false;
  }
}

void loop()
{
  if (Serial.available() > 6){
    // fill array
    for (int i = 0; i < 7; i++){
      incoming[i] = Serial.read();
    }
    // use the values
    x = (incoming[0] << 8) | incoming[1];
    y = (incoming[2] << 8) | incoming[3];
    z = (incoming[4] << 8) | incoming[5];
```

```
    sign = incoming[6];
    if (sign & 1) x = -x;
    if (sign & 2) y = -y;
    if (sign & 4) z = -z;

    if (goTo(x, y, z)) {

Braccio.ServoMovement(movementSpeed,_baseAngle,_shoulderAngle,_elbowAngle,_wrist_verA
ngle,_wrist_rotAngle,10);
      delay(1000);

Braccio.ServoMovement(grippingSpeed,_baseAngle,_shoulderAngle,_elbowAngle,_wrist_verA
ngle,_wrist_rotAngle,60);
      delay(1000);
      goTo(-150, 0, 100);

Braccio.ServoMovement(movementSpeed,_baseAngle,_shoulderAngle,_elbowAngle,_wrist_verA
ngle,_wrist_rotAngle,60);
      delay(1000);

Braccio.ServoMovement(grippingSpeed,_baseAngle,_shoulderAngle,_elbowAngle,_wrist_verA
ngle,_wrist_rotAngle, 10);
      delay(1000);
    }
  }
}


// Quick conversion from the Braccio angle system to radians
float b2a(float b){
  return b / 180.0 * PI - HALF_PI;
}

// Quick conversion from radians to the Braccio angle system
float a2b(float a) {
  return (a + HALF_PI) * 180 / PI;
}
```

2- Python script

```
import torch
import cv2
import serial
import struct
import math
import time
import pathlib
import torch.nn.functional as F
from torchvision import transforms as T

temp = pathlib.PosixPath
pathlib.PosixPath = pathlib.WindowsPath

arduino = serial.Serial(port='COM3', baudrate=9600, timeout=.1)
```

```
detect = torch.hub.load('.', 'custom', path='Object_Detection.pt', source='local',
force_reload=True)
classify = torch.hub.load('.', 'custom', path='classification.pt', source='local',
force_reload=True)

IMAGENET_MEAN = 0.485, 0.456, 0.406
IMAGENET_STD = 0.229, 0.224, 0.225

viewAngleH = 38.2
viewAngleW = 69.4
distance = 30  # cm
cameraHeight = 29
CameraX = -8.4

start_time = time.time()
DELAY_SECONDS = 10

label_color = {'ripened': (255,0,0),
               'half_ripened': (255, 172, 28),
               'unripened': (0,255,0)
               }

def Send_Cordinates(x, y, z):
    x = round(x)
    y = round(y)
    z = round(z)

    sign = 0
    if x < 0:
        sign = sign ^ 1
        x = -x
    if y < 0:
        sign = sign ^ 2
        y = -y
    if z < 0:
        sign = sign ^ 4
        z = -z

    xl = x >> 8
    xr = x & 0xFF

    yl = y >> 8
    yr = y & 0xFF

    zl = z >> 8
    zr = z & 0xFF

    arduino.write(struct.pack('>BBBBBBB', xl, xr, yl, yr, zl, zr, sign))

def Calculate_Real_World_Cordinates(x, y, w, h):
    real_w = math.tan(viewAngleW / 2 * math.pi / 180.0) * distance * 2
    real_h = math.tan(viewAngleH / 2 * math.pi / 180.0) * distance * 2

    real_x = (x / w) * real_w;
    real_y = (y / h) * real_h
```

```
    return real_x, real_y, real_w, real_h

def classify_transforms(size=224):
    return T.Compose([T.ToTensor(), T.Resize(size), T.CenterCrop(size),
T.Normalize(IMAGENET_MEAN, IMAGENET_STD)])

# define a video capture object
vid = cv2.VideoCapture(1)

while (True):
    if time.time() - start_time > DELAY_SECONDS:
        classified_tomato_with_coordinates = []
        ret, frame = vid.read()
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        detect_results = detect(frame)
        detect_results_list = detect_results.xyxy[0].cpu().numpy().tolist()
        print('result count: ', len(detect_results_list))
        for tomato in detect_results_list:
            x1 = tomato[0]
            y1 = tomato[1]
            x2 = tomato[2]
            y2 = tomato[3]
            w = x2 - x1
            h = y2 - y1
            croppedImg = frame[round(y1):round(y2), round(x1):round(x2)]
            croppedImg = cv2.resize(croppedImg, (224, 224),
interpolation=cv2.INTER_AREA)
            transformations = classify_transforms()
            convert_tensor = transformations(croppedImg)
            convert_tensor = convert_tensor.unsqueeze(0)
            classification = classify(convert_tensor)
            pred = F.softmax(classification, dim=1)
            for i, prob in enumerate(pred):
                top5i = prob.argsort(0, descending=True)[:5].tolist()
                # text = '\n'.join(f'{prob[j]:.2f} {classify.names[j]}' for j in
top5i)
                # print(text)
                frame = cv2.rectangle(frame, (round(x1), round(y1)), (round(x2),
round(y2)), label_color[classify.names[top5i[0]]], 5)
                classified_tomato_with_coordinates.append(classify.names[top5i[0]])

