CROP QUALITY MONITORING USING RASPBERRY PI

MINI PROJECT REPORT

SUBMITTED

BY

Bhavadharani 107120028
Harini M 107120048
Pandya Margi Kalpeshkumar 107120084

Under the supervision of

Dr. Ankur Singh Rana

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
NATIONAL INTITUTE OF TECHNOLOGY TIRUCHIRAPPALLI
TAMIL NADU 620015

BATCH: 2020-2024
Session: January 2023

Abstract

In this world of increasing population and varying requirements, food is a staple that every human being requires to live a healthy and stress-free life. It is in everyone's good interests to increase the food quantity by increasing the quality and quantity. The major issue faced by farmers these days is insect infestation which ruin perfectly good crops and reduces the overall efficiency of a field. To take the agriculture field a step further into the future, the proposed automated system is created which uses the Raspberry Pi and object detection in deep learning to identify small insects that might have been missed when checking manually.

This will reduce man made errors and ensure that none of the crops growing are hindered, by insects feasting on them. This system collects images of the crops and identifies the insects and alerts the farmer of the infection for immediate action to be taken.

The basic methodology used in this pest detection is object detection. First, the image of the crop is taken using the camera module attached to the microprocessor- in this case, Raspberry Pi- and is stored into a folder. Next the image is sent to the object detection algorithm (a pre-trained model in TensorFlow Framework) in google collab for training. The trained weights from the model are then deployed on the Raspberry Pi. The algorithm is then run on the microprocessor to test the unseen images and video feed from the pi camera.

From this study, we can find out how effective using Raspberry pi with TensorFlow Lite model for object detection is for detecting insects in the crops in the early stages to reduce the losses due to crop infestations.

List of Figures/Tables/Nomenclature:

Table of Contents

1. Introduction

1.1 Background

In this world of a growing population of 8 billion people, overpopulation is becoming a very serious issue due to the lack of resources. Even through various people have various requirements, the main thing everyone needs is a stable source of food to keep both the mind and body healthy. The land dedicated to growing crops has also been declining due to the increasing industrialization. Hence, the importance of making sure we use the land effectively is important. For this we use the help of machines.

1.2 Motivation

One of the major threats to food security is insect infestation which amounts to billions of dollars' worth of crops going down the drain. The method of manually checking for pests is very time consuming and inefficient and can also lead to a lot of man-made errors which will result in a loss. To reduce the damage made by infestations, early detection of the infestation is necessary, and it will allow the farmer to take further action in the early stages which will give him/her various chances to fix the problem.

1.3 Objective

The objective of this project is to minimize the losses caused in the agriculture sector due to insect infestations. Since it is one of the main reasons for losses, being able to automate the process of detecting infestations at the early stages will significantly improve the yield and give more profits to the workers. It will also reduce the errors, manual labor and time required when compared to manually checking for infestations.

1.4 Organization

Students at National Institute of Technology, Tiruchirappalli.

2. Theoretical Background

The insect detection works using the EfficientDet algorithm in object detection to detect insects and alert the farmer. The image must be fed into the trained model and the model will detect if there is an insect. The image is taken using the camera module in the raspberry pi and stored to be sent to the trained model.

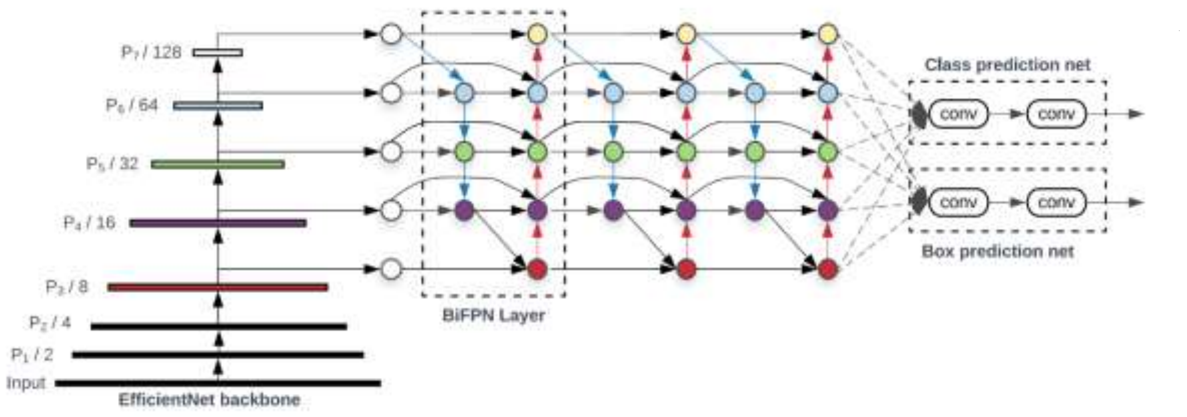


Fig 1.

