Faculty of Engineering

Helwan University

Computer Engineering Department

Graduation Project 2022

Automated Recognition Fruit Quality System

Supervised by

Prof. Hesham Keshk

Presented by

- **Abd El-Rahman Muftah Yasin**
- **Ahmed Abd El-Mohsen**
- **Alyaa Mohamed Omar**
- **Mohamed Nasser El-Sayed**
- **Omar Khaled Ahmed**

ACKNOWLEDGEMENT

First, we would like to express our deepest gratitude and appreciation to our Supervisor Prof. Hesham Keshk after thanks to god “ALLAH” for his support, patience, guidance and encouragement throughout our graduation project.

We would also like to thank our families, especially our parents, for their encouragement, patience, and assistance over the years.

We are forever indebted to our parents, who have always kept us in their prayers and pushed us to Success.

Finally, for our faculty for providing the suitable environment that led us to represent the best image that computer engineering graduates of Helwan University are supposed to represent.

ABSTRACT

Agriculture has a major role in the economic development of our country. Productive growth and high yield production of fruits is essential and required for the agricultural industry.

The main idea of our project is to automate the process of define the fruit quality using a deep-learning technology by detecting the fruit state from production line flow depending on its maturity grade either it Fresh or Rotten then picking it from that production line and put it in its suitable place without any human intervention in the whole process.

Table of content

INTRODUCTION 8

OBJECT DETECTION 12

The basics 13

Modes and types of object detection 15

Why is object detection important? 17

How does object detection work? 18

Basic structure 19

Model architecture overview 22

How object detection works on the edge? 23

Application of object detection 25

Video surveillance 25

Crowd counting 26

Anomaly detection 26

Self-driving cars 27

YOLO ALGORITHM 28

Introduction to YOLO Algorithm for Object Detection 29

What is YOLO? 30

Why the YOLO algorithm is important? 30

How the YOLO algorithm works? 31

Residual blocks 36

Bounding box regression 37

Intersection over union (IOU) 38

Combination of the three techniques 39

How to encode bounding boxes? 40

Training 43

Testing.....44
Implementation 45
Applications of YOLO.....48
Conclusion.....49
FRUITS DETECTION 50
Target.....51
Detect fruits by color using open-cv.....51
Detect fruits by using YOLO.....54
Steps to build the Model 54
Testing.....60
BANANA RIPENESS DETECTION USING YOLOV5 63
Data Preparation.....65
Gathering Data 65
Labeling Data.....66
Creating necessary files 68
Model Training.....68
Inference.....69
HARDWARE IMPLEMENTATION.....71
Intro 72
The project Workflow 72
Robotic Arm 74
Servo motor.....75
MG996R servo motor Pinout.....75
Main specifications MG996R servo motor.....76
Working principle of servo motor.....76
Robotic arm power.....78

Convey 80
L298N motor driver chip.....80
H-Bridge – to control the rotation direction 81
PWM – to control speed 82
L298N Motor Driver Module Pinout 83
Power Pins 83
Output Pins.....84
Direction Control Pins 85
Speed Control Pins.....86
On-board 5V Regulator and Jumper 87
IR Sensor module 88
Working principle 88
MCU 90
PWM drive:.....92
DEMUX.....93
4052-chip as DEMUX pinout 93
USART driver.....94
USB to TTL converter pinout 94
DIO driver 95
Interrupt driver.....96
Static software design 97
Layered Architecture 97
Layer Modules 97
APP layer flow chart 98
Project hardware picture 99

KINEMATICS OF ROBOTIC ARM 101
Introduction 102
The forward kinematics.....104
The inverse kinematics.....107
SYSTEM GUI 114
GUI in Python language.....115
tkinter — Python interface to Tcl/Tk.....117
Tkinter Modules 120
How Tkinter Wraps Tcl/Tk.....123
System GUI.....126
Design.....126
Description 127
GUI importance 127
APPENDICES.....128
Abbreviations 129
REFERENCES.....130

Chapter 1:

INTRODUCTION

Agriculture has a major benefaction towards economic development by providing food and raw material to non- agricultural sectors.

But the agriculture industry has many problems, including the decreasing number of farm workers and increasing cost of fruit harvesting. Saving labor and scale up in agriculture is necessary in solving these problems.

Moreover, agriculture needs a lot of man work which sometimes cannot be available besides it takes a lot of time for the farmers for manual sorting and examining of fruits from harvest till its growth period. In addition to the manual sorting doesn't give adequate results every time, so it needs an efficient smart farming techniques which can used to get better yield and growth with less human efforts.

In recent years, the automation of agriculture has been advancing for labor saving and large-scale agriculture. However, much of the work in the field of fruit harvesting is manually done. The development of an automated fruit harvesting robot is a viable solution to these problems.

The automatic classification of the fruit quality involves two big tasks: fruit detection and classification of its quality in a production line using computer vision technology using a camera and a robot arm to move to the detected fruit and pick it without damaging the fruit.

The fruit detection and localization using computer vision have been investigated in numerous studies. Where, Color, spectral, or thermal cameras have been widely used in these methods. When using spectral camera, detecting the fruit shadowed by another fruit as an object is difficult. While when a thermal camera is used, the fruit is detected based on the temperature difference between the fruit and the background.

This method is affected by the fruit size and exposure to direct sunlight. Various features are used in fruit detection using color camera.

Here are some the techniques have been used: Used luminance and red, green, and blue (RGB) color difference to segment an apple, Used texture analysis to detect an apple, Integrated multiple features to improve the accuracy of fruit detection methods. Various image classification methods for fruit detection can also be performed using a color camera, Used K-mean clustering for apple detection, Used KNN clustering for apple classification, Used an Artificial Neural Network

for apple classification, Used a Support Vector Machine classification method for apple detection.

However, these methods are difficult to use in variable light conditions because the color information cannot be sufficiently acquired. For better accuracy, fruit detection should be performed using multiple features such as color, shape, texture, and refection to overcome challenges like clustering and variable light conditions

Chapter 2:

OBJECT DETECTION

The basics:

Object detection is a phenomenon in computer vision that involves the detection of various objects in digital images or videos. Some of the objects detected include people, cars, chairs, stones, buildings, Fruits and animals.

Object detection draws bounding boxes around these detected objects, which allow us to locate where said objects are in (or how they move through) a given scene.

This phenomenon seeks to answer two basic questions:

1. What is the object? This question seeks to identify the object in a specific image.
2. Where is it? This question seeks to establish the exact location of the object within the image.

Object detection is commonly confused with image recognition, so before we proceed, it's important that we clarify the distinctions between them.

Image recognition assigns a label to an image. A picture of a dog receives the label "dog". A picture of two dogs, still receives the label "dog". Object detection, on the other hand, draws a box around each dog and labels the box "dog". The model predicts where each object

is and what label should be applied. In that way, object detection provides more information about an image than recognition.

Here’s an example of how this distinction looks in practice:

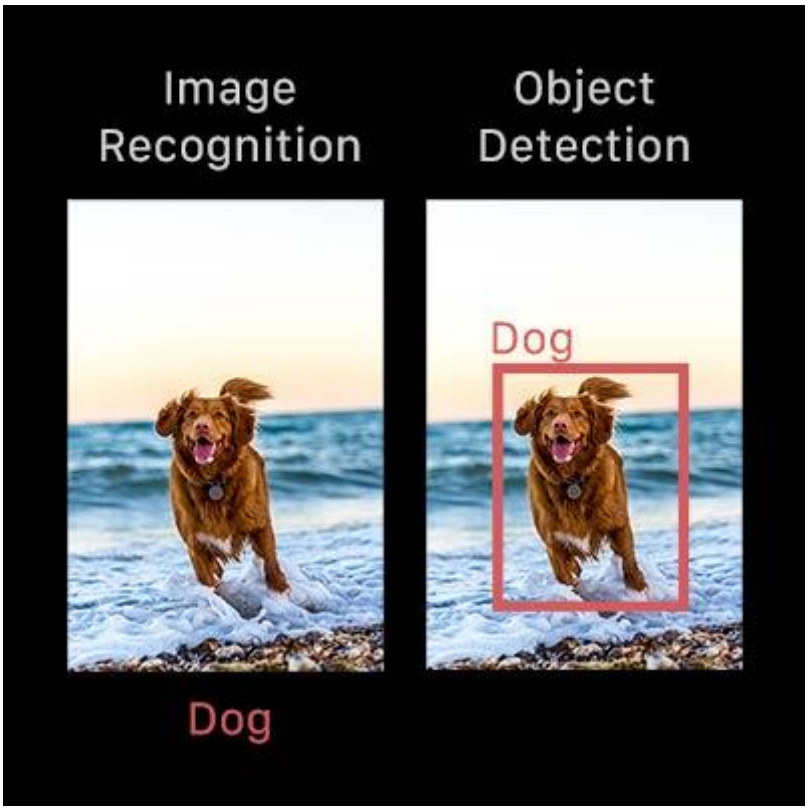


Figure-1

Modes and types of object detection

Broadly speaking, object detection can be broken down into machine learning-based approaches and deep learning-based approaches.

In more traditional ML-based approaches, computer vision techniques are used to look at various features of an image, such as the color histogram or edges, to identify groups of pixels that may belong to an object. These features are then fed into a regression model that predicts the location of the object along with its label.

On the other hand, deep learning-based approaches employ convolutional neural networks (CNNs) to perform end-to-end, unsupervised object detection, in which features don't need to be defined and extracted separately. For a gentle introduction to CNNs, check out this overview.

Object detection consists of various approaches such as fast R-CNN, Retina-Net, and Single-Shot MultiBox Detector (SSD). Although these approaches have solved the challenges of data limitation and modeling in object detection, they are not able to detect objects in a single algorithm run. Yolo Algorithm has gained popularity because of its superior performance over the aforementioned object detection techniques.

Because deep learning methods have become the state-of-the-art approaches to object detection, these are the techniques we'll be focusing on for the purposes of this guide.

Why is object detection important?

Object detection is inextricably linked to other similar computer vision techniques like image recognition and image segmentation, in that it helps us understand and analyze scenes in images or video.

But there are important differences. Image recognition only outputs a class label for an identified object, and image segmentation creates a pixel-level understanding of a scene's elements. What separates object detection from these other tasks is its unique ability to locate objects within an image or video. This then allows us to count and then track those objects.

Given these key distinctions and object detection's unique capabilities, we can see how it can be applied in a number of ways:

- Crowd counting
- Self-driving cars
- Video surveillance
- Face detection
- Anomaly detection

Of course, this isn't an exhaustive list, but it includes some of the primary ways in which object detection is shaping our future.

How does object detection work?

Object detection is inextricably Now that we know a bit about what object detection is, the distinctions between different types of object detection, and what it can be used for, let's explore in more depth how it works.

In this section, we'll look at several deep learning-based approaches to object detection and assess their advantages and limitations. Just as a reminder—for the purposes of this overview, we're going to look at the approaches that use neural networks, which have become the state-of-the-art methods for object detection. In this section, we'll look at several deep learning-based approaches to object detection and assess their advantages and limitations. Just as a reminder—for the purposes of this overview, we're going to look at the approaches that use neural networks, which have become the state-of-the-art methods for object detection.

Basic structure

Deep learning-based object detection models typically have two parts. An encoder takes an image as input and runs it through a series of blocks and layers that learn to extract statistical features used to locate and label objects. Outputs from the encoder are then passed to a decoder, which predicts bounding boxes and labels for each object.

The simplest decoder is a pure regressor. The regressor is connected to the output of the encoder and predicts the location and size of each bounding box directly. The output of the model is the X, Y coordinate pair for the object and its extent in the image. Though simple, this type of model is limited. You need to specify the number of boxes ahead of time. If your image has two dogs, but your model was only designed to detect a single object, one will go unlabeled. However, if you know the number of objects you need to predict in each image ahead of time, pure regressor-based models may be a good option.

An extension of the regressor approach is a region proposal network. In this decoder, the model proposes regions of an image where it believes an object might reside. The pixels belonging to these regions are then fed into a classification subnetwork to determine a label (or reject the proposal). It then runs the pixels containing those regions through a classification network. The benefit of this method is a more accurate, flexible model that can propose arbitrary numbers of

regions that may contain a bounding box. The added accuracy, though, comes at the cost of computational efficiency.

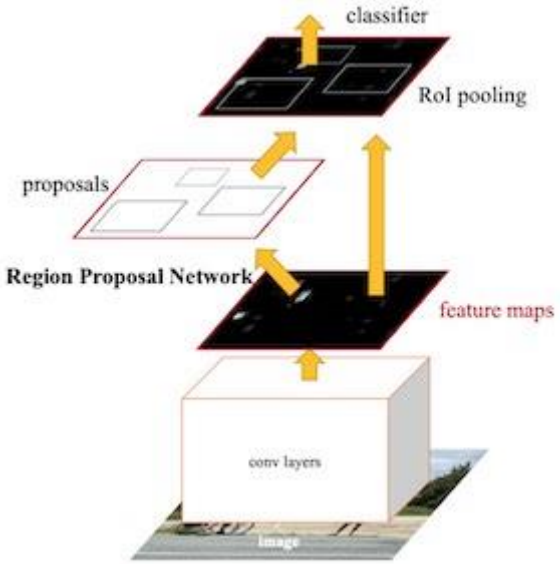


Figure-2

Single shot detectors (SSDs) seek a middle ground. Rather than using a subnetwork to propose regions, SSDs rely on a set of predetermined regions. A grid of anchor points is laid over the input image, and at each anchor point, boxes of multiple shapes and sizes serve as regions. For each box at each anchor point, the model outputs a prediction of whether or not an object exists within the region and modifications to the box's location and size to make it fit the object more closely. Because there are multiple boxes at each anchor point and anchor points may be close together, SSDs produce many potential detections that overlap. Post-processing must be applied to SSD outputs in order to prune away most of these predictions and pick the best one. The

most popular post-processing technique is known as non-maximum suppression.

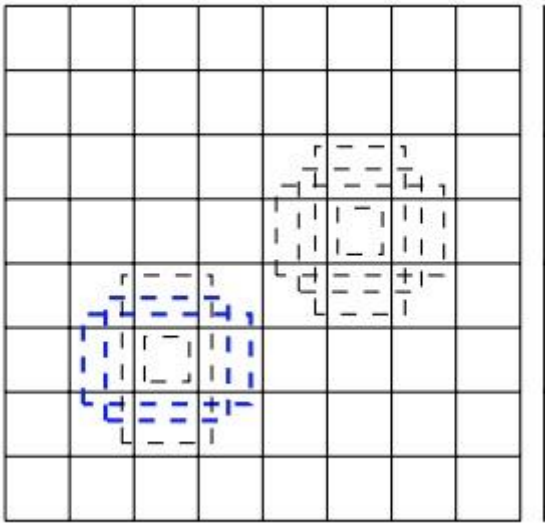


Figure-3

Finally, a note on accuracy. Object detectors output the location and label for each object, but how do we know how well the model is doing? For an object's location, the most commonly-used metric is intersection-over-union (IOU). Given two bounding boxes, we compute the area of the intersection and divide by the area of the union. This value ranges from 0 (no interaction) to 1 (perfectly overlapping). For labels, a simple "percent correct" can be used.

Model architecture overview

R-CNN, Faster R-CNN, Mask R-CNN:

A number of popular object detection models belong to the R-CNN family. Short for region convolutional neural network, these architectures are based on the region proposal structure discussed above. Over the years, they've become both more accurate and more computationally efficient. Mask R-CNN is the latest iteration, developed by researchers at Facebook, and it makes a good starting point for server-side object detection models.

YOLO, MobileNet + SSD, SqueezeDet:

There are also a number of models that belong to the single shot detector family. The main difference between these variants are their encoders and the specific configuration of predetermined anchors. MobileNet + SSD models feature a MobileNet-based encoder, SqueezeDet borrows the SqueezeNet encoder, and the YOLO model features its own convolutional architecture. SSDs make great choices for models destined for mobile or embedded devices.

CenterNet:

More recently, researchers have developed object detection models that do away with the need for region proposals entirely. CenterNet treats objects as single points, predicting the X, Y coordinates of an object's center and its extent (height and width). This technique has

proven both more efficient and accurate than SSD or R-CNN approaches

How object detection works on the edge?

If your use case requires that object detection work in real-time, without internet connectivity, or on private data, you might be considering running your object detection model directly on an edge device like a mobile phone or IoT board.

In those cases, you'll need to choose specific model architectures to make sure everything runs smoothly on these lower power devices. Here are a few tips and tricks to ensure your models are ready for edge deployment:

- Prune your network to include fewer convolution blocks. Most papers use network architectures that are not constrained by compute or memory resources. This leads to networks with far more layers and parameters than are required to generate acceptable predictions.
- Add a width multiplier to your model so you can adjust the number of parameters in your network to meet your computation and memory constraints. The number of filters in a convolution layer, for example, greatly impacts the overall size of your model. Many papers and open-source implementations will treat this number as a fixed constant, but most of these models were never intended for mobile use. Adding a parameter

that multiplies the base number of filters by a constant fraction allows you to modulate the model architecture to fit the constraints of your device. For some tasks, you can create much, much smaller networks that perform just as well as large ones.

- Shrink models with quantization, but beware of accuracy drops. Quantizing model weights can save a bunch of space, often reducing the size of a model by a factor of 4 or more. However, accuracy will suffer. Make sure you test quantized models rigorously to determine if they meet your needs.
- Input and output sizes can be smaller than you think! If you're designing a photo organization app, it's tempting to think that your object detection model needs to be able to accept full resolution photos as an input. In most cases, edge devices won't have nearly enough processing power to handle this. Instead, it's common to train object detection models at modest resolutions, then downscale input images at runtime.

Application of object detection

We'll examine how object detection can be used in the following areas:

- Video surveillance
- Crowd counting
- Anomaly detection (i.e. in industries like agriculture, health care)
- Self-driving cars

Video surveillance:

Because state-of-the-art object detection techniques can accurately identify and track multiple instances of a given object in a scene, these techniques naturally lend themselves to automating video surveillance systems.

For instance, object detection models are capable of tracking multiple people at once, in real-time, as they move through a given scene or across video frames. From retail stores to industrial factory floors, this kind of granular tracking could provide invaluable insights into security, worker performance and safety, retail foot traffic, and more.

Crowd counting:

Crowd counting is another valuable application of object detection. For densely populated areas like theme parks, malls, and city squares, object detection can help businesses and municipalities more effectively measure different kinds of traffic—whether on foot, in vehicles, or otherwise.

This ability to localize and track people as they maneuver through various spaces could help businesses optimize anything from logistics pipelines and inventory management, to store hours, to shift scheduling, and more. Similarly, object detection could help cities plan events, dedicate municipal resources, etc.

