Low-Level Design (LLD)

Unmanned Ground Vehicle: Agriculture Solutions

Revision Number: 1.2
Last date of revision: 04/07/2021

Amarnath Goud
Vasanth Kumar

Document Version Control

Date Issued	Version	Description	Author
19th June 2021	1.0	First Draft	Hemanth ganesh Arvind Chauhan
30th June 2021	1.1	ROS Information and Images Snapshot o Dataset	Hemanth Ganesh Arvind Chauhan
5th June 2021	1.2	Scope, Risks, Constraints, SSD, Faster RCNN, Yolo V5 models and screen shots.	Amarnath Goud Vasanth Kumar

Table of Contents

List of figures

Abstract

UGVs can be operated with or without human presence. UGVs can be used in different areas including agriculture. Here we are focusing only on agriculture. There are a variety of purposes where it can be helpful in agriculture such as soil sampling, irrigation management, precision spraying, mechanical weeding, crop harvesting and disease detection etc.

Here, we will be considering one use case that is tomato crop disease detection i.e., we will collect the different data samples for healthy and infected crops, then we will use those image samples for training our deep learning model. Once the model is ready, we will deploy that model using UGV which will collect the images from the field and classify the crops into various categories i.e., whether that crop is healthy or infected with a certain disease. The other one is weather prediction, this will forecast the weather of that area which will help farmers to give the information about temperature, moisture, rain forecast, etc. and will help to make decisions accordingly.

Term	Description
UGV	Unmanned Ground Vehicle
Database	Collection of all the information monitored by this system
IDE	Integrated Development Environment
AWS	Amazon Web Services

1 Introduction

1.1 Why this Low-Level Design Document?

The purpose of this document is to present a detailed description of the UGV Agriculture Solutions. It will explain the purpose and features, interfaces of the system and its capabilities, the constraints under which it must operate, and how the system will react to external conditions. This document is intended for both the stakeholders and the developers of the system and will be proposed to the higher management for its approval.

The main objective of the project is to detect diseases of a Tomato Crop and notify the farmer about the disease along with Climatic changes using a weather detection approach.
Disease Detection can be achieved using Object Detection Techniques using TensorFlow or Pytorch.

This project shall be delivered in two phases:
Phase 1 : Trained Deep Learning Model with Tomato Crop Data Set and Weather Predictions
Phase 2 : Deployment in UGV using ROS.

1.2 Scope

This system will be designed to detect the diseases of a crop at the earliest for better disease management, Weather Detection which notifies about the coming changes in the climate so that farmers can irrigate their field accordingly.

1.3 Constraints

We will only be selecting a few diseases in the tomato crop as it is difficult to differentiate between the healthy leaf and diseased leaf for few of the diseases. The model accuracy will become low.

1.4 Risks

- Adverse weather conditions can impact UGV's camera and its movement.
- UGV movement may sometimes cause damage to the crops.

1.5 Out of Scope

- Large gap requirements- as UGV cannot be used in the fields where crops are grown very near to each other.
- Random Plantation: UGV requires crops to be grown in a straight line or in an orderly way but if the crops are randomly planted it reduces the reach of UGV.

2 Technical specifications

2.1 Dataset

Disease	Finalized	Source
Yellow Curved	yes	Kaggle
Mosaic	Yes	Kaggle
Late Blight	Yes	Kaggle
Septorial Leaf	Yes	Kaggle

2.1.2 Tomato Dataset Overview

Consists of 4 different folders, each table represents each disease

There are a total of 1600 images of Leaves in the training set and 400 images in the test set.

Bacterial Spot

The symptoms consist of numerous small, angular to irregular, water-soaked spots on the leaves and slightly raised to scabby spots on the fruits. The leaf spots may have a yellow halo. The centre part of the leaf dries out and frequently tear.

Fig. 01: Sample data for Bacterial Spot Disease

Late Blight

Late blight is especially damaging during cool, wet weather. Young leaf lesions are small and appear as dark, water-soaked spots. These leaf spots will quickly enlarge, and a white mold will appear at the margins of the affected area on the lower surface of the leaves. Complete defoliation (browning and shrivelling of leaves and stems) can occur within 14 days from the first symptoms. Fungal spores are spread between plants and gardens by rain and wind.

Fig. 02: Sample data for Late Blight Disease

Septoria Leaf Spot

This destructive disease of tomato foliage, petioles, and stems. Infection usually occurs on the lower leaves near the ground, after plants begin to set fruit. Numerous small, circular spots with dark borders surrounding a beige-coloured centre appear on the older leaves. Tiny black specks, which are spore-producing bodies, can be seen in the center of the spots. Severely spotted leaves turn yellow, die, and fall off the plant.

Fig. 03: Sample data for Septoria Leaf Spot Disease

Yellow Curved

The symptoms include severe stunting, marked reduction in leaf size, upward cupping, chlorosis of leaf margins, mottling, flower abscission, and significant yield reduction. Symptoms in common beans include leaf thickening, leaf crumpling, upward curling of leaves, abnormal lateral shoot proliferation, deformation, and reduction in the number of pods.

Fig. 04: Sample data for Yellow Curved Disease

2.2 Predicting Disease

- Our UGV Robot will move around the specified area of a Tomato field.
- It will capture with the help of a Camera and will identify the infected leaves with the Pre-Trained Model
- If it detects any unhealthy plant, the system will notify the farmer with precautions.

2.3 Deployment

1. AWS

Fig 05: AWS Icon

3 ROS (Robotic Operating System)

For UGV	ROS 2
Deep Learning Model	YOLO v4/v5/Pytorch and SSD/Tensor flow
Language	Python
Dashboard	Power BI
Deployment	AWS

Robot Operating System is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

ROS is a framework on top of the O.S. that allows it to abstract the hardware from the software. This means you can think in terms of software for all the hardware of the robot.