A feature fusion network and a box/class prediction network make up the two primary parts of EfficientDet's object detecting head. A collection of fused feature maps is produced by the feature fusion network by fusing feature maps from various Efficient Net backbone tiers. This enables the model to recognize items of various sizes and scales.

The fused feature maps are sent into the box/class prediction network, which uses them to forecast the bounding boxes and class probabilities for each item in the picture. A set of fully linked layers and a set of convolutional layers are used to accomplish this. To forecast the final bounding boxes, the box/class prediction network additionally employs a collection of anchor boxes, which are preconfigured bounding boxes with various sizes and aspect ratios.
A weighted bi-directional feature pyramid network (BiFPN) to improve feature fusion, a compound scaling approach to improve model size and performance, and a focused loss function to address class imbalance are among the additional optimizations that are included in EfficientDet.

The model will use different equations to detect. A model is first pre-trained using different photo sets to learn how to find the object and identify it. The weights will be determined when the model gets trained so that when the input image is sent, the formulas will be applied with the given weights to do the object detection.

To give the image as an input to the object detection algorithm, we will have to obtain an image through the raspberry pi. Here the camera module is connected and interfaced with the raspberry pi so that the camera module can take pictures and save it in a file mentioned. This image will then be sent to the algorithm so that object detection takes place.

3. Literature Review

Several methods of object detection using microcontrollers have been researched upon which has resulted in many successful outcomes. Object detection using microcontrollers can be used for various purposes. Since this is something that can be used to improve pest control, one of the main reasons for losses in agriculture, in an affordable way, research will continue to happen in this field so that an optimal solution is found.

To start off, a study done by [1] involved detecting pests called thrips using an Arduino and a machine learning model based on Haar cascade classifier. It was used in the onion crops, and it was successful in automatic detecting of the insects.

A study by [2] was conducted. He had used an object detecting system with a Raspberry pi and a TensorFlow deep learning model to find the presence of brown planthoppers in the rice fields. The result of the study showed that this method was in fact an effective method that can be used to detect insects and can be used to prevent insect infestation.

In another study by [3], a method was proposed for real time pedestrian detection using YOLOv3 and Raspberry Pi. This showed that object detection is possible in real time.

From all these studies that have been done so far, we can see that using object detection and microcontrollers is a very cost effective and very efficient way for identification of pests in the earlier stages of infestation to improve the overall crop quality and quantity.

4. Methodology/Modelling

In this project, we have 3 main steps:

1. Collecting Dataset using PI camera:
2. Training in TensorFlow Lite
3. Object Detection in Raspberry Pi

4.1. Collecting Dataset using Pi Camera:

Using Raspberry Pi 3 B+, and Light camera module for Raspberry Pi we can develop a system to take the pictures of crops. And we can have large number of similar images to train our model. This can be done remotely also from a certain range using Rasp Controller application software available in both windows and iOS for controlling Raspberry Pi from phone. Enabling the camera in Raspberry Pi is the first step to taking a picture. And with simple commands in Raspberry Pi command prompt, the camera module takes a picture. When the camera module takes a picture, we can identify it with a red light appearing in the module. These steps can also be done using a programming in Thyone python IDE.

4.2. Training using TensorFlow lite:

TensorFlow Lite is an open source deep learning framework that has pre-trained models from TensorFlow to a specific format that can be optimized for speed or storage. This special format can be deployed on Linux based electronic devices like Raspberry Pi or Microcontroller and on edge devices like mobile phones.