Anomaly detection:

Anomaly detection is a use case of object detection that's best explained through specific industry examples.

In agriculture, for instance, a custom object detection model could accurately identify and locate potential instances of plant disease, allowing farmers to detect threats to their crop yields that would otherwise not be discernible to the naked human eye.

And in health care, object detection could be used to help treat conditions that have specific and unique symptomatic lesions. One such example of this comes in the form of skin care and the treatment

of acne—an object detection model could locate and identify instances of acne in seconds.

What's particularly important and compelling about these potential use cases is how they leverage and provide knowledge and information that's generally only available to agricultural experts or doctors, respectively.

Self-driving cars:

Real-time car detection models are key to the success of autonomous vehicle systems. These systems need to be able to identify, locate, and track objects around them in order to move through the world safely and efficiently.

And while tasks like image segmentation can be (and often are) applied to autonomous vehicles, object detection remains a foundational task that underpins current work on making self-driving cars a reality.

Chapter 3:

YOLO ALGORITHM

Introduction to YOLO Algorithm for Object Detection

The YOLO framework (You Only Look Once) deals with object detection in a different way. It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes. The biggest advantage of using YOLO is its simple and not complicated – it's incredibly fast and can process 45 frames per second. YOLO also understands generalizes object representation. This is one of the best algorithms for object detection and has shown a comparatively similar performance to the R-CNN algorithm.

YOLO is an algorithm that uses neural networks to provide real-time object detection. This algorithm is popular because of its speed and accuracy. It has been used in various applications to detect traffic signals, people, parking meters, and animals.

What is YOLO?

YOLO is an abbreviation for the term 'You Only Look Once'. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

The YOLO algorithm consists of various variants. Some of the common ones include tiny YOLO and YOLOv5.

Why the YOLO algorithm is important?

YOLO algorithm is important because of the following reasons:

- **Speed:** This algorithm improves the speed of detection because it can predict objects in real-time.
- **High accuracy:** YOLO is a predictive technique that provides accurate results with minimal background errors.

- **Learning capabilities:** The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

How the YOLO algorithm works?

YOLO algorithm works using the following three techniques:

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)

  - YOLO first takes an input image:

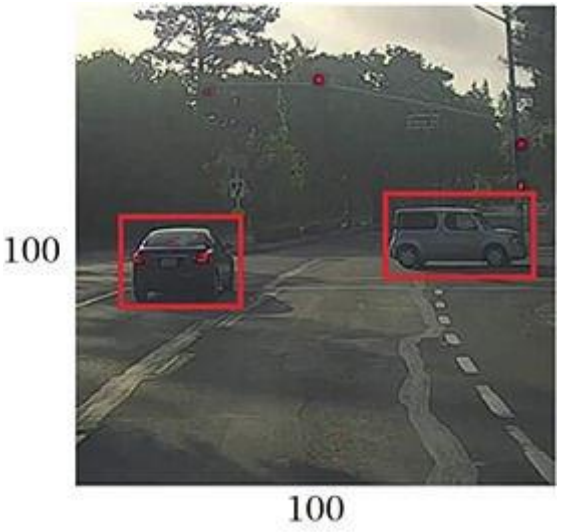


Figure-4

  - The framework then divides the input image into grids(say a 3 x 3 grid):

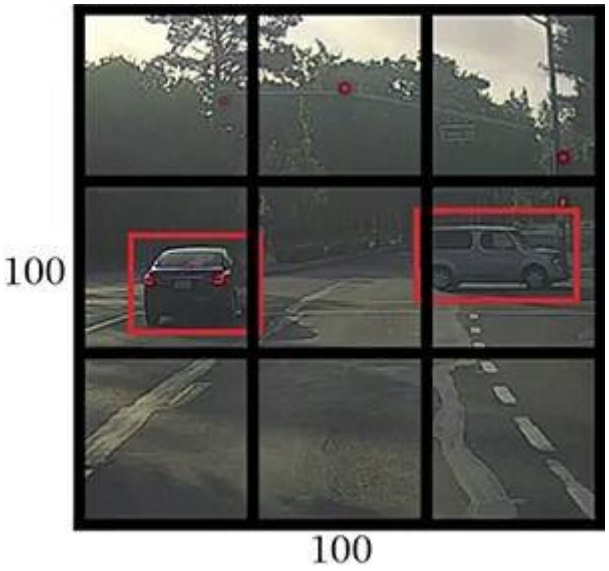


Figure-5

Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for object (if any are found). Suppose we have divided the image into a grid of size 3 x 3 and there are a total of 3 classes which we want the objects to be classified into. Let's say the classes are pedestrian, Car and Motorcycle respectively. So, for each grid cell, the label y will be eight-dimensional vector.

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

Table-1

Here,

- PC defines whether an object is present in the grid or not (it is the probability)
- b_x, b_y, b_h, b_w Specify the bounding box if there is an object
- c_1, c_2, c_3 represent the classes. So, if the object is a car, c_2 will be 1 and $c_1 \& c_3$ will be 0 and so on.

Let's say we select the first grid from the above example:

Figure-6

Since there is no object in this grid, PC will be zero and the y label for this grid will be:

y =	0
	?
	?
	?
	?
	?
	?
	?

Table-2

The $b_x, b_y, b_h, b_w, c_1, c_2\ and\ c_3$ contain as there is no object in the grid, let's take another grid in which we have a car ($c_2 = 1$):

Figure-7

YOLO decides whether there actually is an object in the grid. In the image, there are two objects (two cars), so YOLO will take the mid-point of these two objects and these objects will be assigned to the grid which contains the mid-point of these objects. The y label for the Centre left grid with the car will be:

y =	1
	bx
	by
	bh
	bw
	0
	1
	0

Table-3

Since there is an object in the grid, PC will be equal to 1 b_x, b_y, b_h, b_w will be calculated relative to the particular grid cell we are dealing with. Since car is the second class $c_2 = 1$ and $c_1$and $c_3 = 0$. So, for each of the 9 grids, we will have eight-dimensional output vectors. This output will have a shape 3 x 3 x 8. So now we have an input image and its corresponding target vector. Using the above example (input image – 100 x 100 x 3, output – 3 x 3 x 8), our model will be trained as follows:

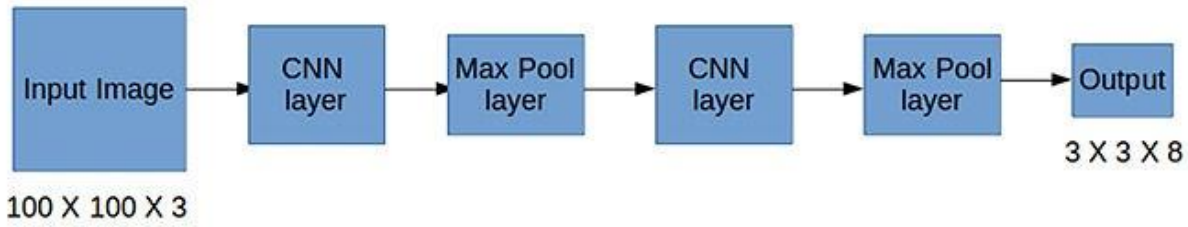


Figure-8

Residual blocks

First, the image is divided into various grids. Each grid has a dimension of S x S. The following image shows how an input image is divided into grids.

Figure-9

In the image above, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them. For example, if an object center appears within a certain grid cell, then this cell will be responsible for detecting it.

Bounding box regression

A bounding box is an outline that highlights an object in an image.

Every bounding box in the image consists of the following attributes:

- Width (b_w)
- Height (b_h)
- Class (for example, person, car, traffic light, etc.)- This is represented by the letter c.
- Bounding box center (b_x, b_y)

The following image shows an example of a bounding box. The bounding box has been represented by a yellow outline.

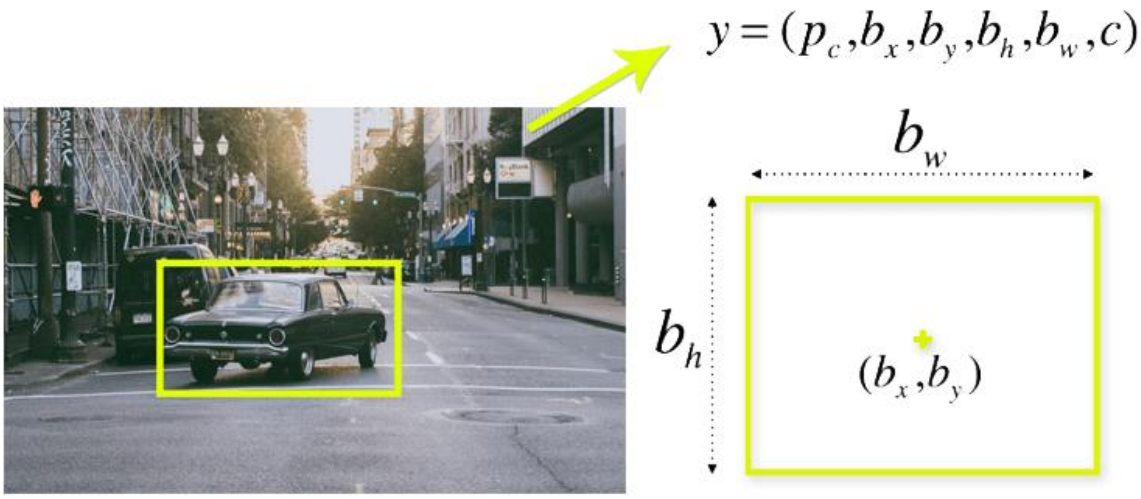


Figure-10

YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. In the image above, represents the probability of an object appearing in the bounding box.

Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.

The following image provides a simple example of how IOU works.

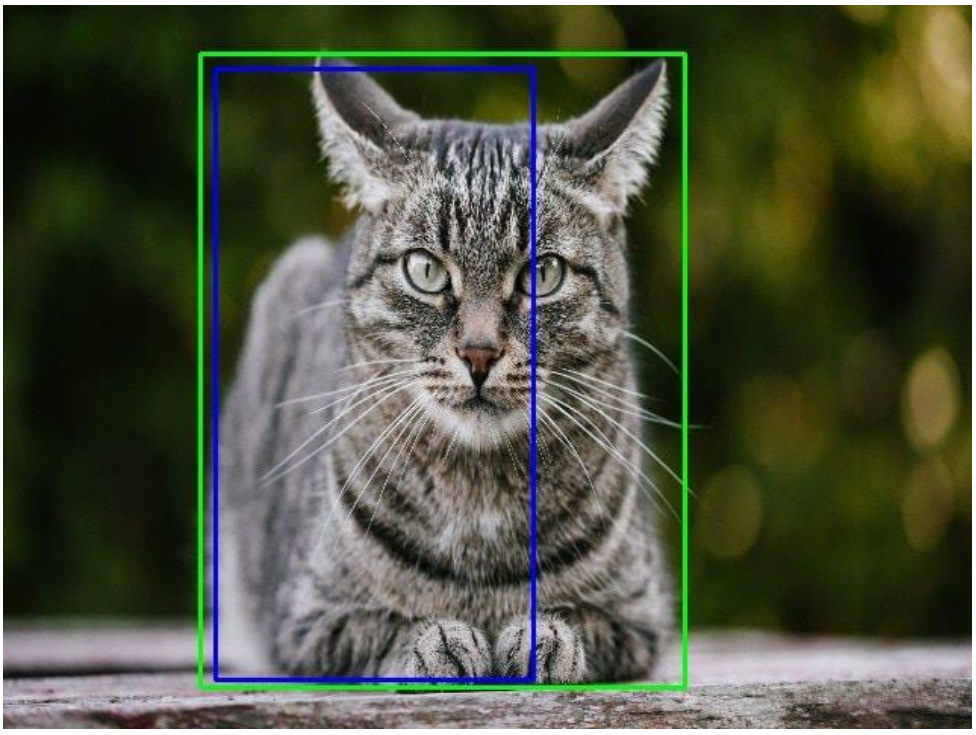


Figure-11

In the image above, there are two bounding boxes, one in green and the other one in blue. The blue box is the predicted box while the green box is the real box. YOLO ensures that the two bounding boxes are equal.

Combination of the three techniques

The following image shows how the three techniques are applied to produce the final detection results.

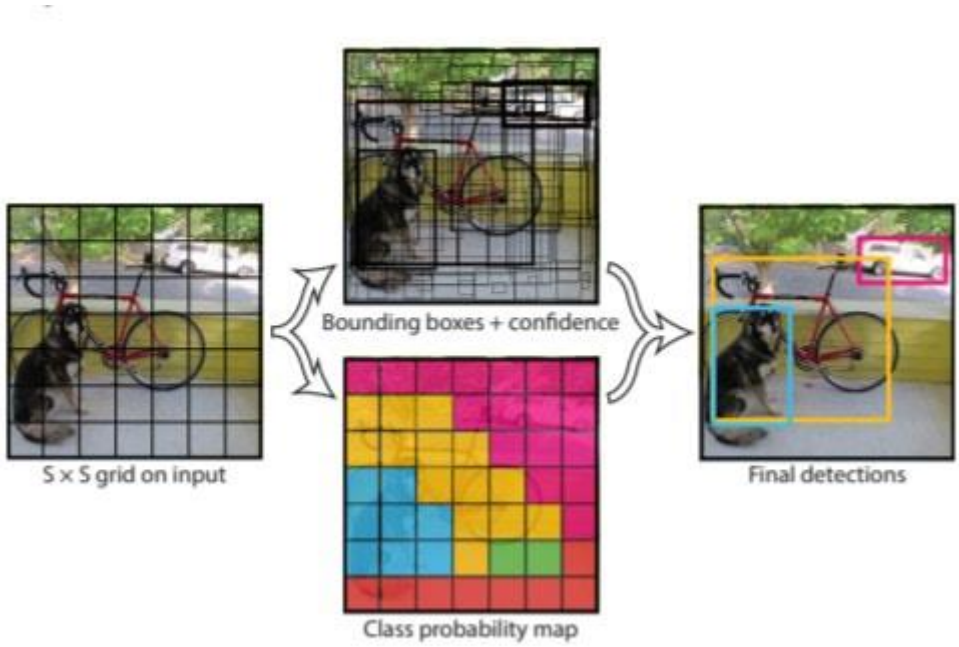


Figure-12

First, the image is divided into grid cells. Each grid cell forecasts B bounding boxes and provides their confidence scores. The cells predict the class probabilities to establish the class of each object.

For example, we can notice at least three classes of objects: a car, a dog, and a bicycle. All the predictions are made simultaneously using a single convolutional neural network.

Intersection over union ensures that the predicted bounding boxes are equal to the real boxes of the objects. This phenomenon eliminates unnecessary bounding boxes that do not meet the characteristics of the objects (like height and width). The final detection will consist of unique bounding boxes that fit the objects perfectly.

For example, the car is surrounded by the pink bounding box while the bicycle is surrounded by the yellow bounding box. The dog has been highlighted using the blue bounding box.

How to encode bounding boxes?

The b_x, b_y, b_h and b_w are calculated relative to the grid cell we are dealing with. Consider the center-right grid which contains a car:

Figure-13

So, b_x, b_y, b_h andb_w will be calculated relative to this grid only. The y label for this grid will be:

y =	1
	bx
	by
	bh
	bw
	0
	1
	0

Table-4

PC = 1 since there is an object in this grid and since it is a car, $c_2 = 1$. in YOLO, the coordinates assigned to all the grids (b_x, b_y, b_h and b_w) are:

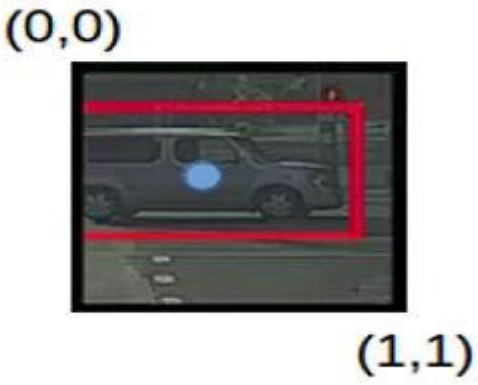


Figure-14

The b_x, b_y are the x and y coordinates of the midpoint of the object with respect to this grid. In this case,

it will be (around) $b_x = 0.4, b_y = 0.3$:

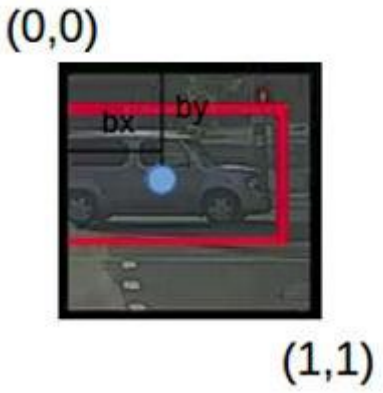


Figure-15

Now, b_h is the ratio of the height of the bounding box (red box in the above example) to the height of the corresponding grid cell, which in our case is around 0.9 So, $b_h = 0.9$.

b_w is the ratio of the width of the bounding box to the width of the grid cell, So $b_w = 0.5$ (approximately). The y label for this grid will be:

y =	1
	0.4
	0.3
	0.9
	0.5
	0
	1
	0

Table-5

The b_x and b_y will always range between 0 and 1 as the midpoint will always lie within the grid. Whereas b_h and b_w can be more than 1 in

case the dimension of the bounding box is more than the dimension of the grid.

Training:

The input for training our model will obviously be images and their corresponding y labels. Let's see an image and make its y label:

Figure-16

Consider the scenario where we are using a 3 x 3 grid with two anchors per grid, and there are 3 different object classes.

So, the corresponding y labels will have a shape of 3 x 3 x 16. Now, suppose if we use 5 anchor boxes per grid and the number of classes has been increased to 5.

So the target will be 3 x 3 x 10 x 5 = 3 x 3 x 50. This is how the training process is done- taking an image of a particular shape and mapping it with a 3 x 3 x 16 target (this may change as per the grid size, number of anchor boxes and the number of classes).

Testing

- The new image will be divided into the same number of grids which we have chosen during the training period. For each grid, the model will predict an output of shape 3 x 3 x 16 (assuming this is the shape of the target during training time).
- The 16 values in this prediction will be in the same format as that of the training label.
- The first 8 values will correspond to anchor box 1, where the first value will be the probability of an object in that grid.
- Values 2-5 will be the bounding box coordinates for that object, and the last three values will tell us which class the object belongs to.
- The next 8 values will be for anchor box 2 and in the same format, i.e., first the probability, then the bounding box coordinates, and finally the classes.
- Finally, the Non-Max Suppression technique will be applied on the predicted boxes to obtain a single prediction per object.
- Exact dimensions and steps that the YOLO algorithm follows:
    1- Takes an input image of shape (608, 608, 3).
    2- Passes this image to convolutional neural network (CNN), which returns a (19 ,19, 5, 85) dimensional output.
    3- The last two dimensions of the above output are flattened to get output volume pf (19, 19, 425):

Here, each cell of 19 x 19 grid returns 425 numbers.