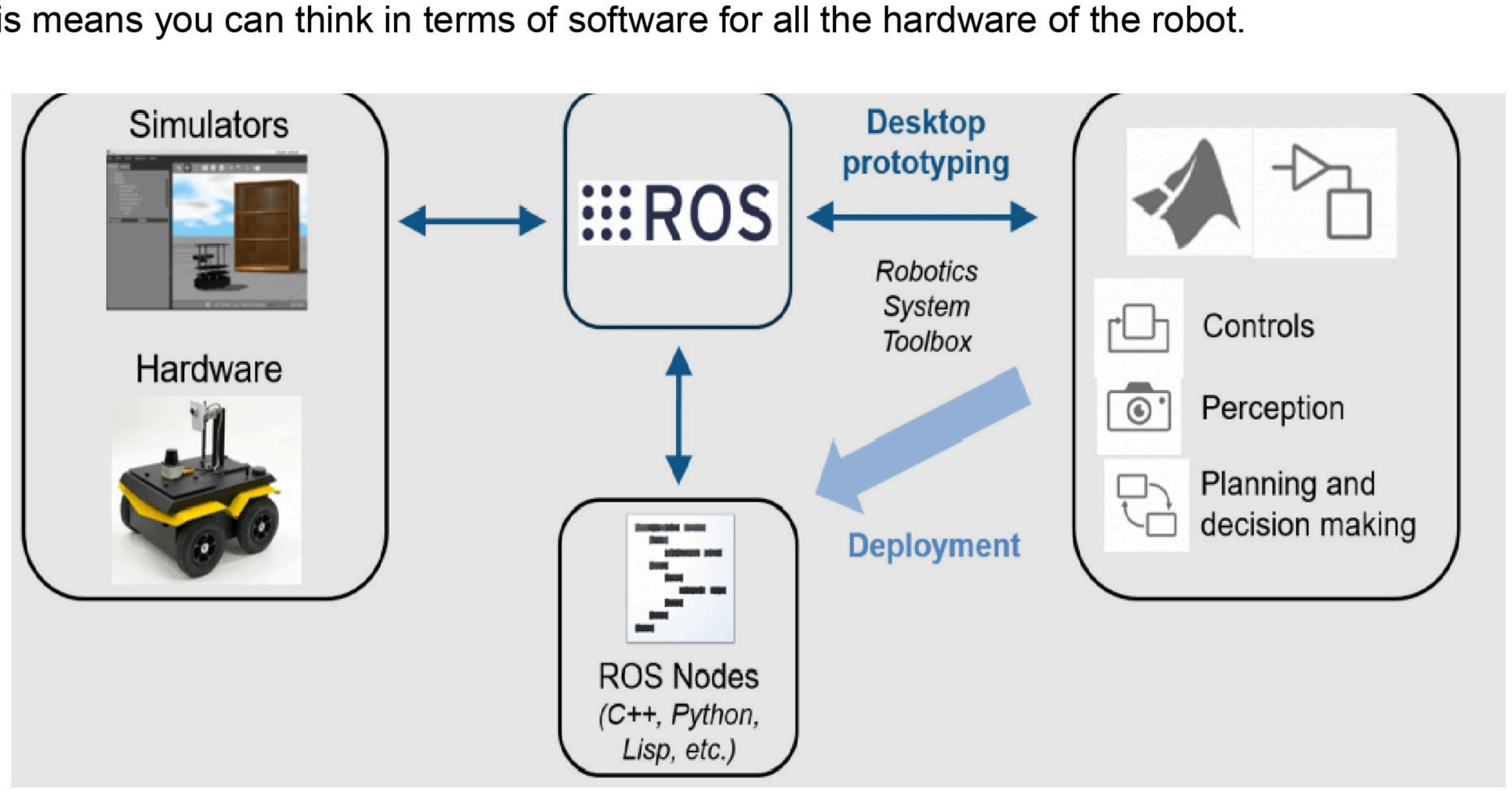

Fig. 06: ROS Workflow

3.1 ROS Navigation

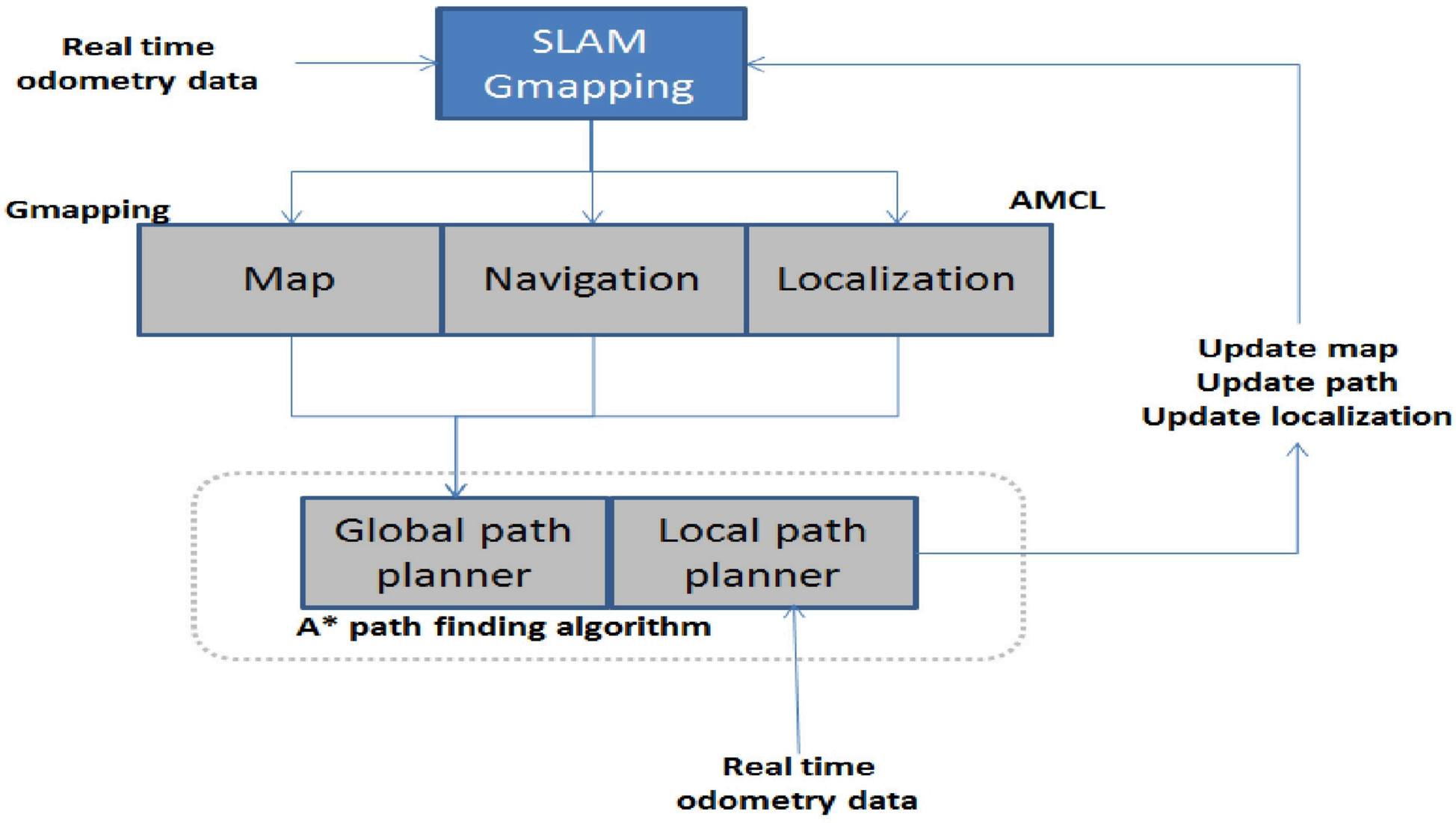

Fig. 07: ROS Navigation

3.2 Gazebo

Gazebo is a 3D simulator, while ROS serves as the interface for the robot. Combining both results in a powerful robot simulator.

With Gazebo you are able to create a 3D scenario on your computer with robots, obstacles, and many other objects. Gazebo also uses a physical engine for illumination, gravity, inertia, etc. You can evaluate and test your robot in difficult or dangerous scenarios without any harm to your robot. Most of the time it is faster to run a simulator instead of starting the whole scenario on your real robot.

3.3 Hardware

- Camera
- UGV

3.4 Robot Specific Features

In addition to the core middleware components, ROS provides common robot-specific libraries and tools that will get your robot up and running quickly.