To build a custom object detector in TensorFlow, we start off by creating a training data set, training the custom model and we deploy the model to the Raspberry Pi. To create the data set, we get the images required from the internet and annotate the images accordingly. To train the custom model, we use Google Collab on the Raspberry Pi to train the model. The model used is EfficientDet-Lite0. The model will then need to be trained by the custom inputs we input so that it will be able to detect the images we input. This model was then exported as a TensorFlow Lite model to the Raspberry pi.

4.3. Object Detection in Raspberry Pi

The trained model of EfficientDet is deployed on the Raspberry Pi. The Pi camera will capture the images/ video of the crops and will give to the model to perform object detection on the given classes. Here, the class label used is “caterpillar” and “centipede”.

4.4. Collecting the temperature and humidity data from sensors:

Using digital temperature and humidity sensor DHT11 and with the help of jumpers connected to Raspberry Pi using female-female jumpers. The temperature and humidity data of the surrounding environment can be measured and monitored in Raspberry Pi screen. In this, the physical connection from Raspberry Pi to PC is given using HDMI port to get the data.

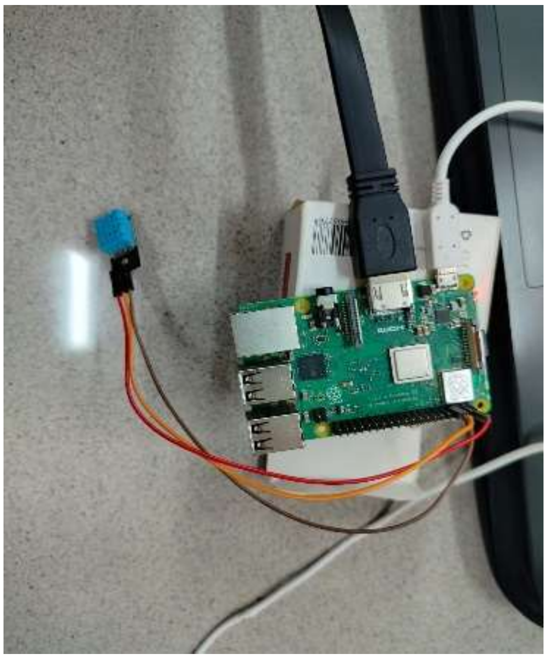
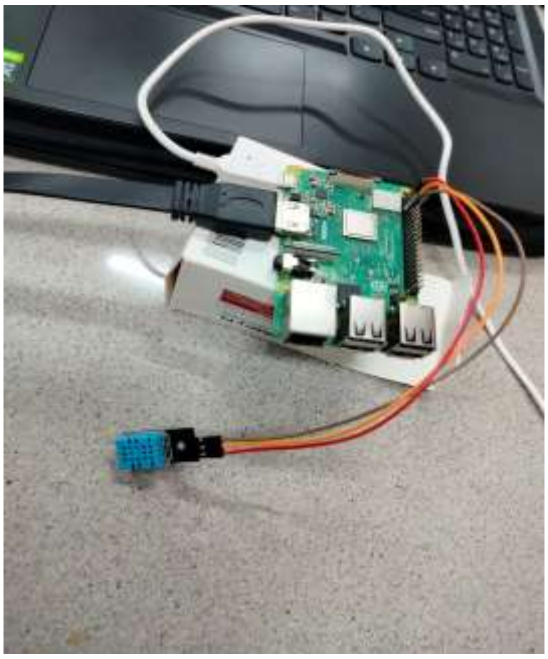
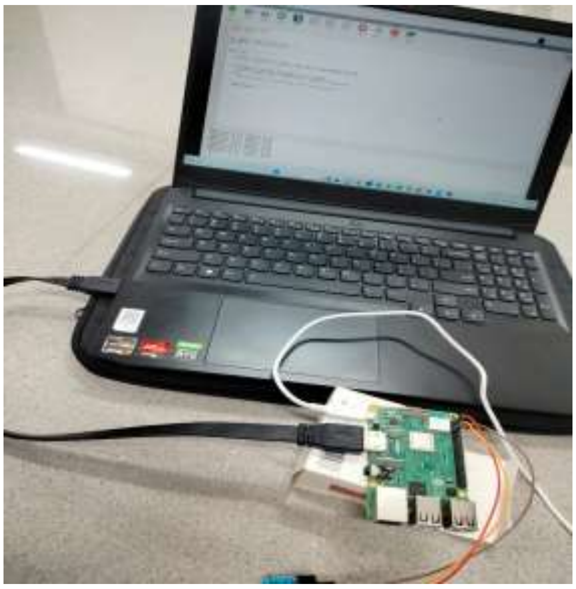




Fig 2

5. Implementation/Simulation

For implementation, we collected pictures of plants/ crops with and without insects using the Raspberry Pi. The images were then processed by the Raspberry Pi object detector and the results are shown.