425 = 5*85, where 5 is the number of anchor boxes per grid.

85 = 5+85, where 5 is (PC, b_x, b_y, b_h, b_w) and 80 is the number of classes we want to detect.

4- Finally, do the IOU and Non- Max Suppression to avoid selecting overlapping boxes.

Implementation:

Step-1: Import the required libraries.

Step-2: Create a function for filtering the boxes based on their probabilities and threshold:

```
def yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = .6):
    box_scores = box_confidence*box_class_probs
    box_classes = K.argmax(box_scores,-1)
    box_class_scores = K.max(box_scores,-1)
    filtering_mask = box_class_scores>threshold
    scores = tf.boolean_mask(box_class_scores,filtering_mask)
    boxes = tf.boolean_mask(boxes,filtering_mask)
    classes = tf.boolean_mask(box_classes,filtering_mask)
    return scores, boxes, classes
```

Step-3: Define a function to calculate the IOU between two boxes:

```
def iou(box1, box2):
    xi1 = max(box1[0],box2[0])
    yi1 = max(box1[1],box2[1])
    xi2 = min(box1[2],box2[2])
    yi2 = min(box1[3],box2[3])
    inter_area = (yi2-yi1)*(xi2-xi1)
    box1_area = (box1[3]-box1[1])*(box1[2]-box1[0])
    box2_area = (box2[3]-box2[1])*(box2[2]-box2[0])
    union_area = box1_area+box2_area-inter_area
    iou = inter_area/union_area
    return iou
```

Step-4: Define a function for Non-Max Suppression.

Step-5: Create a random volume of shape (19,19,5,85) and then predict the bounding boxes.

Step-6: Finally, we will define a function which will take the outputs of a CNN as input and return the suppressed boxes:

Step-7: Use the yolo_eval function to make predictions for a random volume:

```
scores, boxes, classes = yolo_eval(yolo_outputs)
```

Step-8: Use a pretrained YOLO algorithm on new images and see how it works:

```
sess = K.get_session()
class_names = read_classes("model_data/coco_classes.txt")
anchors = read_anchors("model_data/yolo_anchors.txt")
yolo_model = load_model("model_data/yolo.h5")
```

Step-9: Define a function to predict the bounding boxes and save the images with these bounding boxes included.

Step-10: Read an image and make predictions using the predict function:

```
img = plt.imread('images/img.jpg')
image_shape = float(img.shape[0]), float(img.shape[1])
scores, boxes, classes = yolo_eval(yolo_outputs, image_shape)
```

Step-11: Plot the predictions:

```
out_scores, out_boxes, out_classes = predict(sess, "img.jpg")
```

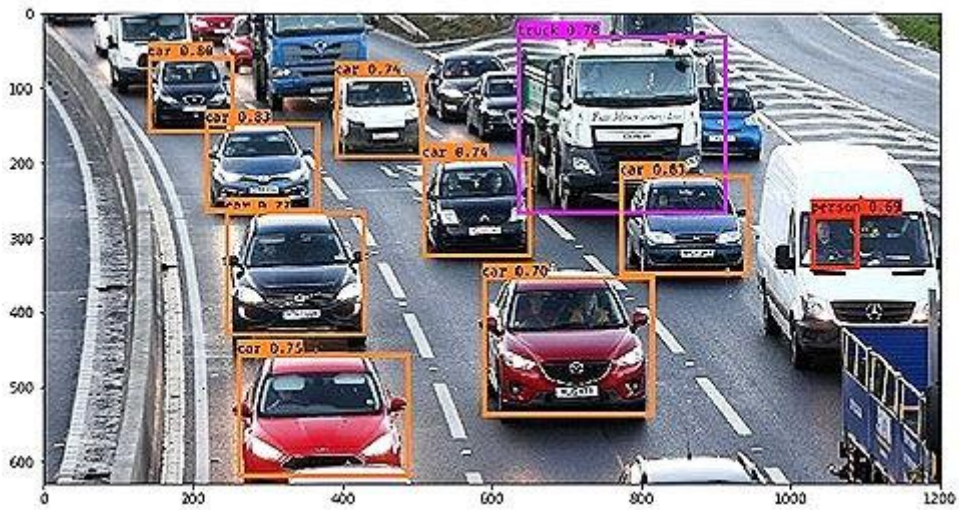


Figure-17

Applications of YOLO

YOLO algorithm can be applied in the following fields:

- **Autonomous driving:** YOLO algorithm can be used in autonomous cars to detect objects around cars such as vehicles, people, and parking signals. Object detection in autonomous cars is done to avoid collision since no human driver is controlling the car.
- **Wildlife:** This algorithm is used to detect various types of animals in forests. This type of detection is used by wildlife rangers and journalists to identify animals in videos (both recorded and real-time) and images. Some of the animals that can be detected include giraffes, elephants, and bears.
- **Security:** YOLO can also be used in security systems to enforce security in an area. Let's assume that people have been restricted from passing through a certain area for security reasons. If someone passes through the restricted area, the YOLO algorithm will detect him/her, which will require the security personnel to take further action.

Conclusion

This article has provided an overview of the YOLO algorithm and how it is used in object detection. This technique provides improved detection results compared to other object detection techniques such as Fast R-CNN and Retina-Net.

To summarize:

- YOLO is a state-of-the-art object detection algorithm that is incredibly fast and accurate
- We send an input image to a CNN which outputs a 19 X 19 X 5 X 85 dimensions volume.
- Here, the grid size is 19 X 19 and each grid contains 5 boxes
- We filter through all the boxes using Non-Max Suppression, keep only the accurate boxes, and also eliminate overlapping boxes
- We have gained an overview of object detection and the YOLO algorithm.
- We have gone through the main reasons why the YOLO algorithm is important.
- We have learned how the YOLO algorithm works. We have also gained an understanding of the main techniques used by YOLO to detect objects.

Chapter 4:

FRUITS DETECTION

Target

Identify the fruits in the Region and determine their type and location

To achieve Target Can Using Two Method for Fruits Detection

1- Detect Fruits by color
2- Using object detection model

Ex: Yolo, SSD, Fast-RCNN, CNN

Detect fruits by color using open-cv

Detect objects using colors in the **HSV color space** using OpenCV-Python. You need to specify a range of color values by means of which the object you are interested in will be identified and extracted.

1- Read the input and background image. Convert the input image from BGR into the HSV color space

Figure-18

2- Create a mask for the color of object by selecting a possible range of HSV object color that the colors can have:

Determine HSV (Hue Saturation Value), every color has 6 values determine range (l-h, l-s, l-v, h-h, h-s, h-v)

Ex: red → [170,100,80,180,255,255]

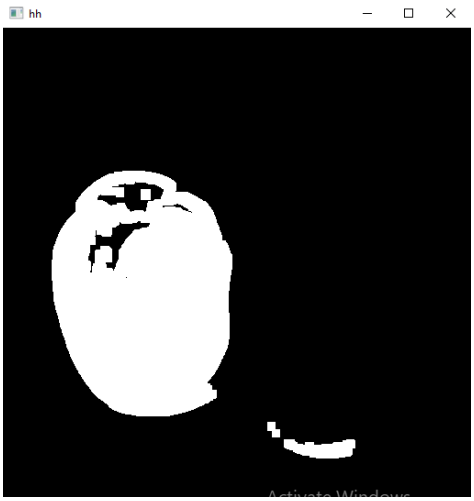


Figure-19

3- Slice the red using the mask:

Mask the color in interest range to get the interest color we need only

4- Finally, create the transparent object image by first extracting the background without the input image with the object, and then extracting the area corresponding to the foreground object from the background image and adding these two images

Figure-20

5- Draw rectangle of ROI and save x_center, y_center:

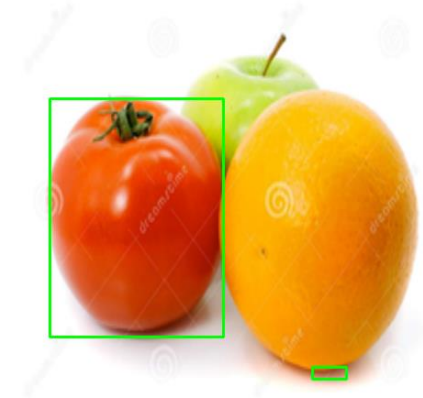


Figure-21

Detect fruits by using YOLO

We used Yolov5 Model because it is:

YOLO performs classification and bounding box regression in one step, making it much faster than most convolutional neural networks. For example, YOLO object detection is more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

Steps to build the Model

1- Preparing the dataset for training:

- Web scraping to calculate fruits data.

- Read the training data in train. CVS file

Each line represents the data of a bounding box. Due to the high density of fruit, each image may contain multiple fruit to be detected, so there may be multiple bounding boxes in an image:

Each column of data respectively corresponds to the unique image id, width, height of image, data of bounding box [$xmin$, $ymin$, $width$, $heigh$

the above data file is a common format for bounding boxes in the object detection dataset. The bounding boxes data in YOLO is formatted as [$class$, $xcenter$, $ycenter$, $width$, $height$]

-handle the data to fit yolov5 format:

With minimum point (x_{min}, y_{min}), the center point

(x_{center}, y_{center}) can be calculated as follow:

$x_{center} = x_{min} + width/2$

$y_{center} = y_{min} + height/2$

-normalize the data coordinate to fit yolov5 format:

box coordinates must be normalized in the range 0-1. So, x_{center} and $width$ are divided by image width, y_{center} and $height$ are divided by image height.

Now each column of data respectively corresponds to the unique image id, class, x_{center}, y_{center}, width and height of bounding box

	ImageID	classNumber	center x	center y	width	height
3438	00020ebf74c4881c	1	0.885938	0.586835	0.199375	0.239963
8735	0006124f7e35107e	2	0.427553	0.298252	0.731591	0.525219
8736	0006124f7e35107e	2	0.600554	0.723604	0.729216	0.442501
16565	000d1976fc8ebfe7	1	0.015312	0.525968	0.030625	0.079320
16566	000d1976fc8ebfe7	1	0.026875	0.782814	0.053750	0.103872

2- Splitting data into training set and validation set:

By dividing it proportionally, the common rate is 80-20, with 80% of the dataset used for training and 20% used for evaluation. Thus, model performance will be assessed on 20% of data that was not used during training, or in other words, these data is data which the model

has never seen before. This ensures the review will be equitable. The data used in the training process is called the training set and the data used in the model evaluation is called the validation set.

3- Creating the custom_data.yaml file:

The YOLOv5 model on PyTorch accesses the images and uses them as input through a yaml file containing summary information about the data set. The custom_data.yaml file used in the YOLO model has the following structure:

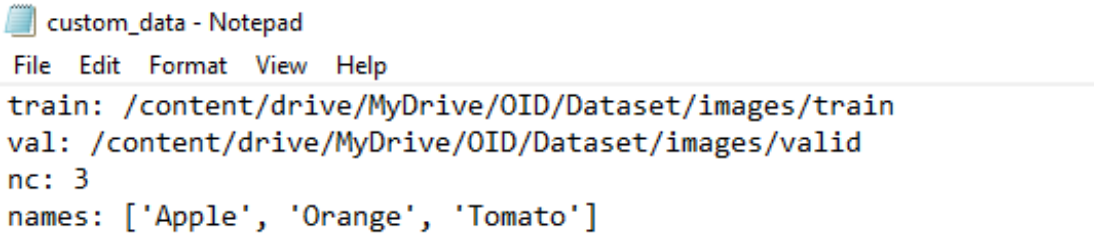


Figure-22

4- Training model:

The model will be trained by compile file train.py along with its attributes.

These following attributes represent for:

1- Image: define input image size smaller size makes the training process faster. After many experiments, the size 416 × 416 is the ideal size to use as input without losing much detail

2- Batch: determine the batch size. The forwarding of thousand images into the neural network at the same time makes the number of weights that the model learns in one time (one epoch) to increase a lot. Thus, the dataset is usually divided into multiple batches of n images and training batch by batch. The results of each batch are then saved to RAM and aggregated after the training for all batches is completed. Because the weights learned from the batches are stored in RAM, so the larger the number of batches, the more memory consumption will be consumed. The training set contains 2738 images, with $batch\ size$ = 32, the number of batches will be 2738 ÷ 32 = 86 $batches$.

3- Epochs: define the number of training epochs. An epoch is responsible for learning all input images, in other words, training all input. Since the dataset is split into multiple batches, one epoch will be responsible for training all the batches. The number of epochs represents the number of times the model trains all the inputs and updates the weights to get closer to the ground truth labels. Often chosen based on experience and intuition. The number of epochs more than 3000 is normal

4- Data: the path to custom_data.yaml file containing the summary of the dataset. The model evaluation process is executed immediately after each epoch, so the model will also access the

validation directory via the path in custom_data.yaml file and use its contents for evaluation at that moment

5- Weights: specify a path to weights. A pretrained weight can be used for saving training time. If it is left blank, the model will automatically initialize random weights for training.

6- cache: cache images for training faster

Result of training:

The model saves 2 weighting results as pt file, last.pt file is the weight at the last epoch and best.pt file is the weight at the last epoch for the highest accuracy. The size of both files is only 14MB, making it very light to integrate into AI systems as a pretrained weight while maintaining 95.4% accuracy

5- Inference with trained weight

Trained weights can be used to identify the Fruits on any image. If the presence of a fruit is detected, a bounding box is drawn to encase the object and display the probability that the object is the Fruits.

detect.py file will be compiled, and it rebuilds the architecture used in the training. Trained weights will be used to predict objects and limit boxes for them with 95.4% accuracy.

After the detection is complete, the predicted bounding boxes that cover the objects (Fruits) will be drawn into the image. They will be saved in the same folder containing results from the training phase.

The model gives out extremely impressive predictive results on even the images it has not seen before. Although the density of fruit in each image is very high, the model almost detects that all fruits appear in the image. As can be seen in images, although the fruits are correctly predicted, the probability for them is not high.

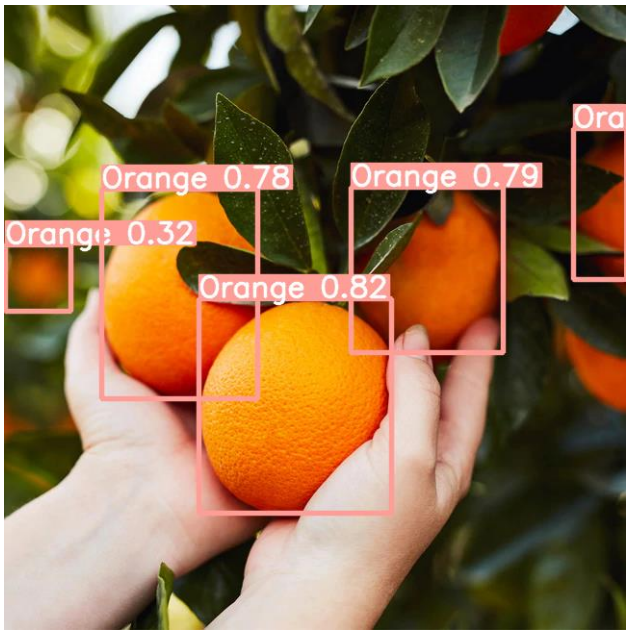


Figure-23

Testing

Test Cases:

1- Image don’t have any trained object:

Ex: Avocado tree

Figure-24

2- Image contains of Tomato

Figure-25

3- Image Contain of Apple

Figure-26

4- Image Contain of Orange

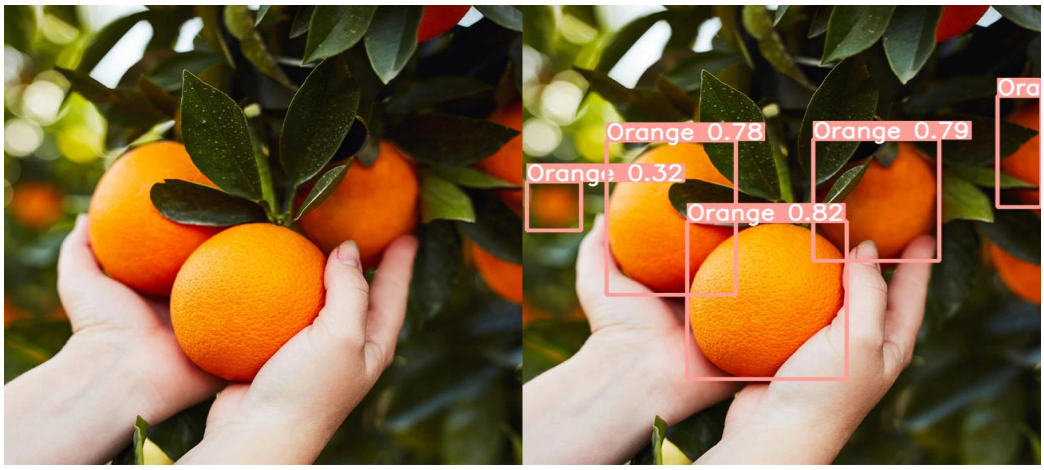


Figure-27

5- Image Contain Apple and Orange

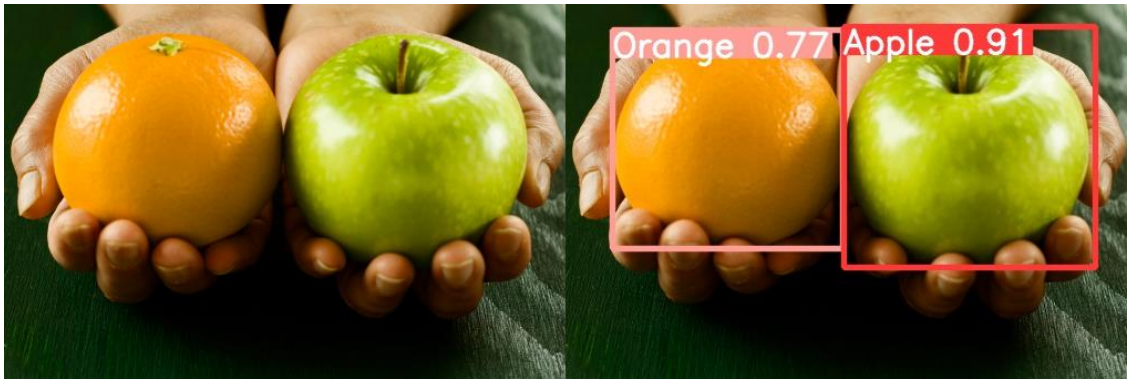


Figure-28

6- Image Contain Apple and Tomato

Figure-29

7- Image Contain Orange and Tomato

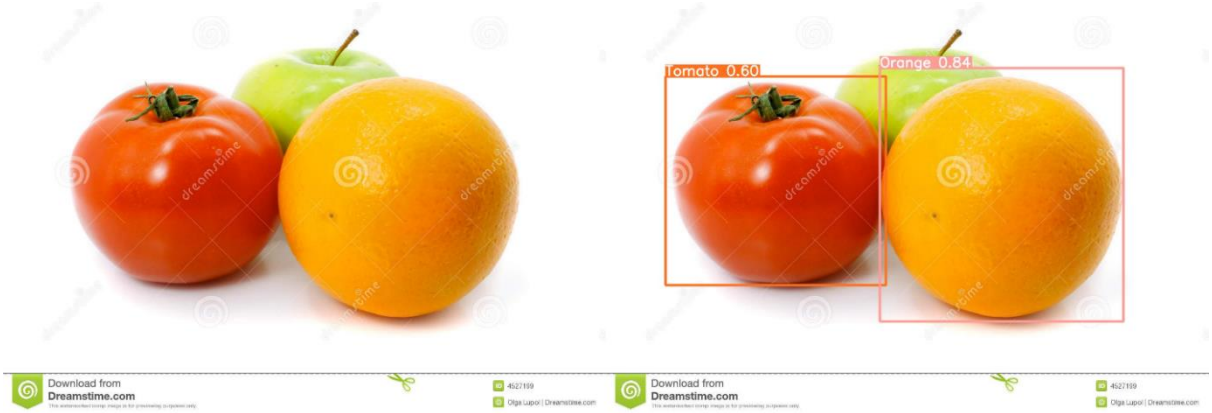


Figure-30

Chapter 5:

BANANA RIPENESS DETECTION USING YOLOV5

Bananas are one of the most consumed fruits globally. It contributes about 16% of the world's fruit production according to FAO (Food and Agriculture Organization). Almost every third person in the world eats the banana. Banana is a great source of energy; it provides fullness in the body. In every sport, banana is given as an energy booster for players. Eating an overripe banana can be caused you a series of diseases, problems with the stomach, and more. For the last 10 years, people are trying to find the correct ripening stage of banana. The ripening stage of fresh banana fruit is a principal factor that affects the fruit quality during ripening and marketability after ripening, it is also very helpful in increasing the production. Finding the correct ripening stage of banana can be tougher if we know that we produce an average of 100tons of banana every day. we need some intelligent systems that can detect the correct stage of banana.