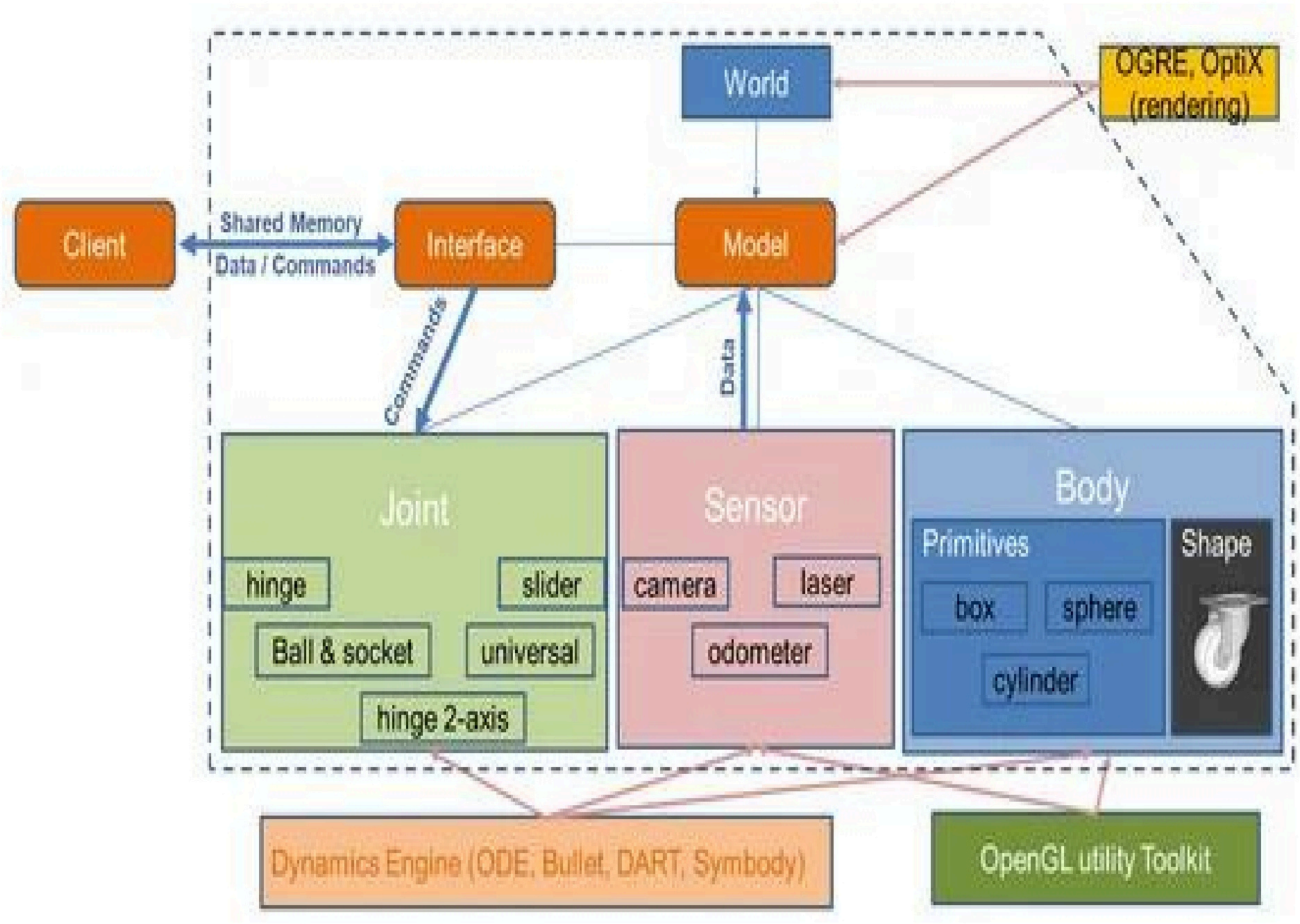

Fig. 08: ROS Components

Here are just a few of the robot-specific capabilities that ROS provides:

- Standard Message Definitions for Robots
- Robot Geometry Library
- Robot Description Language
- Preemptable Remote Procedure Calls
- Diagnostics
- Pose Estimation
- Localization
- Mapping
- Navigation

4 Proposed Solution

Our mission is to build a smart machine that is designed to detect, identify & make critical decisions around each & every plant in an agricultural field with the help of Computer Vision Techniques. Nowadays, technology is growing so rapidly and is implemented in many major areas of interest. One of the most important areas is agriculture, which is given less importance as compared to other engineering activities. Good care of crops leads to better hygiene. It is important to yield good crops which are free from any diseases for human growth.

It is essential to identify any crop diseases before it gets wasted or before it has been accidentally consumed. Crop diseases are sometimes difficult to identify for the naked eye. Thus, it is a very important task to identify crop diseases and AI comes in handy here. AI systems are proved to identify crop diseases with accuracy sometimes greater than human-level accuracy.

Use Case

- **Identify the diseased crop among the field and notify the farmer about the affected disease.**

- **Weather detection API will forecast the climate changes which will help farmers to take the necessary actions to reduce excessive irrigation.**