5.1. Train Dataset:

Fig 3.

5.2. Results:

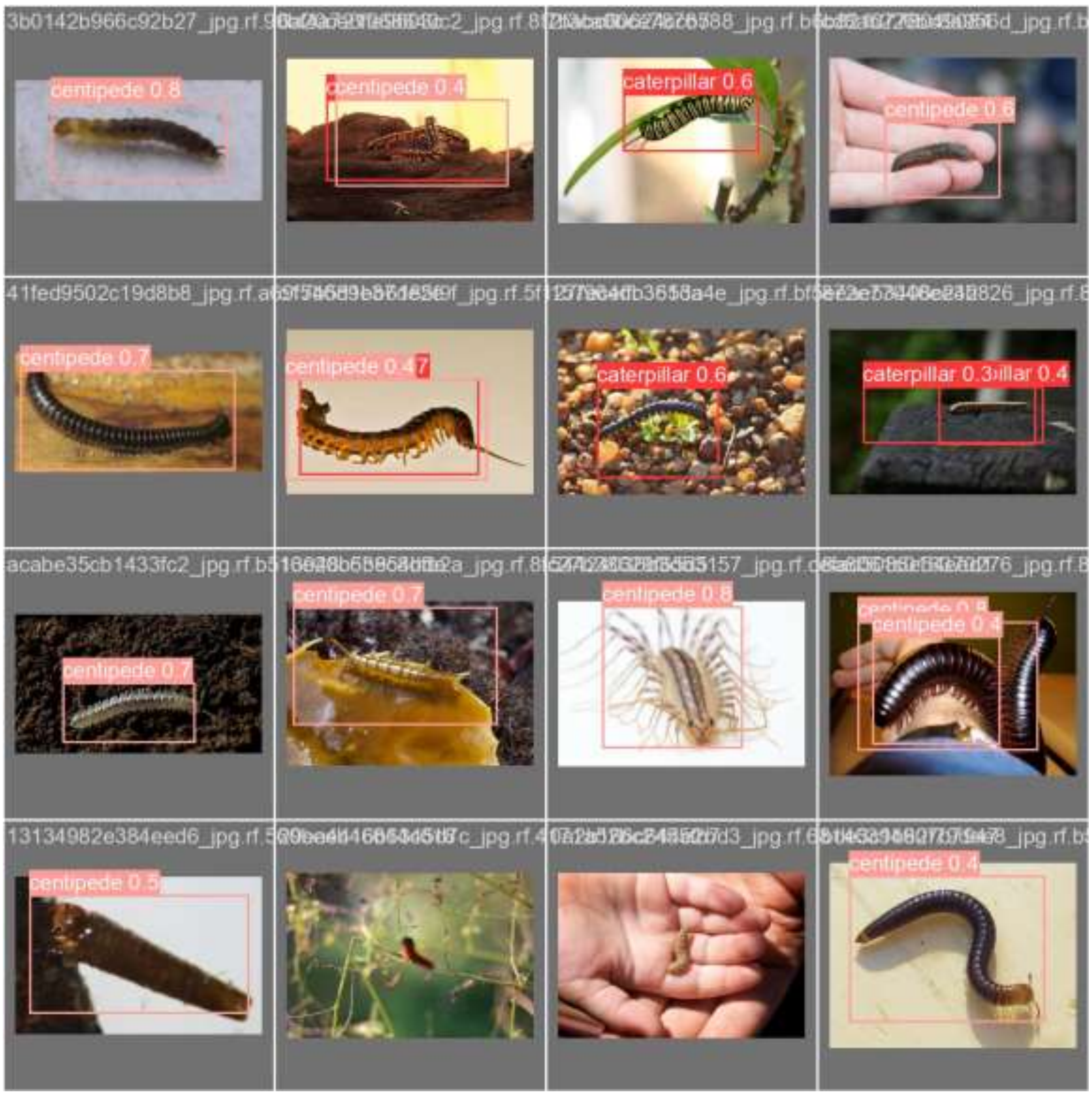
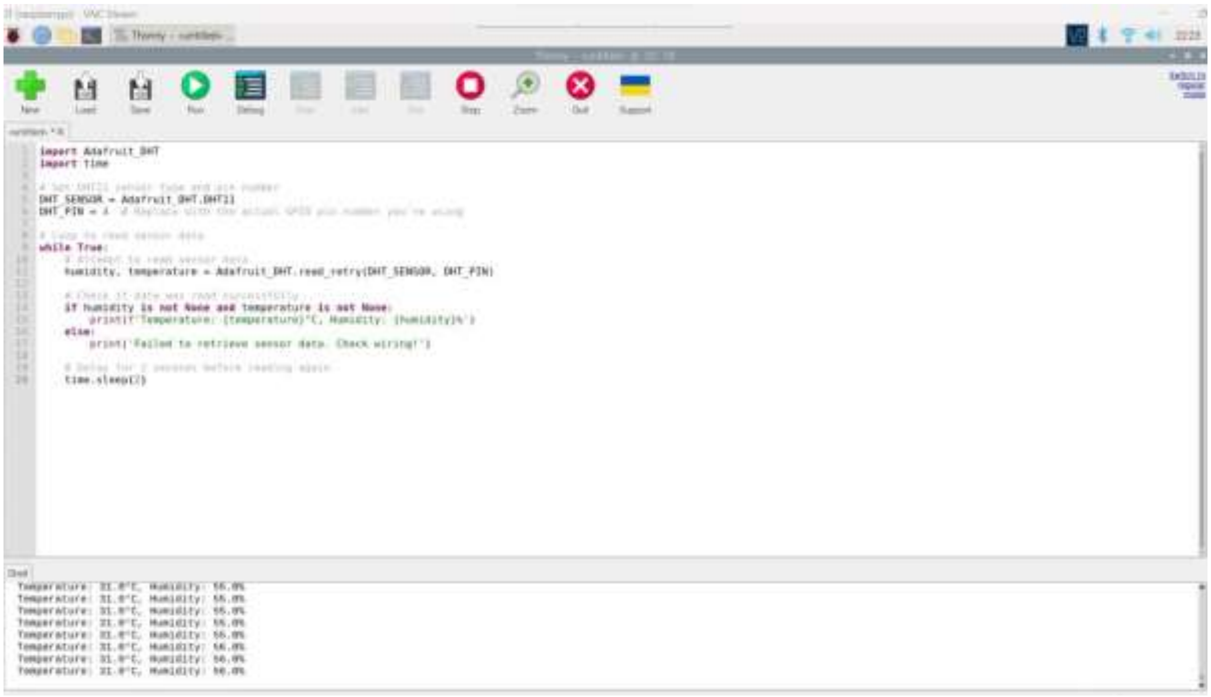