Figure-30

Data Preparation

Our task to create a machine learning model that can predict the ripeness stage of banana based on the image. In our project, we are classifying the ripeness stage into three categories ripe, unripe, and overripe. We need to prepare the data for the same.

Gathering Data:

One of the crucial parts of building machine learning systems is gathering high quality dataset. You can expect to spend significant amount of time on data. It is essential because our model is only as good as the data it learns from. In this series, we will try build an object detector that is trained to detect level ripeness of banana in the scene. So how do we teach a machine to detect ripeness level? As you might have guessed, by showing a lot of examples. There are various ways to collect data. When it comes to images, one of the easiest ways for an individual to collect images is to use Google Image Search.

We will try build an object detector that is trained to detect ripe and unripe banana in the scene. So how do we teach a machine to detect that? As you might have guessed, by showing a lot of examples.

There are various ways to collect data. When it comes to images, one of the easiest ways for an individual to collect images is to use Google Image Search.

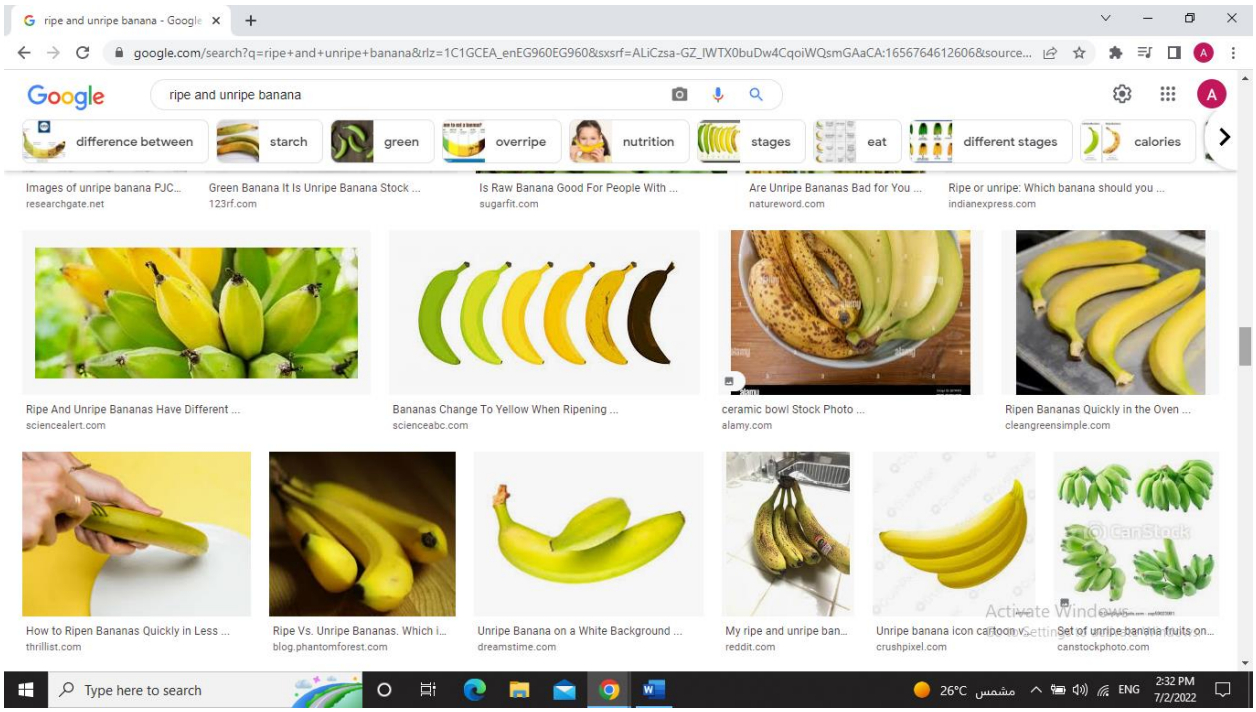


Figure-31

But downloading the images show up in search results one by one manually is a tedious task. Luckily there are some tools to help. Came across a chrome extension called Download All Images which, as the name says, downloads all images present in a web page with a click of a button.

Labeling Data:

Once we gather data, next step is to label / annotate them. In the context of object detection, labeling means drawing bounding boxes around the objects that we are interested in detecting in the images and associating them with corresponding object classes / categories so that we can show it to the machine clearly.

Figure-32

There are lot of tools available to help us annotate the images. One of the widely used Open-Source tools is LabelImg.

Good thing about LabelImg is, it lets us save the annotations directly into YOLO format. Some tools don't directly do that. We need to convert the annotations ourselves into the format YOLO requires.

You can easily install LabelImg with pip from the Terminal.

```
pip install labelimg
```

Once labelImg is successfully installed, launch it by typing

```
labelImg [path to image] [classes file]
```

[path to image] is the path to an image in the directory containing the downloaded helmet images we are going to label.

[classes file] is the file where we list the object classes that we are going to label.

Creating necessary files:

File contains:

1. train, test, and val: Locations of train, test, and validation images.
2. nc: Number of classes in the dataset.
3. names: Names of the classes in the dataset. The index of the classes in this list would be used as an identifier for the class names in the code. We have 6 classes ['freshripe', 'freshunripe', 'overripe', 'ripe', 'rotten', 'unripe']

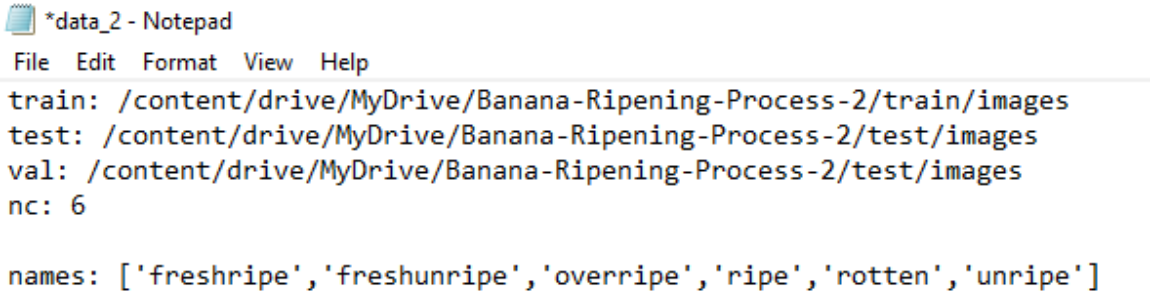


Figure-33

Model Training

The model will be trained by compile file train.py along with its attributes.

```
!python train.py --img 640 --batch 16 --epochs 3 --data data_2.yaml --weights yolov5s.pt –cache
```

Inference

There are many ways to run inference using the detect.py file.

The source flag defines the source of our detector, which can be:

1. A single image
2. A folder of images
3. Video
4. Webcam

...and various other formats.

- The weights flag defines the path of the model which we want to run our detector with.
- conf flag is the thresholding abjectness confidence.
- `best.pt` contains the best-performing weights saved during training.

```
python detect.py --weights best.pt --img 640 --conf 0.35 --source tt3.jpg
```

Figure-34

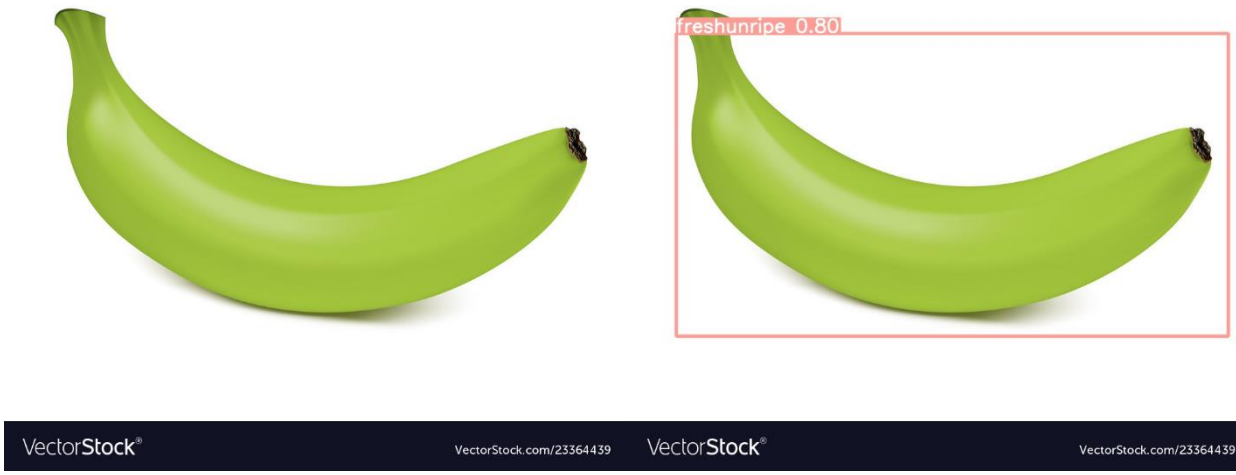


Figure-35

Figure-36

Chapter 6:

HARDWARE IMPLEMENTATION

Intro

This chapter contains the required hardware components, their concept of operation (how they work).so we show the project idea again to abstract what project needs.

The project Workflow:

The proposed system is powerful in banana production lines where it can classify banana if it is rotten, ripe or unripe.

The banana is put on production line (convey). As soon as the IR sensor detect object (banana), the MCU is interrupted and the convey is stopped then MCU send READY signal by USART on serial port of laptop then the laptop make the camera capture picture of current banana which positioned under camera after that the ML model take decision if it is rotten, ripe or unripe then laptop send decision on its serial port to USART of MCU then the robotic arm take the current banana and put it in specific box according to its class (rotten, ripe or unripe) where rotten and unripe bananas are put in same box and ripe banana is put in another box then the MCU send FINISH signal by USART on serial port of laptop then perform that operation on another banana.

So, we need to illustrate the following peripherals:

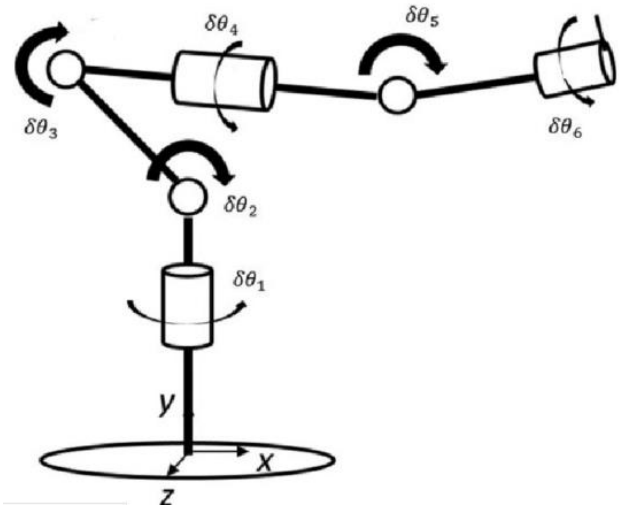
Peripherals
1. Robotic Arm
2. Convey
3. IR Sensor
4. MCU
5. Laptop
6. Camera

1. Robotic Arm:

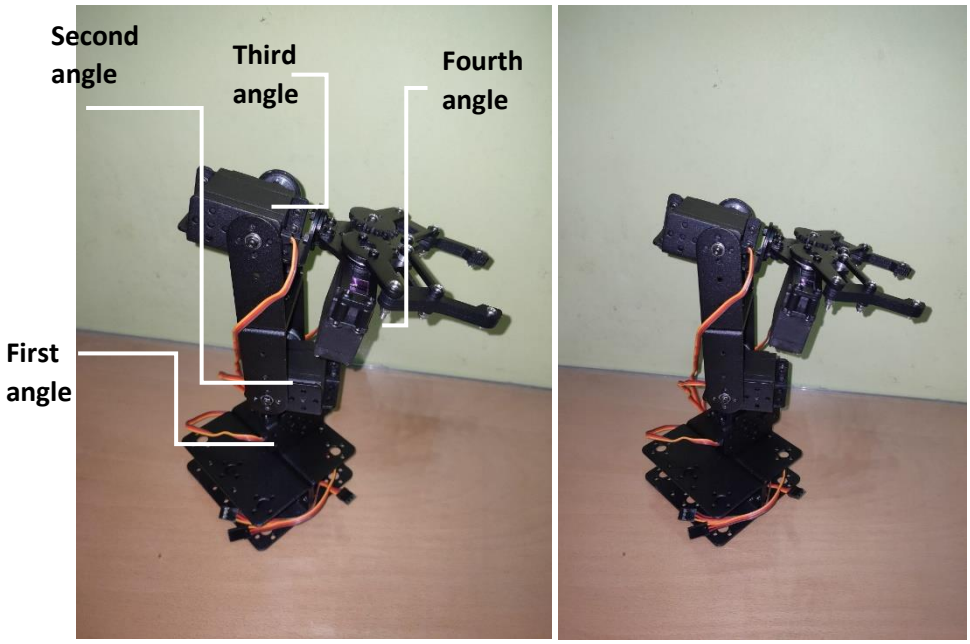
In this section, we will illustrate how robot arm move with specific angles, how many degrees of freedom it have, the software and hardware that we need to control the robotic arm.

The concept of operation:

ARFQS prototype uses robotic arm with 4 DOF (degree of freedom) where the degree of freedom term means the number of angels or joint that the robotic arm contain. For example, 6 degree of freedom robotic arm means that it contains six joint (six axes) as shown



In practical life, we replace every joint with any type of motor whether servo or stepper. But we use servo motor. So, our robotic arm has 4 servo motor to correspond 4 DOF as shown in following figure. So, we will illustrate briefly how servo motor work.



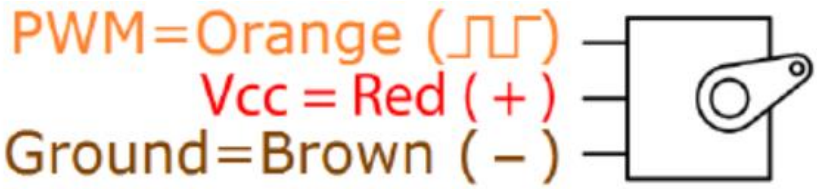
Servo motor:

The used type servo is MG996R servo motor.MG996R wiring:

MG996R servo motor Pinout: [10]

Wire Color	Description
Brown	Ground wire connected to the ground of system
Red	Powers the motor typically +5V is used
Orange	PWM signal is given in through this wire to drive the motor

Main specifications MG996R servo motor:



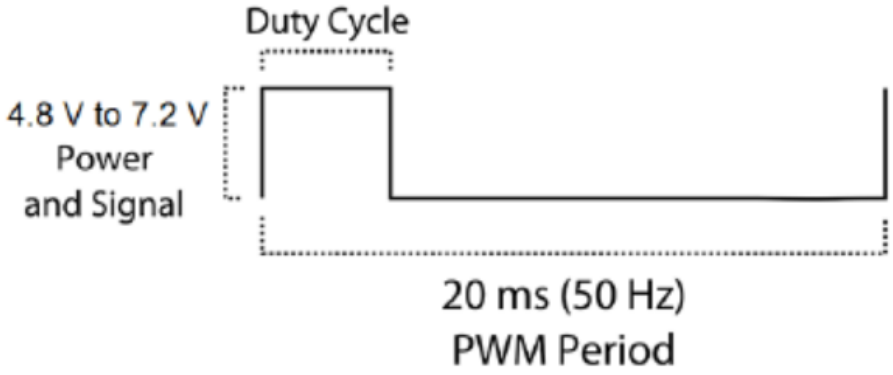
From datasheet, we get the following information:

Operating voltage: 4.8 V a 7.2 V
Running Current 500 mA – 900 mA (6V)

So, the typical operation condition is 6 volt and 900 mA.