5 Model training/validation workflow

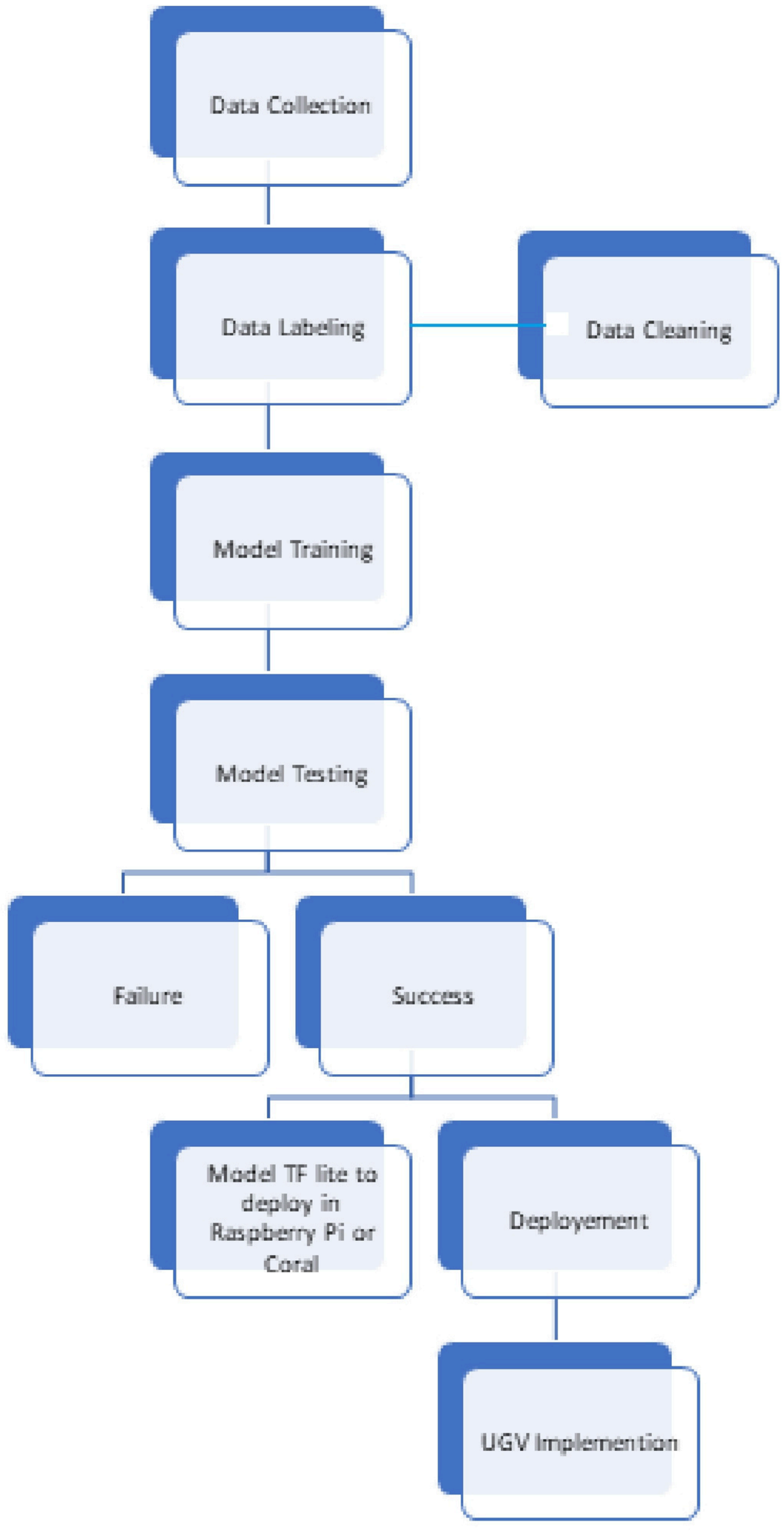

Fig: 9 Model Workflow

6 User I/O workflow

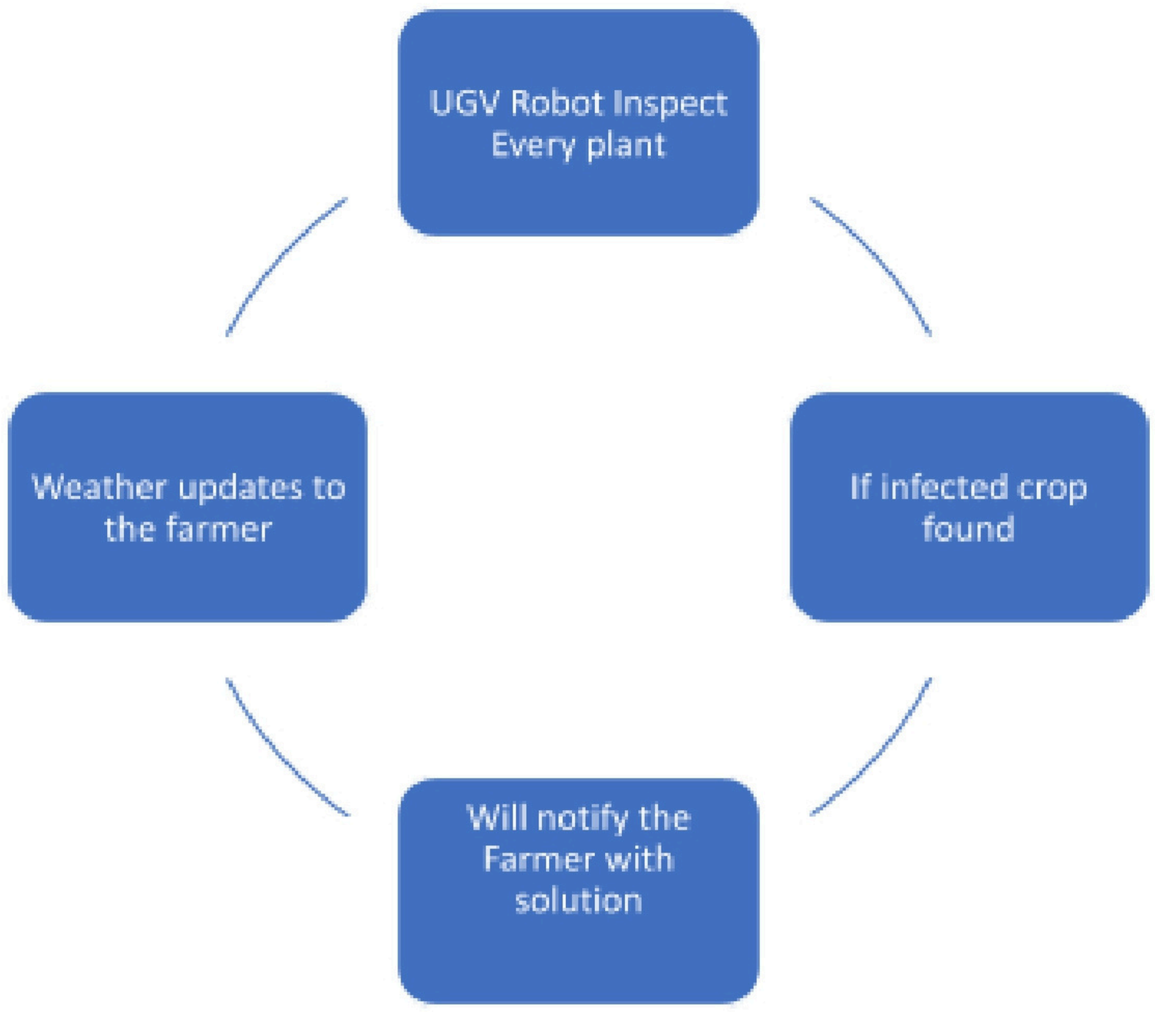

Fig: 10 User I/O Workflow

7 Tensorflow Object Detection

The **TensorFlow object detection API** is the framework for creating a deep learning network that solves object detection problems.

There are already pretrained models in their framework which they refer to as **Model Zoo**. This includes a collection of pretrained models trained on the COCO dataset and the Open Images Dataset. These models can be used for inference if we are interested in categories only in this dataset.

LabelImg is a free, open source tool for graphically labeling images for tensorflow object detection. It's written in Python and uses QT for its graphical interface. It's an easy way to label a huge number of images.