        for x in range(len(classified_tomato_with_coordinates)):
            if classified_tomato_with_coordinates[x] == 'ripened':
                x1 = detect_results_list[x][0]
                y1 = detect_results_list[x][1]
                x2 = detect_results_list[x][2]
                y2 = detect_results_list[x][3]
                w = x2 - x1
                h = y2 - y1
                real_x, real_y, real_w, real_h = Calculate_Real_World_Cordinates(x1 +
w / 2, y1 + h / 2,

frame.shape[1], frame.shape[0])
                real_x = real_x - real_w / 2 - CameraX
```

```
                real_y = (real_h - real_y) + (cameraHeight - real_h / 2)
                Send_Cordinates(distance * 10, real_x * 10, real_y * 10)
                print(real_x, real_y)
                break

        cv2.imshow('Autonomous Fruit Harvesting Robot', cv2.cvtColor(frame,
cv2.COLOR_RGB2BGR))
        start_time = time.time()
    # input("Press Enter to continue...")
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

vid.release()
cv2.destroyAllWindows()
```

3- Augmentation by flipping code

```
import cv2
import numpy as np
from PIL import Image
import os
import glob
import math

inputFolder = os.getcwd()+ "\\train"
inputLabel = os.getcwd() + "\\labels"
folderlen = len(inputFolder)

os.mkdir('Flip')

for img in glob.glob(inputFolder + "/*.jpg"):
    # print(img)
    image = cv2.imread(img)
    label = open(inputLabel+ "\\" + img[folderlen:-4] + ".txt", "r")
    labelOut = open("Flip/" + img[folderlen:-4] + ".txt", "w")
    for line in label:
        line = line.split()
        cid, cx, cy, w, h = int(line[0]), float(line[1]), float(line[2]) ,
float(line[3]), float(line[4])

        cy -= 0.5
        cy = -cy
        cy += 0.5

        labelOut.write(f"{cid} {cx} {cy} {w} {h}\n")

    height, width = image.shape[:2]
    flip = cv2.flip(image,0)
    cv2.imwrite("Flip/" + img[folderlen:], flip)
    label.close()
    labelOut.close()
```

4- Augmentation by rotation code

```
import cv2
import numpy as np
from PIL import Image
import os
import glob
import math

inputFolder = os.getcwd()+ "\\train"
inputLabel = os.getcwd() + "\\labels"
folderlen = len(inputFolder)

os.mkdir('Rotation')

angle = 270

for img in glob.glob(inputFolder + "/*.jpg"):
    # print(img)
    image = cv2.imread(img)
    label = open(inputLabel+ "\\" + img[folderlen:-4] + ".txt", "r")
    labelOut = open("Rotation/" + img[folderlen:-4] + ".txt", "w")
    for line in label:
        line = line.split()
        cid, cx, cy, w, h = int(line[0]), float(line[1]), float(line[2]) ,
float(line[3]), float(line[4])

        cx -= 0.5
        cy -= 0.5
        x = [cx - w / 2, cx - w / 2, cx + w / 2, cx + w / 2]
        y = [cy - h / 2, cy + h / 2, cy - h / 2, cy + h / 2]

        nx = []
        ny = []

        for i in range(4):
            nx.append(x[i] * math.cos(-angle*math.pi/180) - y[i] * math.sin(-
angle*math.pi/180))
            ny.append(x[i] * math.sin(-angle*math.pi/180) + y[i] * math.cos(-
angle*math.pi/180))

        xmin = min(nx)
        xmax = max(nx)
        ymin = min(ny)
        ymax = max(ny)

        nw = xmax - xmin
        nh = ymax - ymin

        ncx , ncy = nw / 2 + xmin, nh / 2 + ymin
        ncx += 0.5
        ncy += 0.5
        labelOut.write(f"{cid} {ncx} {ncy} {nw} {nh}\n")
```

```
height, width = image.shape[:2]
rotation_matrix = cv2.getRotationMatrix2D((width/2,height/2),angle,1)
rotated_image = cv2.warpAffine(image,rotation_matrix,(width,height))
cv2.imwrite("Rotation/" + img[folderlen:], rotated_image)
label.close()
labelOut.close()
```