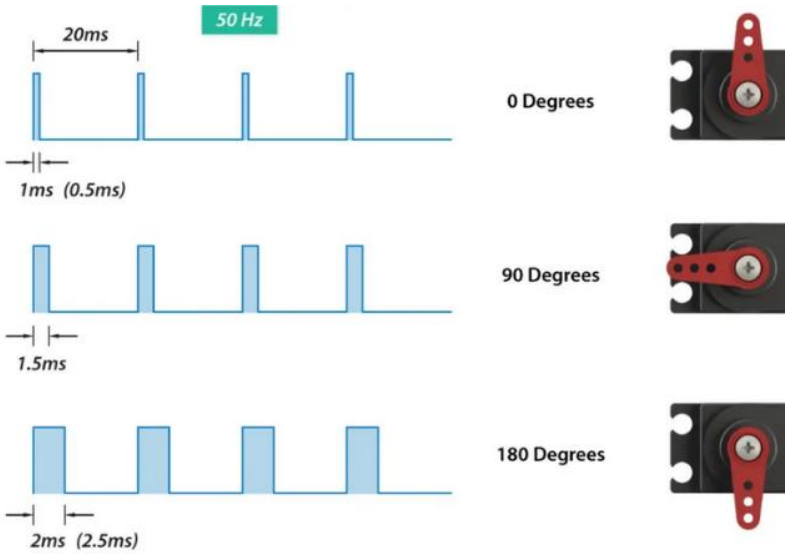
Working principle of servo motor:

Controlling a servo is easy by using a microcontroller, no external driver like h-bridge etc. are required. Just a control signal is needed to be feed to the servo to position it in any specified angle. The frequency of the control signal is 50 Hz (i.e. the period is 20ms) and the width of positive pulse controls the angle. As we know there are three wires coming out of this motor. The description of the same is given on top of this page.



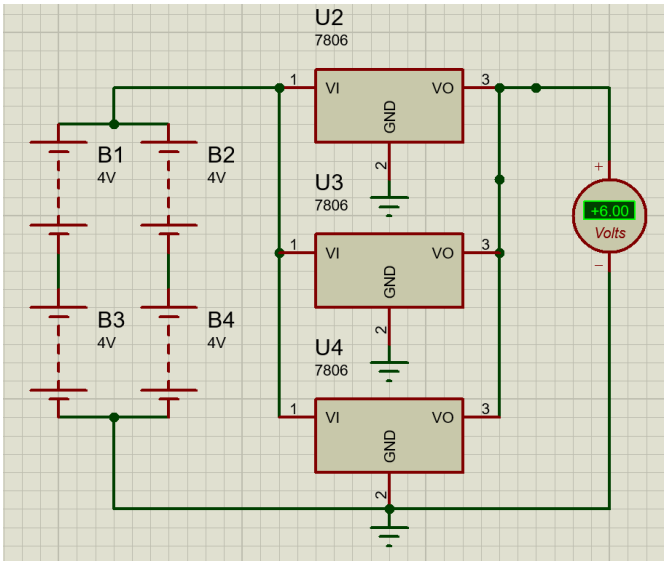
To make this motor rotate, we have to power the motor using the Red and Brown wire and send PWM signals to the orange color wire. Hence, we need something that could generate PWM signals to make this motor work, this something could be anything like a 555 Timer or other Microcontroller platforms like Arduino, PIC, ARM, AVR or even a microprocessor like Raspberry Pi. To understand that let us a look at the picture given in the datasheet.

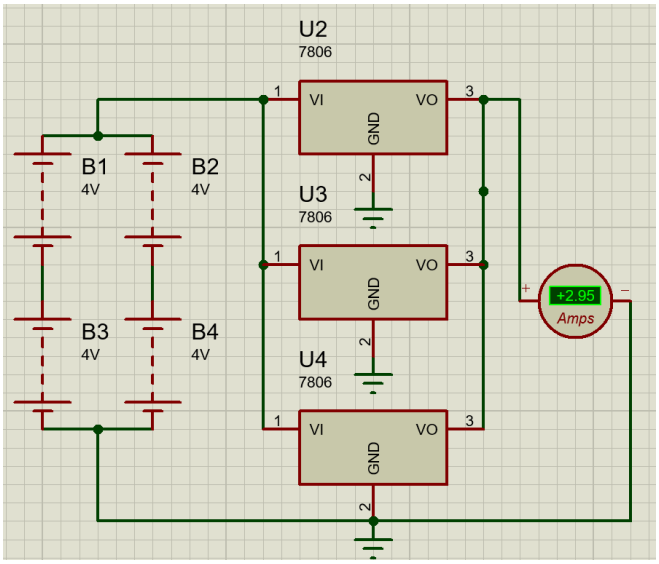
From the picture we can understand that the PWM signal produced should have a frequency of 50Hz that is the PWM period should be 20ms. Out of which the On-Time can vary from 1ms to 2ms. So as shown in following figure when the on-time is 1ms the motor will be in 0° and when 1.5ms the motor will be 90°, similarly when it is 2ms it will be 180°. So, by varying the on-time from 1ms to 2ms the motor can be controlled from 0° to 180°.



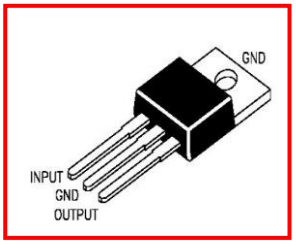
Robotic arm power:

Our robotic arm has 4 MG996R servo motor. So, we need 6-volt 900 mA power source for each servo. So, we can use three parallel 6-volt voltage regulator to regulate 8-volt battery where each voltage regulator supply 6 volt and 1 A to 1.5 A. Then overall circuit is supply 6 volt and 3 A to 4.5 A as shown in following figure:





Where we use LM 7806 voltage regulator and its specification as follow: [11]



Features

■ Output Current up to 1A

Electrical Characteristics (LM7806)

(Refer to the test circuits. $-40°C < T_J < 125°C$, $I_O = 500mA$, $V_I = 11V$, $C_I = 0.33\mu F$, $C_O = 0.1\mu F$, unless otherwise specified)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
Output Voltage	V_O	$T_J = +25°C$		5.75	6.0	6.25	V
		$5mA \le I_O \le 1A$, $P_O \le 15W$, $V_I = 8.0V$ to $21V$		5.7	6.0	6.3	
Line Regulation (Note 4)	Regline	$T_J = +25°C$	$V_I = 8V$ to $25V$	–	5.0	120	mV
			$V_I = 9V$ to $13V$	–	1.5	60.0	
Load Regulation (Note 4)	Regload	$T_J = +25°C$	$I_O = 5mA$ to $1.5mA$	–	9.0	120	mV
			$I_O = 250mA$ to $750mA$	–	3.0	60.0	
Quiescent Current	I_Q	$T_J = +25°C$		–	5.0	8.0	mA
Quiescent Current Change	ΔI_Q	$I_O = 5mA$ to $1A$		–	–	0.5	mA
		$V_I = 8V$ to $25V$		–	–	1.3	
Output Voltage Drift (Note 5)	$\Delta V_O/\Delta T$	$I_O = 5mA$		–	-0.8	–	mV/°C
Output Noise Voltage	V_N	$f = 10Hz$ to $100KHz$, $T_A = +25°C$		–	45.0	–	$\mu V/V_O$
Ripple Rejection (Note 5)	RR	$f = 120Hz$, $V_O = 8V$ to $18V$		62.0	73.0	–	dB
Dropout Voltage	V_{DROP}	$I_O = 1A$, $T_J = +25°C$		–	2.0	–	V
Output Resistance (Note 5)	rO	$f = 1KHz$		–	19.0	–	mΩ
Short Circuit Current	I_{SC}	$V_I = 35V$, $T_A = +25°C$		–	250	–	mA
Peak Current (Note 5)	I_{PK}	$T_J = +25°C$		–	2.2	–	A

2. Convey:

Any production line usually needs convey which its working depends on motors either servo, stepper or DC motor and for simplification we use DC motor. We cannot control DC motor directly from MCU because DC motor drives more than current which MCU can supply any device (40 mA). If we use MCU to control DC motor directly, that leads to damage to this pin. So, we use device called motor driver (L298N motor driver chip).

L298N motor driver chip: [12]

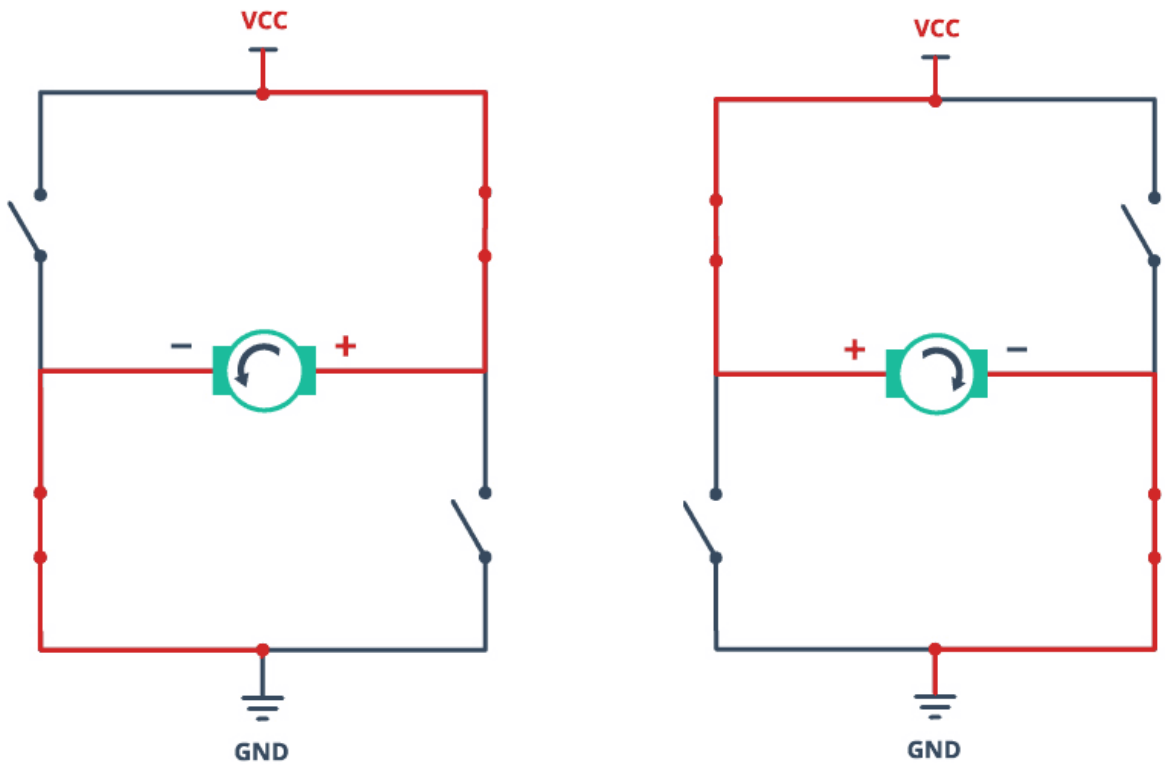
To have complete control over DC motor we have to control its speed and rotation direction. This can be achieved by combining these two techniques.

- H-Bridge – to control the rotation direction
- PWM – to control speed

H-Bridge – to control the rotation direction

The spinning direction of a DC motor can be controlled by changing the polarity of its input voltage. A common technique for doing this is to use an H-bridge. An H-bridge circuit consists of four switches (transistor) with the motor in the center forming an H-like arrangement. Closing two specific switches at a time reverses the polarity of the voltage applied to the motor. This causes a change in the spinning direction of the motor.

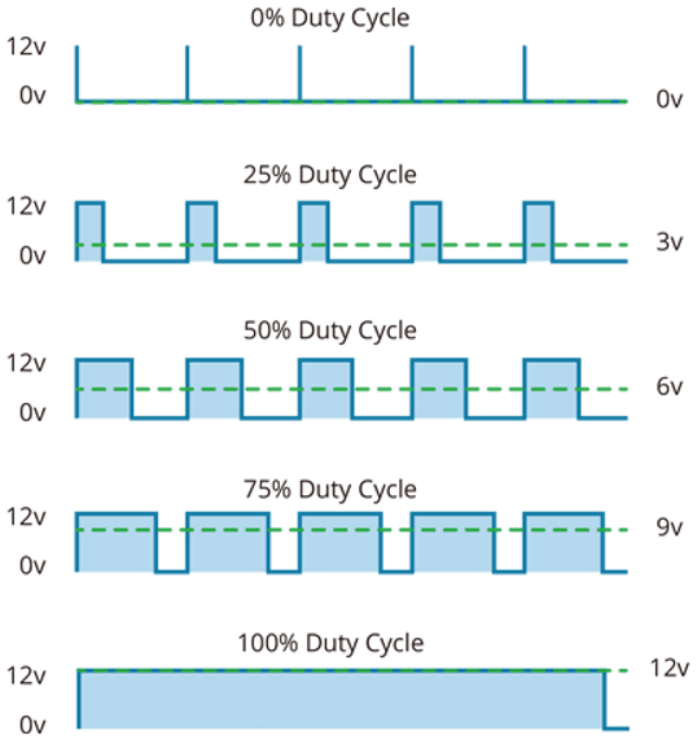
The following figures show the working of the H-bridge circuit:



PWM – to control speed:

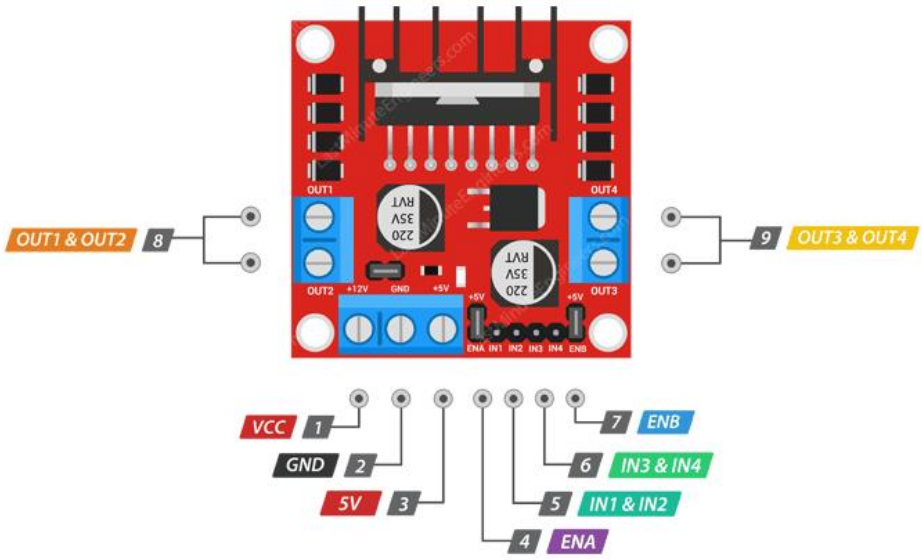
The speed of a DC motor can be controlled by changing its input voltage. A common technique to do this is to use PWM (Pulse Width Modulation). PWM is a technique where the average value of the input voltage is adjusted by sending a series of ON-OFF pulses. The average voltage is proportional to the width of the pulses known as the Duty Cycle. The higher the duty cycle, the higher the average voltage applied to the DC motor (resulting in higher speed) and the shorter the duty cycle, the lower the average voltage applied to the DC motor (resulting in lower speed).

The image below shows PWM technology with different duty cycles and average voltages:



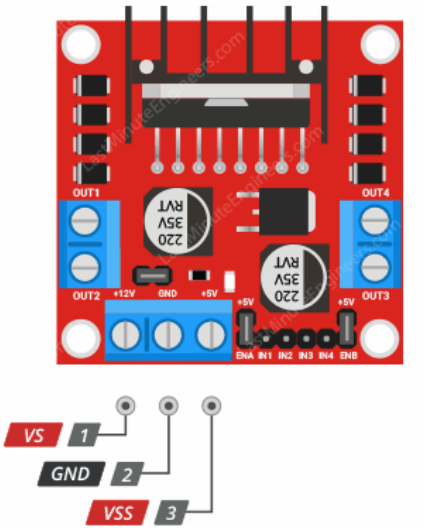
L298N Motor Driver Module Pinout:

The L298N module has a total of 11 pins that connect it to the outside world. The pins are as follows:



Power Pins:

The L298N motor driver module is powered through 3-pin 3.5mm-pitch screw terminal.



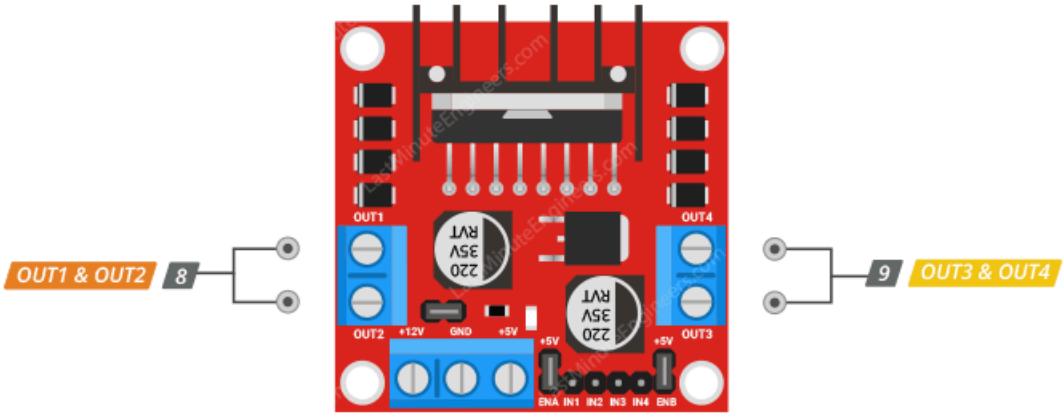
V_S pin gives power to the internal H-Bridge of the IC to drive the motors. You can connect an input voltage anywhere between 5 to 12V to this pin.

V_{SS} is used to drive the logic circuitry inside the L298N IC which can be 5 to 7V.

GND is the common ground pin.

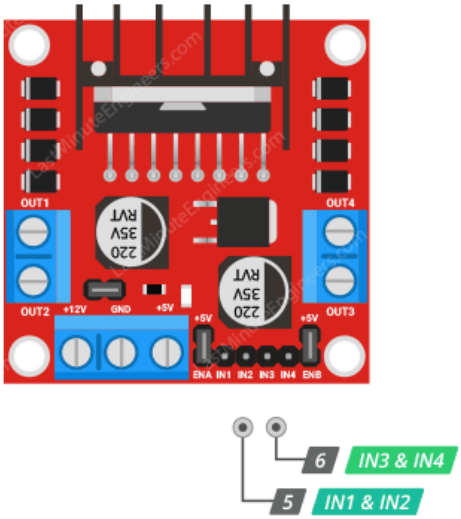
Output Pins:

The L298N motor driver's output channels OUT1 and OUT2 for motor A and OUT3 and OUT4 for motor B are broken out to the edge of the module with two 3.5mm-pitch screw terminals. You can connect two 5-12V DC motors to these terminals. Each channel of the module can deliver up to 2A to the DC motor. However, the amount of current supplied to the motor depends on the power supply of the system.



Direction Control Pins:

By using the direction control pins, you can control whether the motor rotates forward or backward. These pins actually control the switches of the H-Bridge circuit inside the L298N chip.