Fig 4.



```python
import Adafruit_DHT
import time

# Set DHT11 sensor type and pin number
DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 4  # Replace with the actual GPIO pin number you're using

# Loop to read sensor data
while True:
    # Attempt to read sensor data
    humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR, DHT_PIN)

    # Check if data was read successfully
    if humidity is not None and temperature is not None:
        print(f'Temperature: {temperature}°C, Humidity: {humidity}%')
    else:
        print('Failed to retrieve sensor data. Check wiring!')

    # Delay for 2 seconds before reading again
    time.sleep(2)
```

```
Temperature: 31.0°C, Humidity: 55.0%
Temperature: 31.0°C, Humidity: 55.0%
Temperature: 31.0°C, Humidity: 55.0%
Temperature: 31.0°C, Humidity: 55.0%
Temperature: 31.0°C, Humidity: 55.0%
Temperature: 31.0°C, Humidity: 56.0%
Temperature: 31.0°C, Humidity: 56.0%
Temperature: 31.0°C, Humidity: 56.0%
```

Fig 5

5.3. Evaluation Metrics:

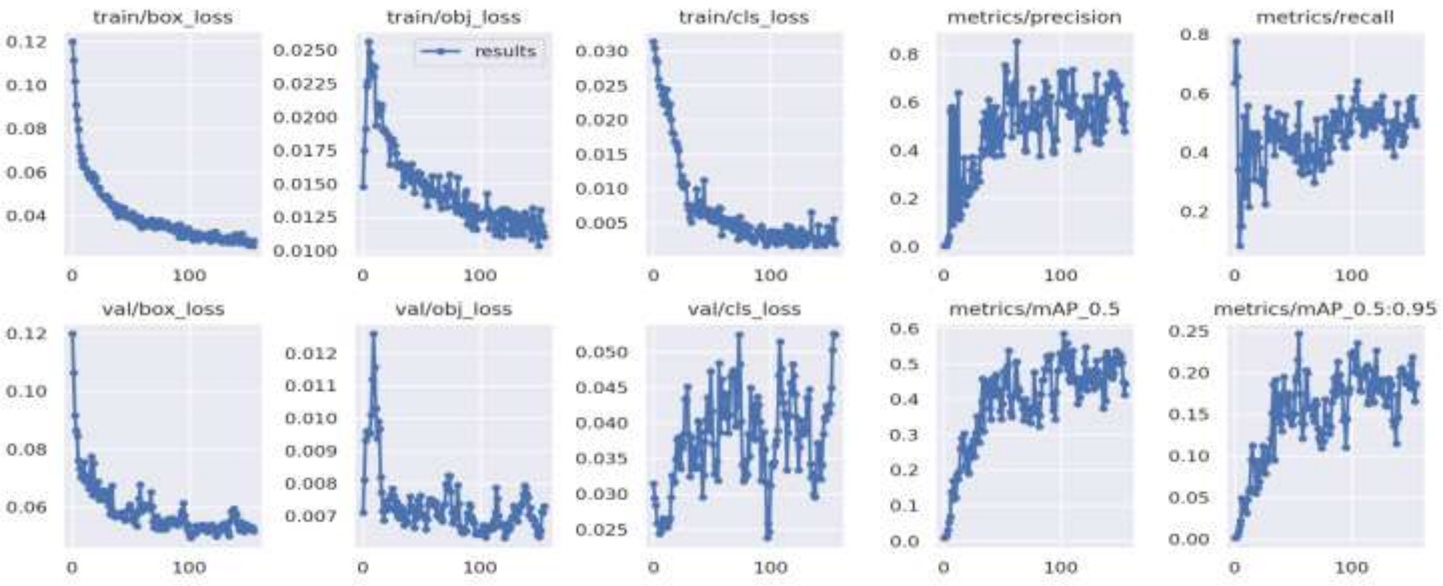


Fig 6.

mAP (mean Average Precision) is a popular evaluation metric used in object detection to measure the accuracy of a model's predictions. Precision measures how accurate the predictions of a model are, i.e., how many of the predicted objects are correct. Recall, on the other hand, measures how well a model can detect all the objects in an image. Average Precision (AP) combines both precision and recall giving an overall performance score for a model. mAP is the mean of AP values for all the classes of objects. It is calculated by first calculating the AP for each class and then taking the mean of all the AP values. **A higher mAP value indicates better performance of the model in detecting objects of different classes.**

Conclusion

In this project, we have used a Raspberry Pi and TensorFlow Lite to differentiate crops that have been infested with 2 main insects: Caterpillars and Centipede. This provides us with a highly effective and cost-effective.

To obtain high-quality photographs for this project, we made use of the Raspberry Pi camera module's capabilities. Additionally, we created a user-friendly interface that makes it simple for users to interact with the system and get alerts when pests are found. The accomplishment of this project creates new opportunities for the use of Raspberry Pi in pest control and agriculture. This system may be scaled out to cover greater regions and identify a wider variety of pests with additional upgrades and optimizations.

References:

[1] Shetty, S., Pai, R. M., & Nayak, N. (2019). Automatic Detection of Thrips Using Haar Cascade Classifier and Arduino. International Journal of Advanced Science and Technology, 28(14), 214-220.

[2] Yan, Y., Li, X., Li, M., Li, L., Li, Y., & Zhang, H. (2019). Detection of Brown Planthopper Based on Raspberry Pi and Deep Learning. Journal of Sensors, 2019, 1-9

[3] Tian, Y., Xu, W., Huang, Y., & Sun, Y. (2020). Real-time pedestrian detection based on YOLOv3 and Raspberry Pi. Journal of Physics: Conference Series, 1652(1), 012014.