Tensorflow Installation:
https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html

```
In [11]: VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_tf2_test.p
         # Verify Installation
         !python {VERIFICATION_SCRIPT}
```

```
[ RUN      ] ModelBuilderTF2Test.test_session
[  SKIPPED ] ModelBuilderTF2Test.test_session
[ RUN      ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor): 0.0s
I0608 15:12:27.772622 12040 test_util.py:2102] time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor):
0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
[ RUN      ] ModelBuilderTF2Test.test_unknown_meta_architecture
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
I0608 15:12:27.772622 12040 test_util.py:2102] time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_meta_architecture
[ RUN      ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
I0608 15:12:27.773622 12040 test_util.py:2102] time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
----------------------------------------------------------------------
Ran 24 tests in 39.396s

OK (skipped=1)
```

Fig: 11 Tensorflow Installation

Tensorflow Model Zoo:
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

Tensorflow Object Detection:
https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html

TensorFlow 2 Object Detection API tutorial

latest

Search docs

CONTENTS

- Installation
- Training Custom Object Detector
  - Preparing the Workspace
  - Preparing the Dataset
  - Configuring a Training Job
    - Download Pre-Trained Model
    - Configure the Training Pipeline
  - Training the Model
  - Evaluating the Model (Optional)
  - Monitor Training Job Progress using TensorBoard
  - Exporting a Trained Model
- Examples
- Common issues

```
      warmup_steps: 2000
    }
  }
  momentum_optimizer_value: 0.8999999761581421
 }
 use_moving_average: false
}
fine_tune_checkpoint: "pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckp
num_steps: 25000
startup_delay_steps: 0.0
replicas_to_aggregate: 8
max_number_of_boxes: 100
unpad_groundtruth_tensors: false
fine_tune_checkpoint_type: "detection" # Set this to "detection" since we want to be training the
use_bfloat16: false # Set this to false if you are not training on a TPU
fine_tune_checkpoint_version: V2
}
train_input_reader {
 label_map_path: "annotations/label_map.pbtxt" # Path to label map file
 tf_record_input_reader {
  input_path: "annotations/train.record" # Path to training TFRecord file
 }
}
eval_config {
 metrics_set: "coco_detection_metrics"
 use_moving_averages: false
}
eval_input_reader {
 label_map_path: "annotations/label_map.pbtxt" # Path to label map file
 shuffle: false
 num_epochs: 1
 tf_record_input_reader {
  input_path: "annotations/test.record" # Path to testing TFRecord
 }
}
```

Fig: 12 Tensorflow Pipeline.config

7.1 SSD Resnet 101:

SSD, discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes.

Single shot detection – this means that the tasks of object localization and object classification are ready in a single forward pass of the network.

http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet101_v1_fpn_640x640_coco17_tpu-8.tar.gz

```
      agnostic_mode=False)

print('Done')
# DISPLAYS OUTPUT IMAGE
cv2_imshow(image_with_detections)
# CLOSES WINDOW ONCE KEY IS PRESSED
```

```
Loading model...Done! Took 25.621776342391968 seconds
Running inference for /content/training/images/train/b111.jpg... Done
```

Fig: 13 SSD Model Output

Trained the images using the above model using a batch size:8 and the **model accuracy is very low**.

7.2 Faster RCNN:

Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions. The entire system is a single, unified network for object detection.

http://download.tensorflow.org/models/object_detection/tf2/20200711/faster_rcnn_resnet50_v1_640x640_coco17_tpu-8.tar.gz

Trained model **accuracy was 60-70%**

```
# detection_classes should be ints.
detections['detection_classes'] = detections['detectio

label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
            detections['detection_boxes'],
            detections['detection_classes']+label_id_o
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=5,
            min_score_thresh=.8,
            agnostic_mode=False)

plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.
plt.show()
```

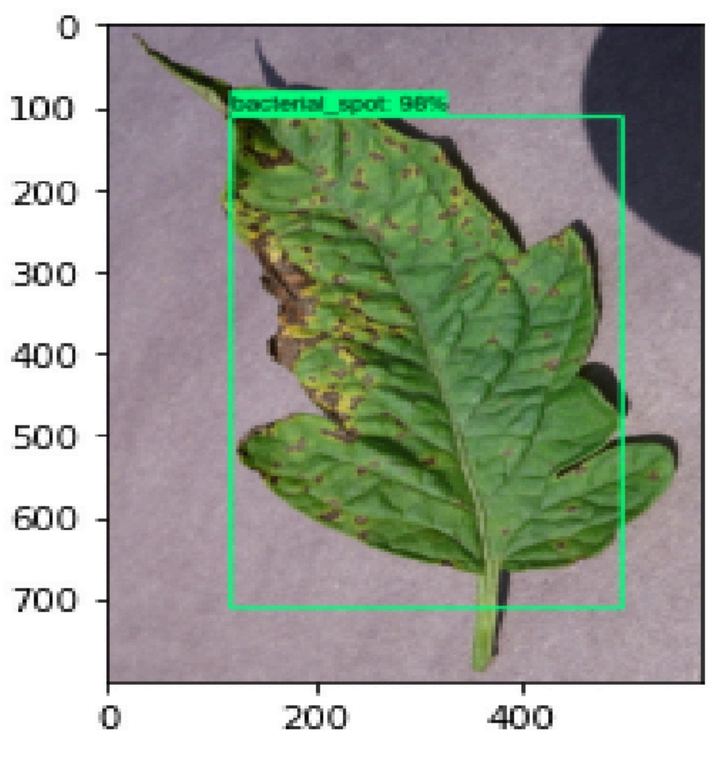

Fig: 14 Faster RCNN Output

8 .YoloV5

Ultralytics' YOLOv5 ("**You Only Look Once**") model family enables real-time object detection with convolutional neural networks.

This algorithm divides images into a grid system. Each cell in the grid is responsible for detecting objects within itself. YOLO is one of the most famous object detection algorithms due to its speed and accuracy.