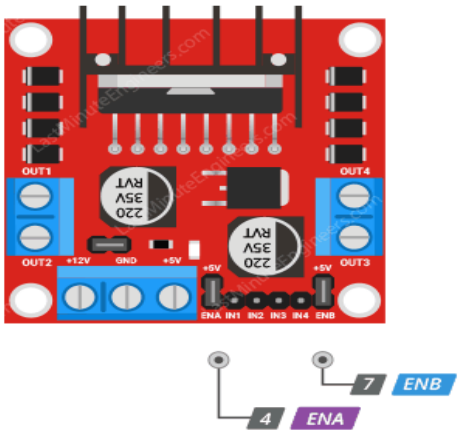
The module has two direction control pins for each channel. The IN1 and IN2 pins control the spinning direction of motor A; While IN3 and IN4 control the spinning direction of motor B.

The spinning direction of the motor can be controlled by applying logic HIGH (5V) or logic LOW (Ground) to these inputs. The chart below shows how this is done.

IN1	IN2	Spinning Direction
LOW (0)	LOW (0)	Motor OFF
LOW (0)	HIGH (1)	Forward
HIGH (1)	LOW (0)	Backward
HIGH (1)	HIGH (1)	Motor OFF

Speed Control Pins:

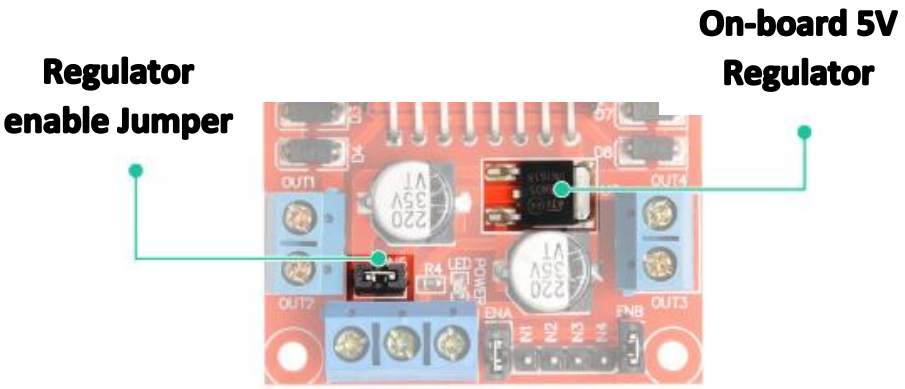
The speed control pins ENA and ENB are used to turn on/off the motors and control its speed.



Pulling these pins HIGH will cause the motors to spin, while pulling it LOW will stop them. But, with Pulse Width Modulation (PWM), you can actually control the speed of the motors. The module usually comes with a jumper on these pins. When this jumper is in place, the motor spins at maximum speed. If you want to control the speed of the motors programmatically, you need to remove the jumpers and connect them to the PWM pins of the MCU.

On-board 5V Regulator and Jumper:

The module has an on-board 5V regulator – 78M05. It can be enabled or disabled via a jumper.



When this jumper is in place, the 5V regulator is enabled, which derives the logic power supply (VSS) from the motor power supply (VS). In this case, the 5V input terminal acts as the output pin and delivers 5V 0.5A. You can use it to power an Arduino or other circuitry that requires a 5V power supply.

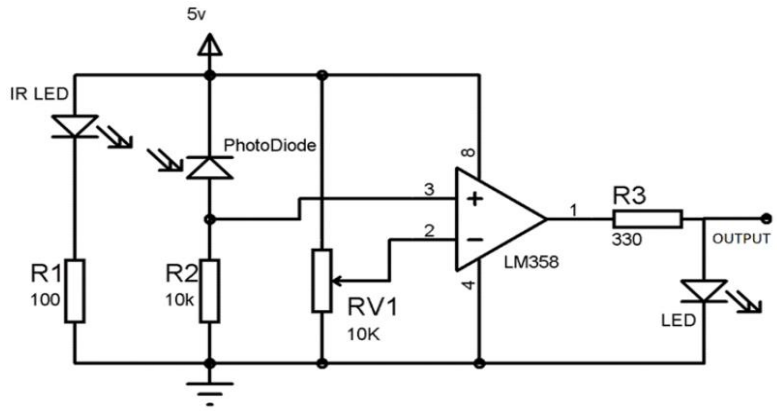
When the jumper is removed, the 5V regulator is disabled and we have to separately supply 5V through the VSS pin.

3. IR Sensor module:

Working principle:

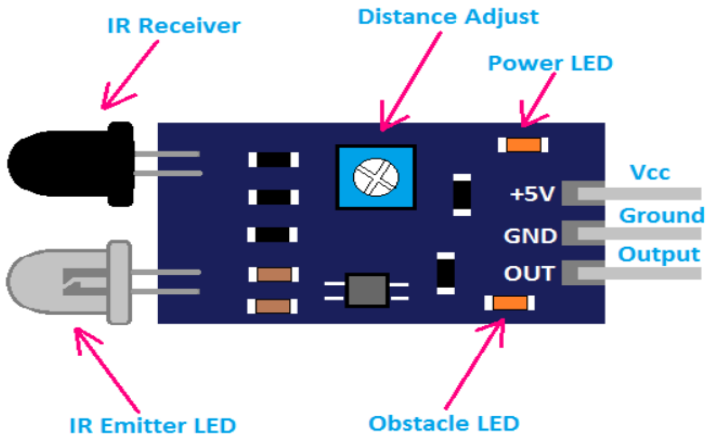
The IR sensor module includes five essential parts like IR TX, RX, Operational amplifier, trimmer pot (variable resistor) & output LED.

The emitter is simply an IR LED (Light Emitting Diode) and the detector is simply an IR photodiode. Photodiode is sensitive to IR light of the same wavelength which is emitted by the IR LED. When the photodiode is exposed to IR light, the electrical resistance across the diode decreases, increasing the reverse current. If the photodiode is not exposed to light, the resistance across it will be high. When the IR receiver does not receive a signal, the potential at the inverting input goes higher than that non-inverting input of the comparator IC. Thus, the output of the comparator goes low, but the LED does not glow. When the IR receiver module receives a signal to the potential at the inverting input goes low. Thus, the output of the comparator goes high and the LED starts glowing.



The pin configuration of the IR sensor module is discussed below:

Pin	Description
V_{CC}	Power supply input
GND	Power supply ground
OUT	Active high O/P



The main specifications and features of the IR sensor module include the following:

- The operating voltage is 5 VDC.
- I/O pins – 3.3V & 5V.
- Mounting hole.
- The range is up to 20 centimeters.
- The supply current is 20mA.
- The range of sensing is adjustable.
- Fixed ambient light sensor.

4. MCU:

The MCU is widely used in embedded systems life. So this project need MCU from AVR family which is ATMEGA32. This section contains explanation what ATMEGA32 do in BDR project.

As we explained before, the ARFQS mainly based on convey which based on DC motor, robotic arm which based on servo motors, the IR module working in our project on interrupt concept and communication between ATMEGA32 and laptop which based on USART communication protocol. So, the driver (peripherals) we need is:

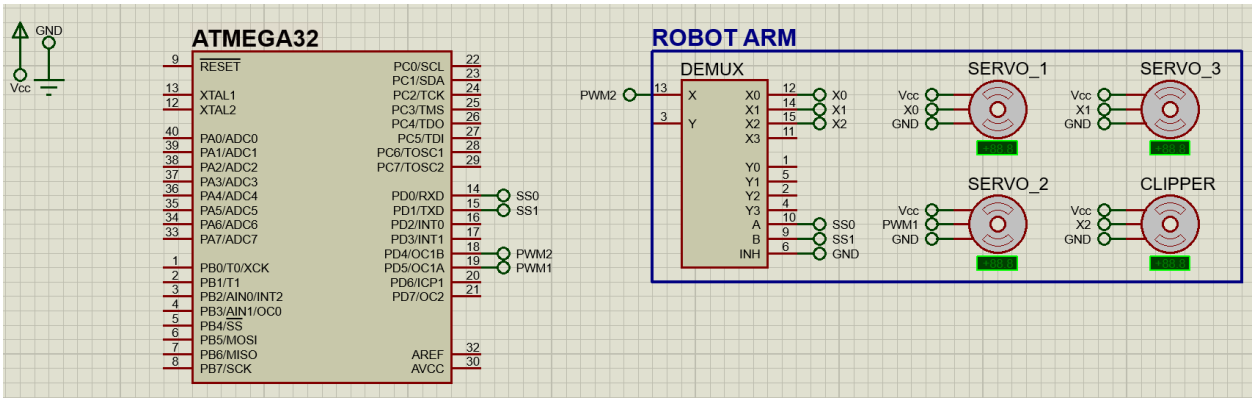
- Robotic arm driver:
    - DIO driver.
    - Servo driver:
        - PWM driver.
- DC motor driver:
    - PWM driver.
- Interrupt driver.
- USART driver.

Basic driver	using
PWM driver	To drive motor like servo and DC motor
USART driver	Communication between laptop and AVR
Interrupt driver	With IR module
DIO driver	To set pin direction.

PWM driver:

We use this driver in both of servo driver and DC motor driver. The timer peripheral is commonly used to generate PWM signal where ATMEGA32 have three timers (TIMER0, TIMER1, and TIMER2). The DC motor needs PWM signal with any frequency to control its speed so TIMER0 used for that purpose. The servo motor needs PWM signal with specific frequency (50 HZ) to control its rotation angle so only TIMER1 used for that purpose.

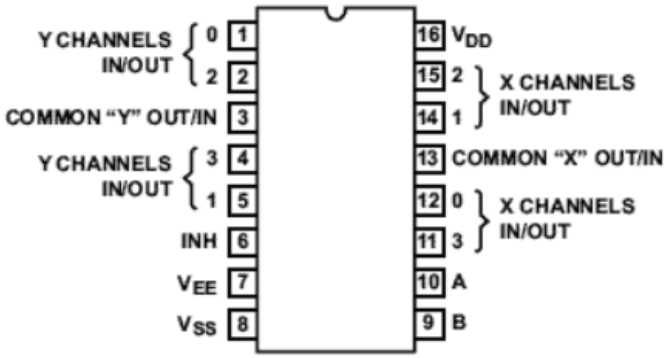
But there are problem that we have only two channel in TIMER1 and we have four servo motor so we solve that problem with switching concept using DEMUX (demultiplexer) where one channel used to control three unloaded servo motor and another channel used to control the fourth servo motor (we need dedicated channel for loaded servo because if we turn off the PWM signal from that servo that causes error in rotation angle or drift in required angle after turning off the PWM signal due to load).



DEMUX:

We use 4052 chip which generally used as both MUX and DEMUX.

4052-chip as DEMUX pinout:



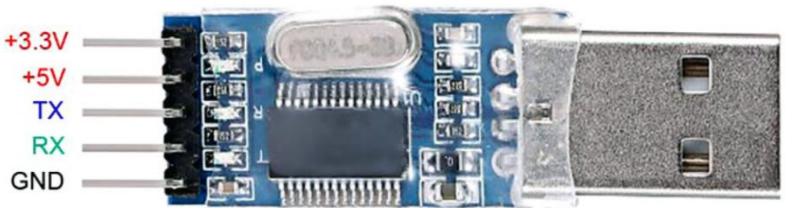
Pin Number	Pin Name	Description
12, 14, 15, 11	X0, X1, X2, X3	output pins of channel x
1,5,2,4	Y0, Y1,Y2, Y3	output pins of channel y
3, 13	Y, X	input for channel X and Y
6	INH (Enable pin)	For normal operation, it should be connected to ground.
7	VEE	Negative supply
8	VSS	Ground of the circuit
10,9	A, B	Select pin for channel 0, 1, 2, 3
16	VDD	Positive Supply

USART driver:

USART peripheral used for communication between laptop and ARMEGA32 to transfer data from ATMEGA32 to laptop which tell that one banana is ready to be captured and classified and transfer data to ATMEGA32 which tell that if captured banana is rotten or fresh.

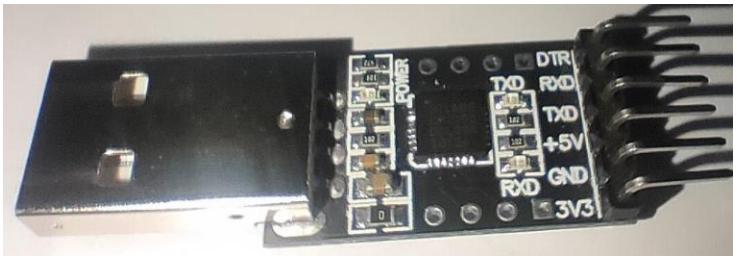
But there is small problem that laptop have USB port so we need device to convert from USB to TTL (UART) which called USB to TTL converter.

USB to TTL converter pinout:

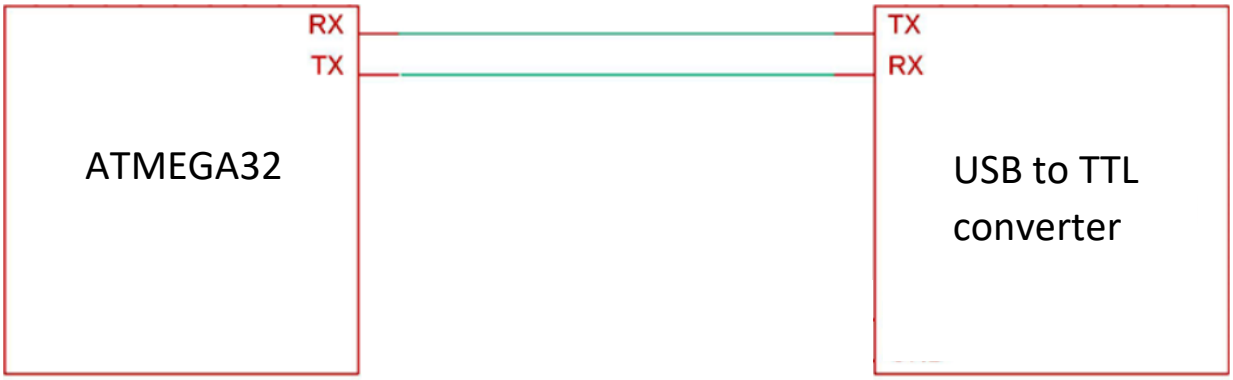


Pin Name	Description
3V3	3.3V VCC pin out
+5V	5.0V VCC pin out
TXD	Asynchronous data output (UART Transmit)
RXD	Asynchronous data input (UART Receive)
GND	Ground

There are another shape chips with DTR pin like the following chip (which we used in our project):



Where the TX of USB to TTL converter connected to RX of ATMEGA32 and RX of USB to TTL converter connected to TX of ATMEGA32.

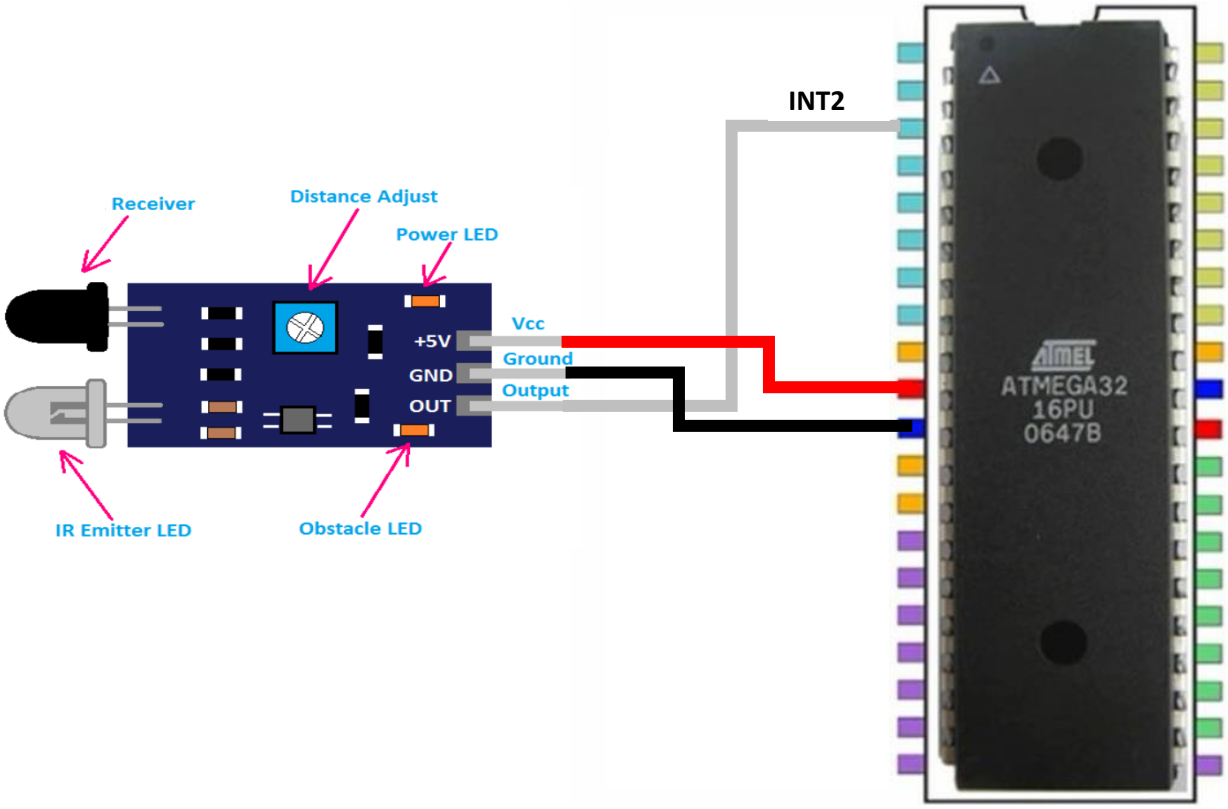


DIO driver:

DIO peripheral used to set pin direction like set IN1 and IN2 as two output pins for rotation direction of DC motor, set INT2 pin as input pin to receive interrupt signal and so on.