[4] Anitha Ramachandran, Arun Kumar Sangaiah (2021). A review on object detection in unmanned aerial vehicle surveillance. International Journal of Cognitive Computing in Engineering, Volume 2, Pages 215-228, ISSN 2666-3074

[5] Online: https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/raspberry_pi: Code for general object detection.

Appendices:

Python Code for object detection:

Sourced: [5]

```
import argparse
import sys
import time

import cv2
from tflite_support.task import core
from tflite_support.task import processor
from tflite_support.task import vision
import utils

def run(model: str, camera_id: int, width: int, height: int, num_threads: int,
        enable_edgetpu: bool) -> None:
```

```
counter, fps = 0, 0
  start_time = time.time()

  cap = cv2.VideoCapture(camera_id)
  cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
  row_size = 20  # pixels
  left_margin = 24  # pixels
  text_color = (0, 0, 255)  # red
  font_size = 1
  font_thickness = 1
  fps_avg_frame_count = 10

  base_options = core.BaseOptions(
      file_name=model, use_coral=enable_edgetpu, num_threads=num_threads)
  detection_options = processor.DetectionOptions(
      max_results=3, score_threshold=0.3)
  options = vision.ObjectDetectorOptions(
      base_options=base_options, detection_options=detection_options)
  detector = vision.ObjectDetector.create_from_options(options)


  while cap.isOpened():
    success, image = cap.read()
    if not success:
      sys.exit(
          'ERROR: Unable to read from webcam. Please verify your webcam settings.'
      )

    counter += 1
    image = cv2.flip(image, 1)
    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    input_tensor = vision.TensorImage.create_from_array(rgb_image)
    detection_result = detector.detect(input_tensor)

    image = utils.visualize(image, detection_result)

    if counter % fps_avg_frame_count == 0:
      end_time = time.time()
      fps = fps_avg_frame_count / (end_time - start_time)
      start_time = time.time()

    fps_text = 'FPS = {:.1f}'.format(fps)
    text_location = (left_margin, row_size)
    cv2.putText(image, fps_text, text_location, cv2.FONT_HERSHEY_PLAIN,
                font_size, text_color, font_thickness)

    if cv2.waitKey(1) == 27:
      break
    cv2.imshow('object_detector', image)

  cap.release()
  cv2.destroyAllWindows()


def main():
  parser = argparse.ArgumentParser(
      formatter_class=argparse.ArgumentDefaultsHelpFormatter)
  parser.add_argument(
      '--model',
      help='Path of the object detection model.',
      required=False,
      default='efficientdet_lite0.tflite')
  parser.add_argument(
      '--cameraId', help='Id of camera.', required=False, type=int, default=0)
  parser.add_argument(
      '--frameWidth',
      help='Width of frame to capture from camera.',
      required=False,
      type=int,
      default=640)
  parser.add_argument(
      '--frameHeight',
      help='Height of frame to capture from camera.',
      required=False,
      type=int,
      default=480)
  parser.add_argument(
      '--numThreads',
      help='Number of CPU threads to run the model.',
      required=False,
      type=int,
      default=4)
  parser.add_argument(
      '--enableEdgeTPU',
      help='Whether to run the model on EdgeTPU.',
      action='store_true',
      required=False,
      default=False)
  args = parser.parse_args()

  run(args.model, int(args.cameraId), args.frameWidth, args.frameHeight,
      int(args.numThreads), bool(args.enableEdgeTPU))


if __name__ == '__main__':
  main()
```

Python code for camera setup:

```
from picamera import Picamera
import time

camera = Picamera();
camera.resolution = (1200,720)
camera.rotation = 180
time.sleep(2)

filename = "home/pi/MPMCProject/img.jpg"
camera.capture(filename)
print("Captured picture ")
```

Python code for Sensor Data:

```
import Adafruit_DHT

sensor = Adafruit_DHT.DHT11
pin = 4

humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)

print('Temperature={0:0.1f}C Humidity ={1:0.1f}%'.format(temperature, humidity))
else:
    print('Failed to read sensor data')
```