Weights & Biases is directly integrated into YOLOv5, providing experiment metric tracking, model and dataset versioning, rich model prediction visualization, and more.

Using Roboflow: https://models.roboflow.com/object-detection/yolov5

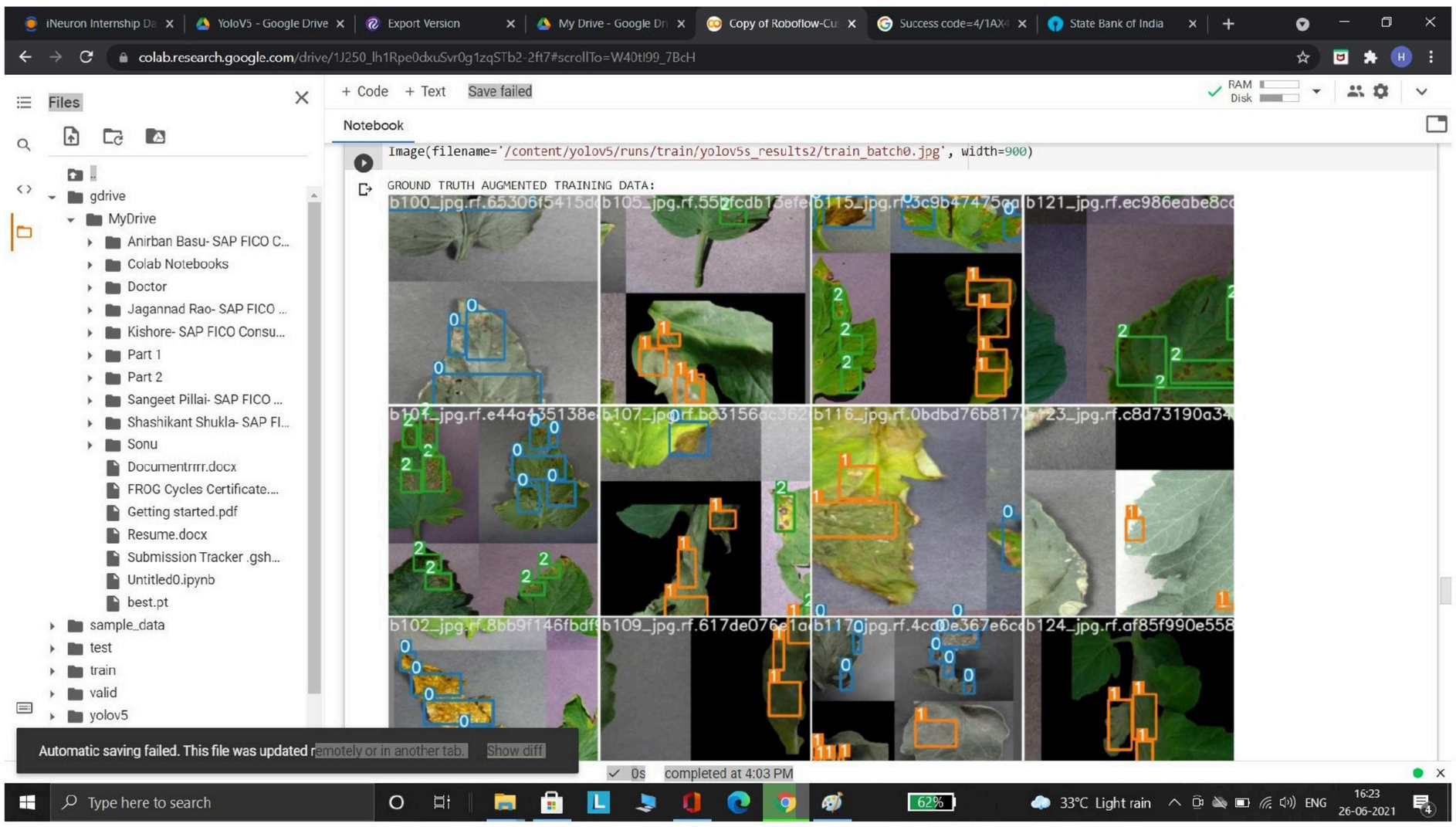

Fig: 15 Yolo V5 Output

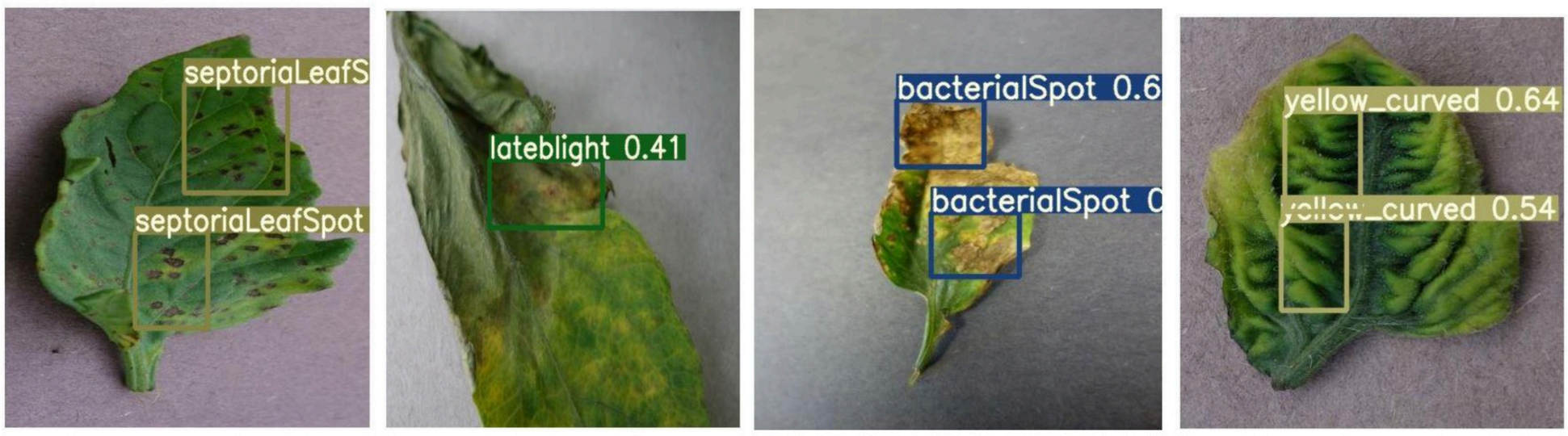

Fig: 16 Yolo V5 Output2

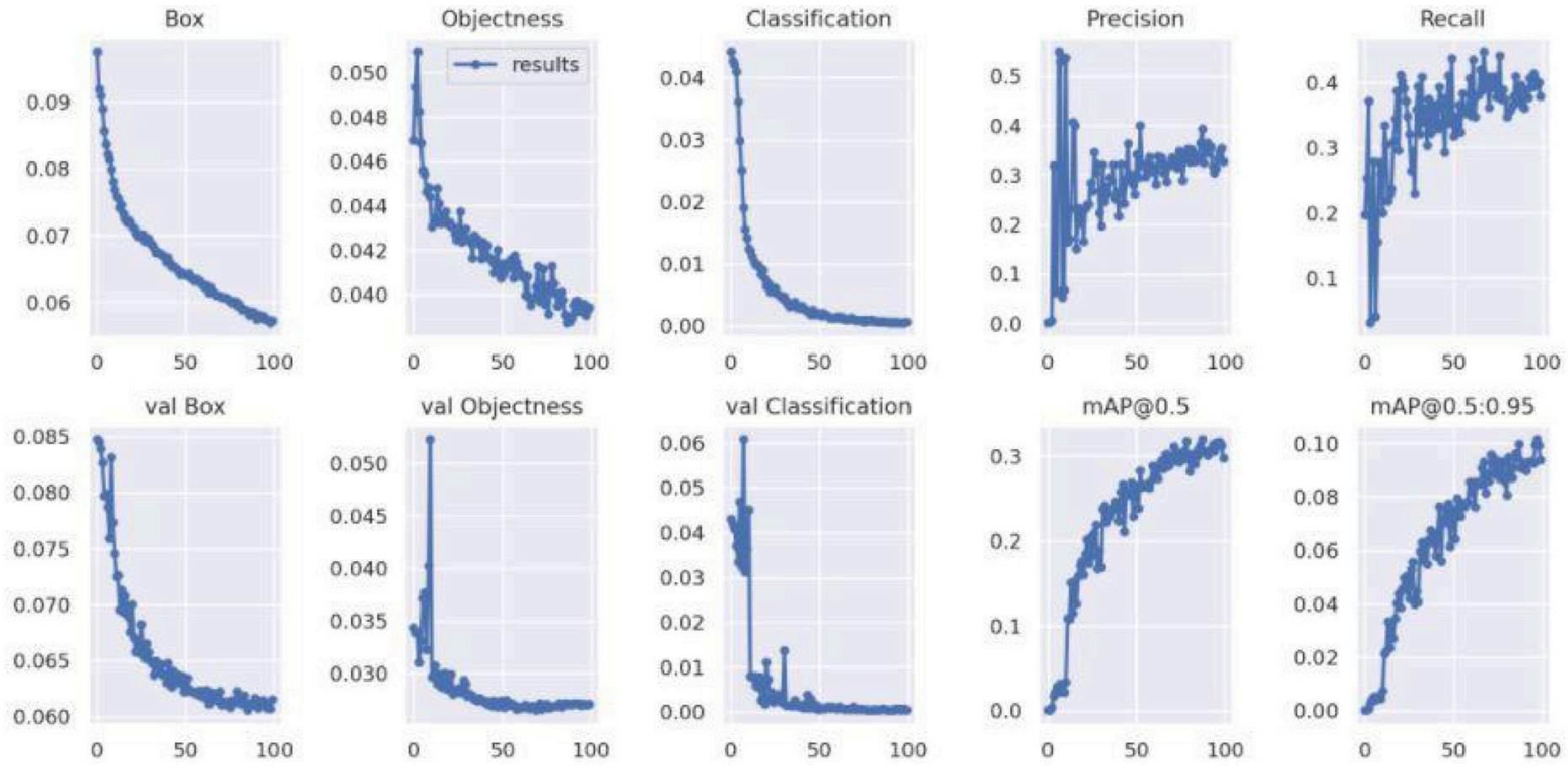

Fig: 17 Yolo V5 Output3

9 Test cases

Test case	Steps to perform test case	Module	Pass/Fail

10 Key performance indicators (KPI)

- Time and workload reduction using the UGV based agriculture solution model.
- Comparison of accuracy of model prediction and farmer's manual prediction.
- Area of the field for respective crops covered by UGV based agriculture solution model.
- The time between symptom onset and detection of disease.
- Timely identification of various crop diseases using UGV based agriculture solution model.
- UGV based agriculture solution model will also forecast the weather prediction/climate changes for helping farmers to take the necessary action accordingly.
- A number of images are captured by UGV every 10 minutes.
- Timely prediction of various crop diseases using UGV based agriculture solution model.
- Probability of crop disease.
- Display of battery life and percentage of UGV.
- Distance travelled by UGV.
- Evaluation of individual disease affecting the crop.