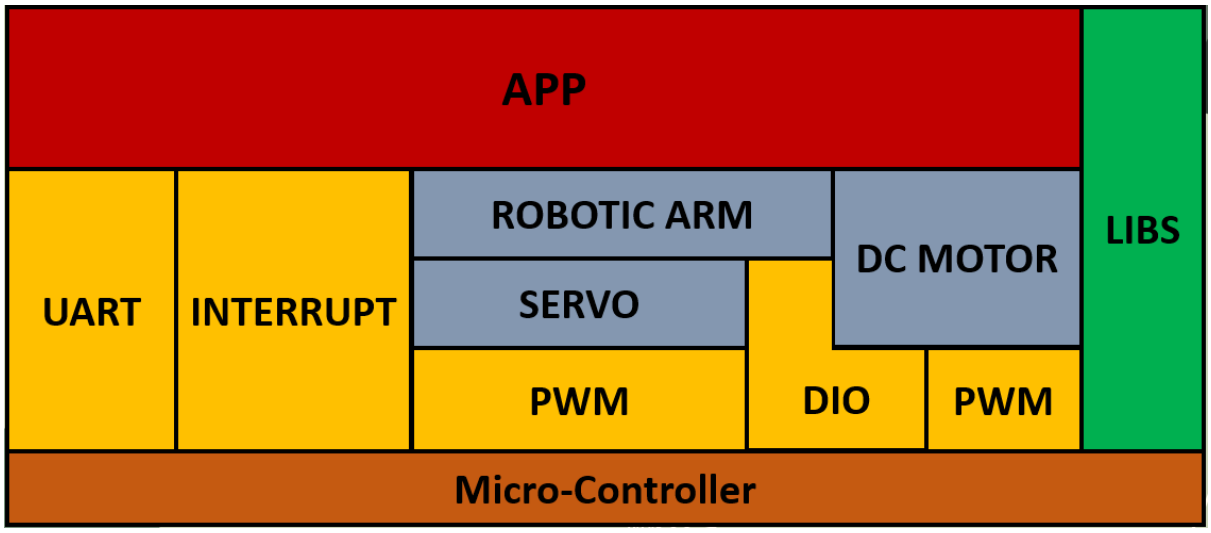
Interrupt driver:

Interrupt peripheral used for interrupt ATMEGA32 when the IR module detects banana to send ready signal to laptop. We use INT2 in our project.



Static software design:

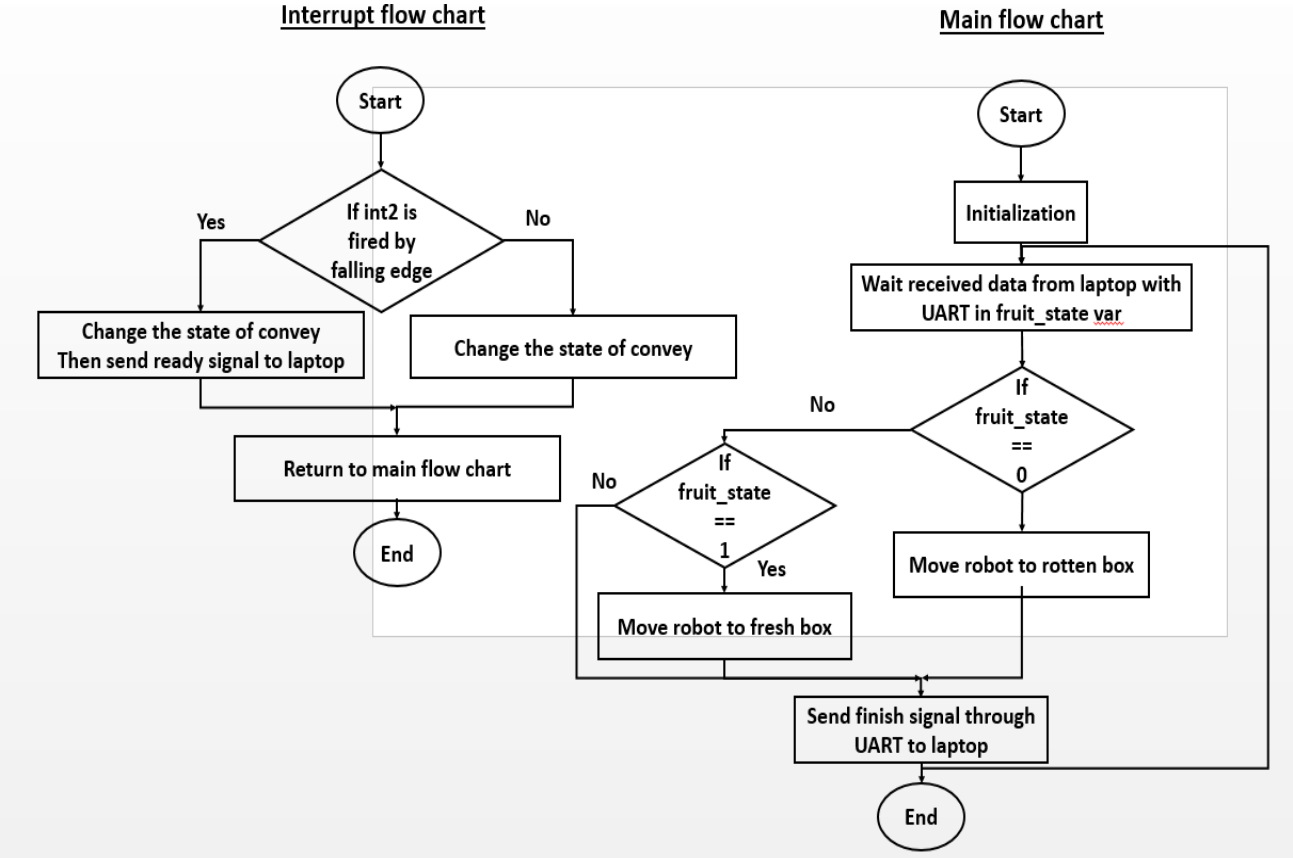
Layered Architecture:



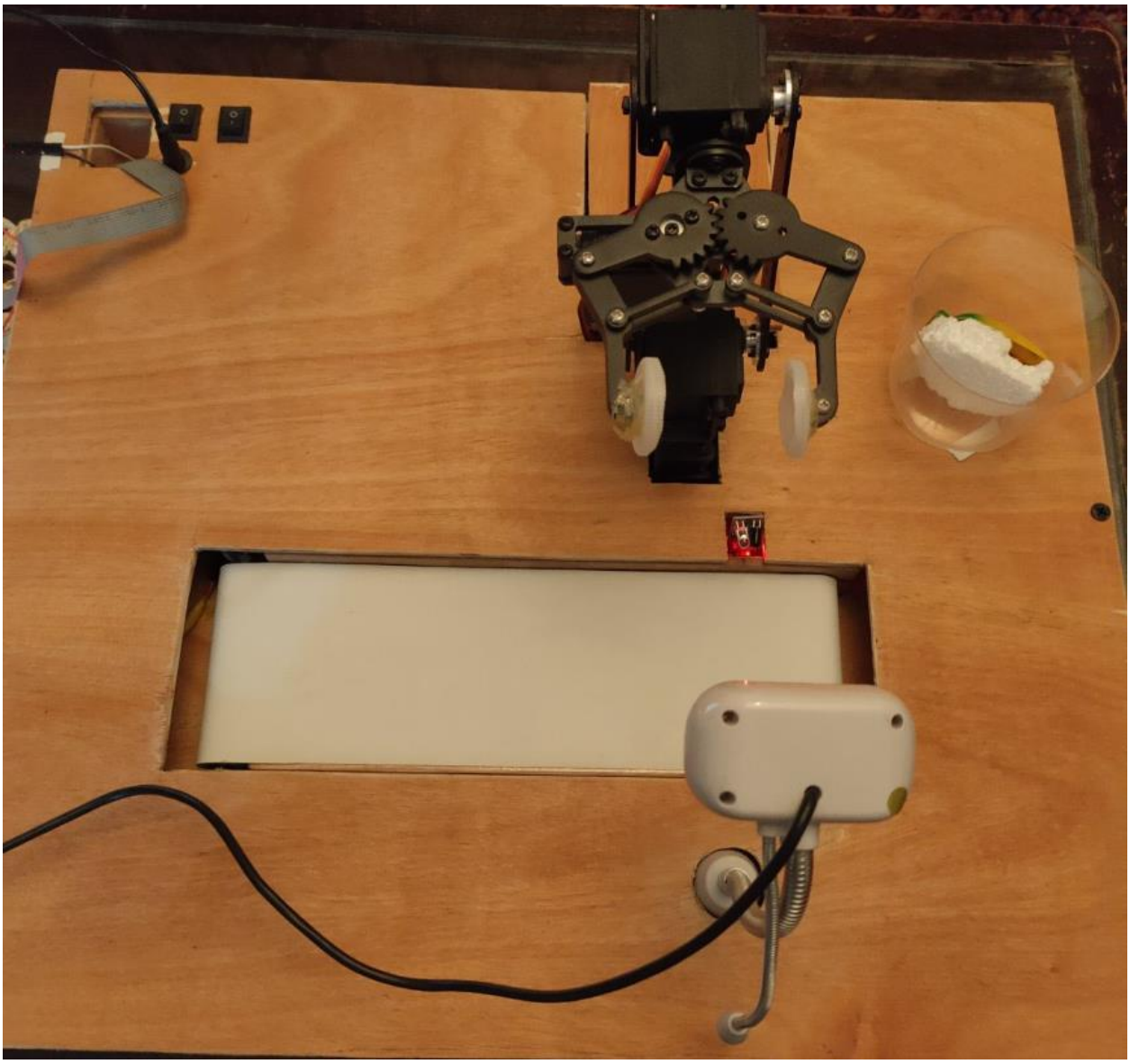
Layer Modules:

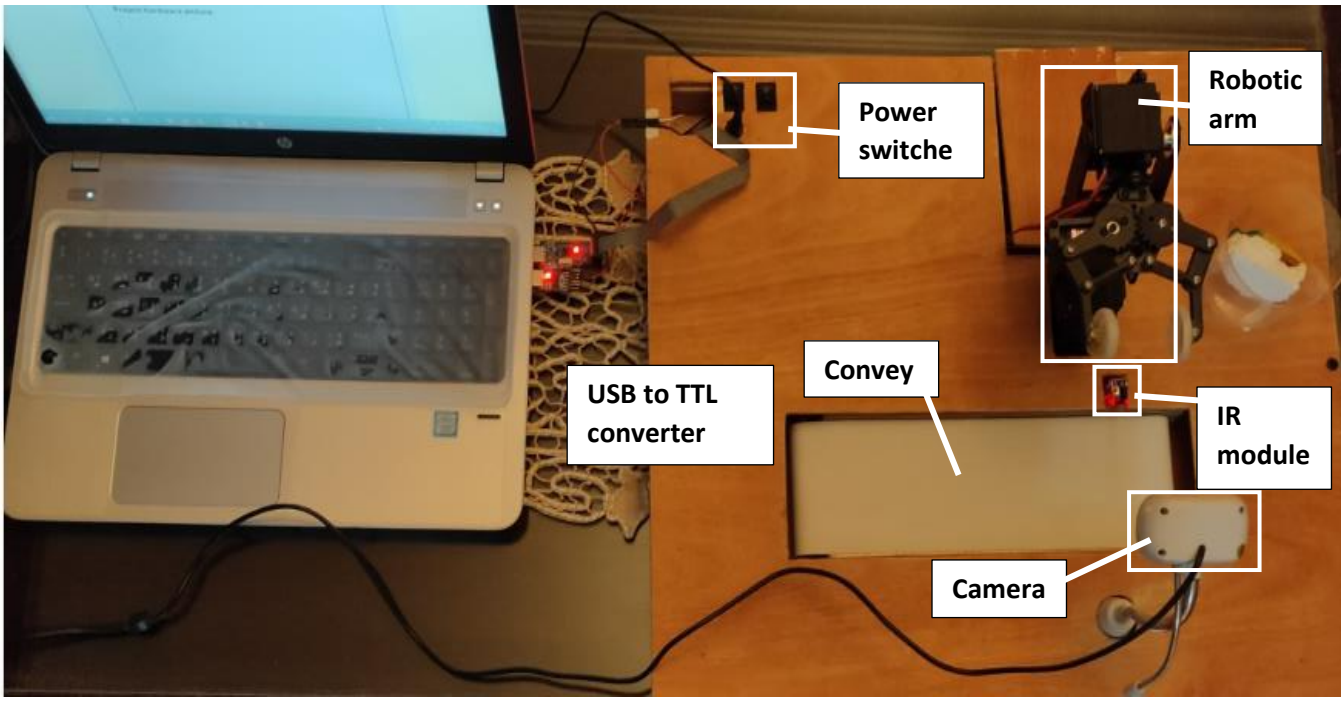
- **MCAL modules:**
  - DIO module.
  - Interrupt module.
  - PWM module.
- **HAL modules:**
  - Servo module:
    - Robotic arm module.
  - DC motor module.

APP layer flow chart:



Project hardware picture:




Power switche
Robotic arm
USB to TTL converter
Convey
IR module
Camera

Chapter 7:

KINEMATICS OF ROBOTIC ARM

Abstract:

In the next papers I will explain the solution and meaning of kinematics of 3-DOF for a robot arm , Which the kinematics divided into two solution forward kinematics and inverse kinematics with simulation by using MATLAB program .

The forward kinematics will be solved by Denavit Hardenberg (D-H) method.

The inverse kinematics will be solved by using geometric method.

Keywords: forward kinematics, inverse kinematics, robot arm

Introduction:

Robots play a huge role in this area of technology. Many industries utilize these manipulators to reduce cost and time of production. We use robot arm on line production to classify the fruits and its classes.

To control in robot arm we use the kinematics that describe the motion of robot arm without consideration the forces that cause the motion , the kinematics divided to two sub problems forward and inverse kinematics .

Forward kinematics, the values of all the joint

Variables (angles) and length of each link to robot arm are known and based upon on it the movement of the robotic arm takes place.

The reverse mechanism is followed in inverse kinematics. In inverse kinematics, the value of the joint variables (angles) is to be calculated based on the coordinates(X, Y, Z) of the object and the length of links to robot arm to be reached by the arm as fig (1). In general, the problems involving inverse kinematics are far more complicated to solve compared to the problem of forward kinematics.

Forward kinematics have a unique solution defined for them, unlike the case of inverse kinematics. Inverse kinematics usually have multiple solutions for the same given input. In some cases, it might even not have a result defined. There also exits situations, where even though the solution exists, its unreachable by the arm, because the given coordinates fall outside the arms working space and that occur when the distance between the object and base of robot arm is bigger than the summation of all length of links.

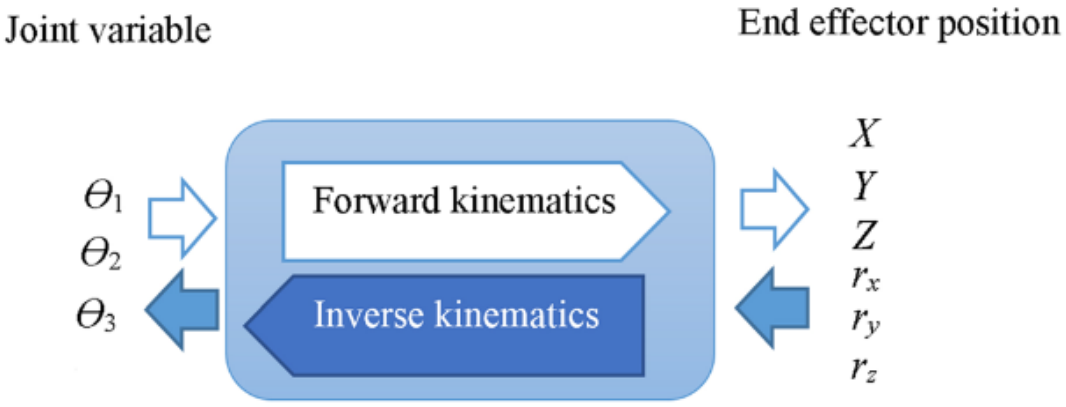


Figure 1. The relation between inverse and forward kinematics

The forward kinematics:

In this section, the main goal is to determine the collective effect of joint angles on the direction and position of the end effector in the arm robot. Due to the high accuracy the

Denawit–Hartenberg (DH) method is applied for the forward kinematics problem analysis.

On the D-H method first you must find the D-H parameters

Joint angle θi angle from X_{i-1} to X_i about Z_{i-1}

Link offset d_i distance from O_{i-1} to X_i along Z_{i-1}

Link length a_i distance between Z_{i-1} and Z_i along X_i

Link twist α_i angle from Z_{i-1} and Z_i along X_i

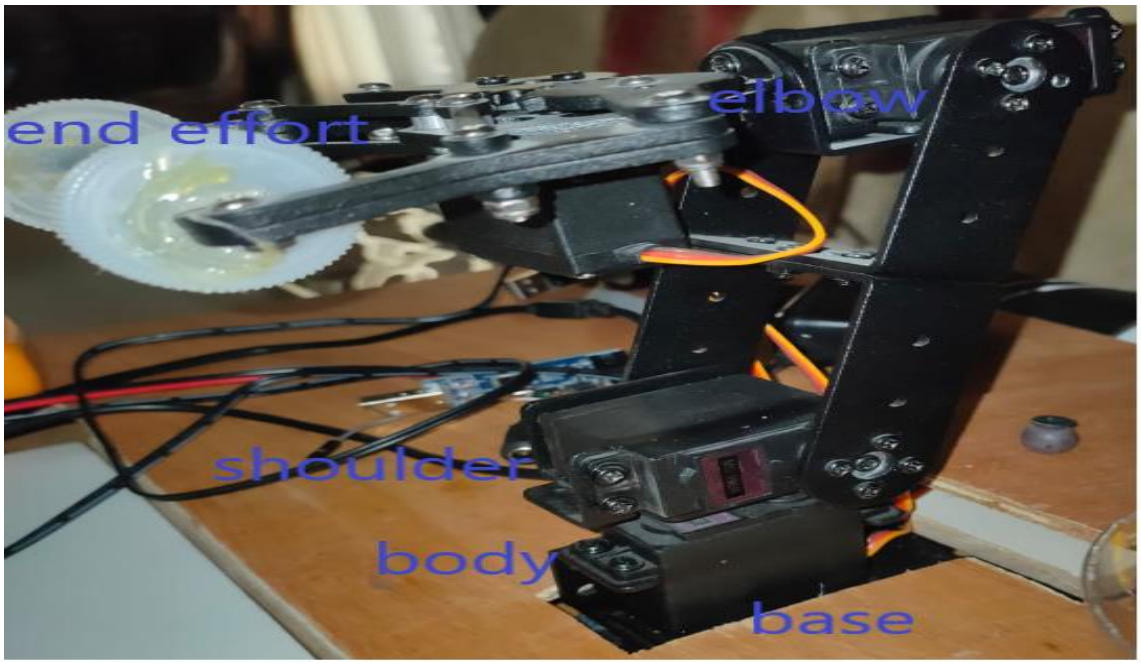


Figure 2: the description of robot arm.

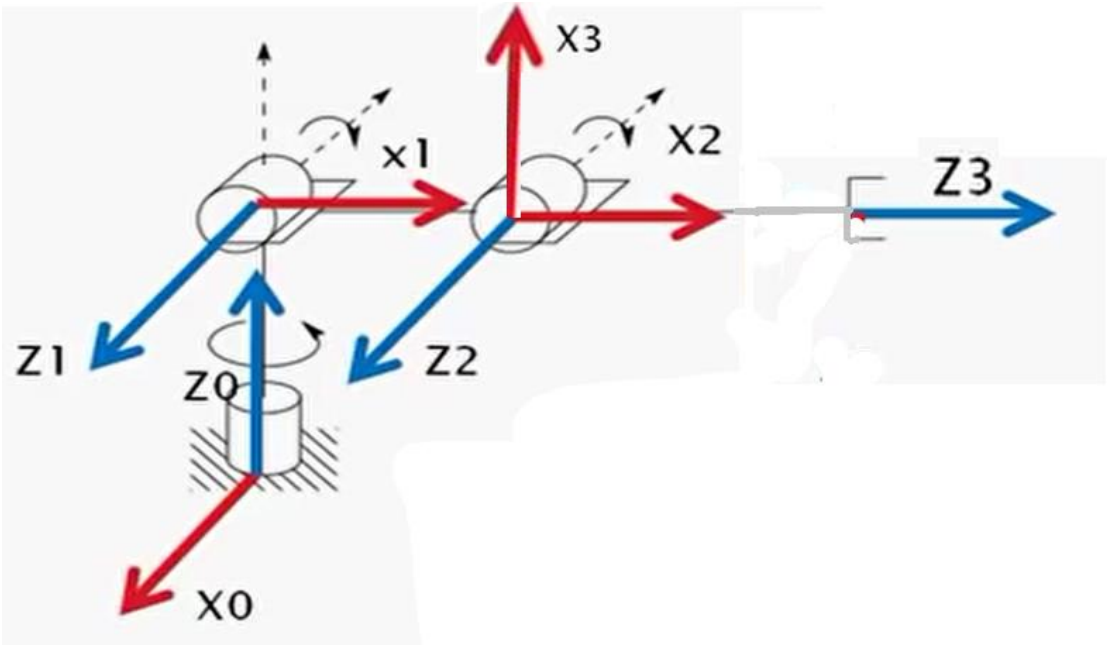


Figure 3: the axis definition of RRR robot arm.

From figure (3) we will get the D-H table.

link	a_i	alpha	Di	theta
1	0	Pi/2	L1	Th1
2	L2	0	0	Th2
3	L3	Pi/2	0	Th3

After getting D-H table then we will simulate the robot.

The code:

```
% forward kinematics for robot arm
% in this model i will use DH parameters with using peter
% robotics tool box to plot the robot
startup_rvc   % to call the library
%% DH parameters by using peter robotics
L1=100; L2=117; L3=130;
%  L = Link([th d a alpha]);
L(1)=Link([0 L1 0 pi/2]);
L(2)=Link([0 0 L2 0]);
L(3)=Link([0 0 0  pi/2]);
L(4)=Link([0 0 L3  pi/2]);
Rob = SerialLink(L)
rob.name='agr-robot';
rob.plot([0 0 0  ])
rob.teach;
```

The output of simulation:

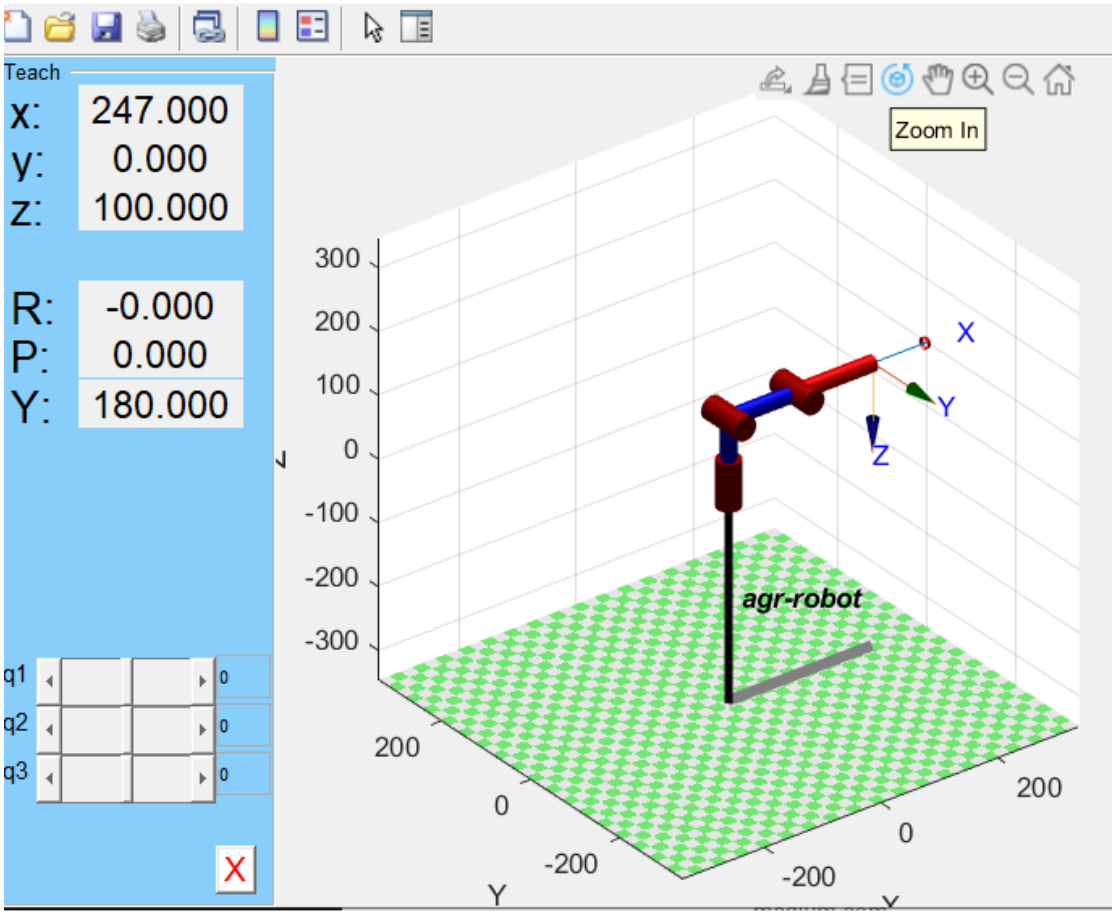


Figure 4 the output of simulation for forward kinematics.

The inverse kinematics:

Inverse Kinematics (IK) analysis determines the joint angles for desired position and orientation in Cartesian space.

IK is more difficult problem than forward kinematics.

The solution of inverse kinematic is more complex than direct kinematics.

There are several ways to solve the inverse kinematics problem.

1) Closed-form solutions

- algebraic method
- Geometric method

2) The numerical solutions

- the Jacobian transpose method
- the pseudoinverse method
- cyclic coordinate descent methods
- the Levenberg-Marquardt damped least squares methods
- quasi-Newton and conjugate gradient methods
- neural net and artificial intelligence methods
- the singular value decomposition

Closed-form solutions operate faster than numerical solutions and can be used to identify all possible variants.

From closed-form I will use the geometric method to solve the IK due to is more accurate.

In the inverse kinematics the coordinates of the object (X_o ,Y_o , Z_o) and the length of links are known.

The goal is getting the angles of robot arm (Th1, Th2, and Th3)

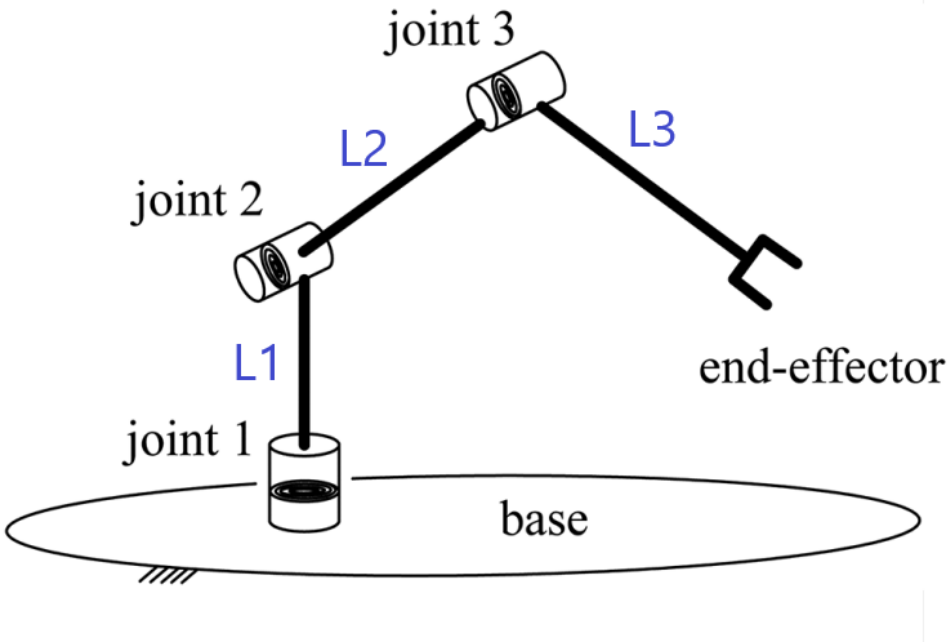


Figure 5 3-DOF robot arm with end effort.

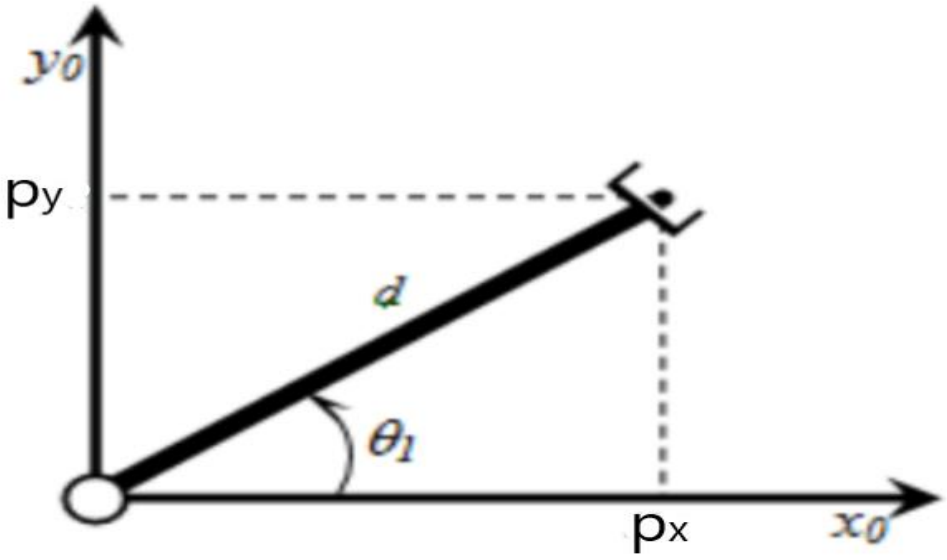


Figure 6 top view of robot arm.

From figure 6

$$\tan(\theta_1) = \frac{Yd}{Xd}$$

$$\therefore \theta_1 = \tan^{-1}\frac{Yd}{Xd}$$

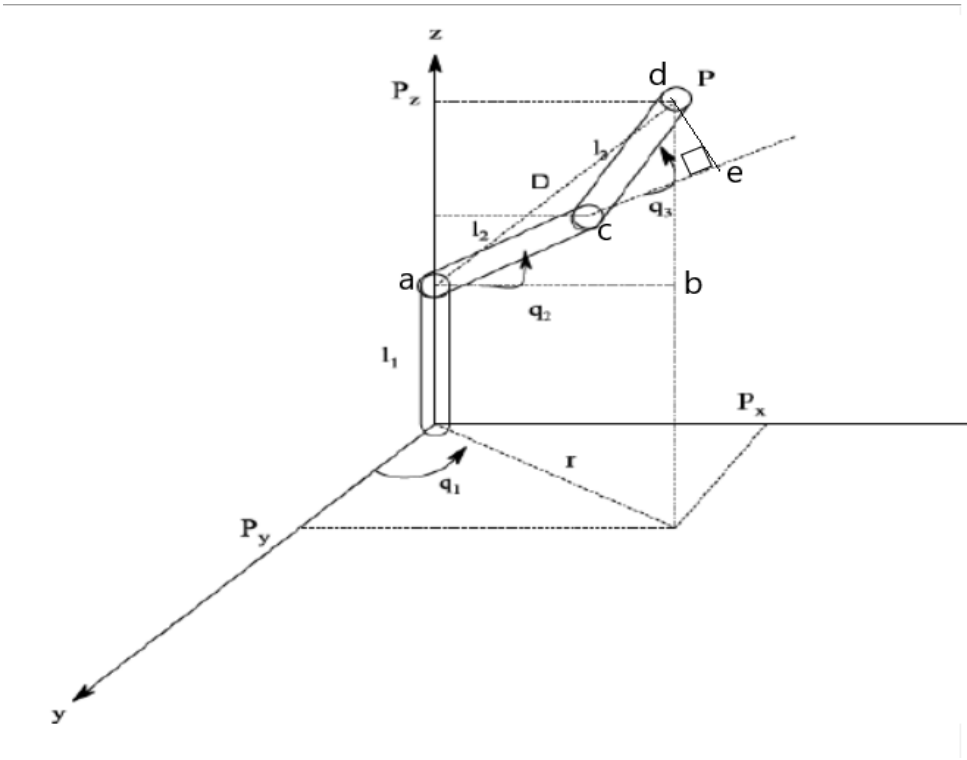


Figure 7. 3DOF articulated manipulator in Spherical coordinates for inverse kinematic analysis.

From figure 7.

$$r = \sqrt{p_x^2 + p_y^2}\,2$$
$$db = p_z - L_1$$
$$D = \sqrt{r^2 + db^2}$$
$$on \quad \Delta acd$$
$$D^2 = L_2^2 + L_3^2 - 2L_2L_3\cos(180 - \theta_3)$$
$$\cos(\theta_3) = \frac{D^2 - L_2^2 - L_3^2}{2L_2L_3}$$
$$\therefore \theta_3 = \cos^{-1}\left(\frac{D^2 - L_2^2 - L_3^2}{2L_2L_3}\right)$$

$$on \quad \Delta abd \quad let(\hat{dab}) = \beta$$
$$\tan(\beta) = \frac{p_z - L1}{r}$$
$$\therefore \beta = \tan^{-1}\left(\frac{p_z - L1}{r}\right)$$

$$on \quad \Delta abd \quad let(\hat{dae}) = \alpha$$
$$pe = L3\sin(\theta_3)$$
$$\sin(\alpha) = \frac{pe}{D}$$
$$\alpha = \sin^{-1}\left(\frac{pe}{D}\right)$$
$$\theta_2 = \beta - \alpha$$

The code on MATLAB:

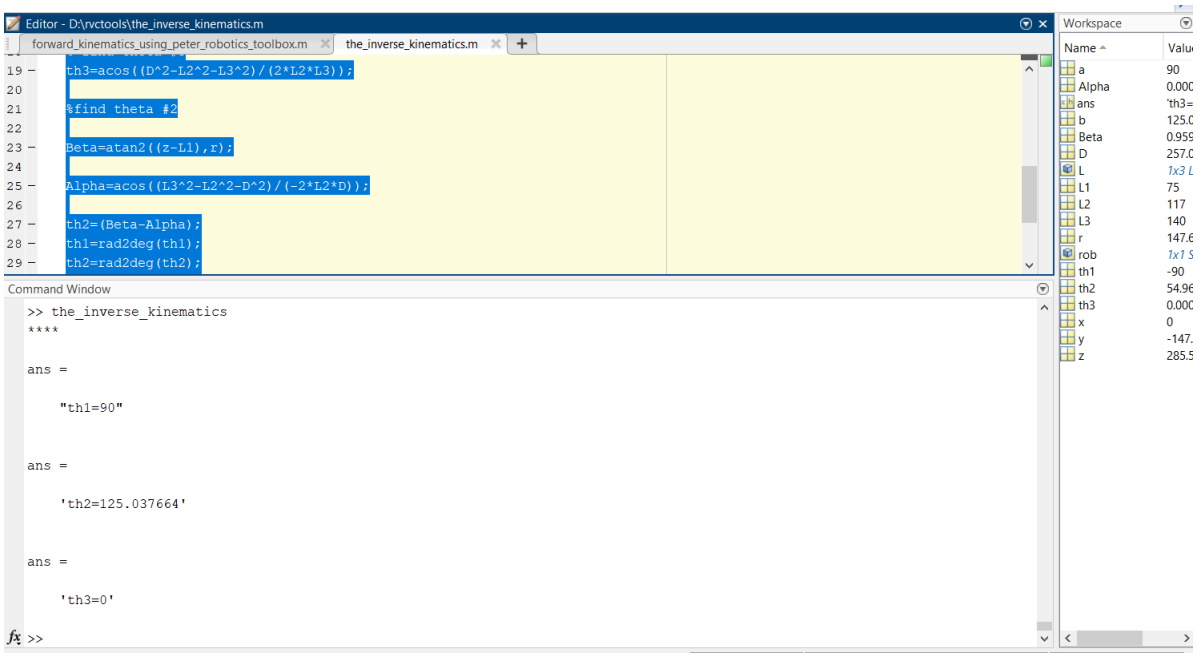
```
%% the inverse kinematics

% the x,y,z in mm
x=0;
y=-147.6;
z=285.5;
```

```
% length of links for robot arm
L1=75;
L2=117;
L3=140;
r=sqrt(x^2+y^2);
D=sqrt(r^2+(z-L1)^2);
% find theta #1
th1=atan2(y,x);
% find theta #3
th3=acos((D^2-L2^2-L3^2)/(2*L2*L3));
%find theta #2
Beta=atan2((z-L1),r);
Alpha=acos((L3^2-L2^2-D^2)/(-2*L2*D));
th2=(Beta-Alpha);
th1=rad2deg(th1);
th2=rad2deg(th2);
th3=rad2deg(th3);
disp('***********')
a=180+th1;
sprintf("th1=%d",a)
b=180-real(th2);
sprintf('th2=%f',b)
sprintf('th3=%d',th3)
```

The output:

Form the output this is one position or the first position.



Chapter 8:

SYSTEM GUI

GUI in Python language:

- Tkinter ("Tk Interface") is python's standard cross-platform package for creating graphical user interfaces (GUIs).
- Tkinter was written by 'Steen Lumholt' and 'Guido van Rossum', then later revised by 'Fredrik Lundh'.
- Tkinter is free software released under a Python license.
- Tkinter is included with standard Linux, Microsoft Windows and macOS installs of Python.
- The name Tkinter comes from the Tk GUI toolkit.

- Tkinter provides access to an underlying Tcl interpreter with the Tk toolkit, which itself is a cross-platform, multilanguage graphical user interface library.
- The `tkinter` package is a thin object-oriented layer on top of Tcl/Tk, It is implemented as a Python wrapper around complete Tcl interpreter embedded in the Python interpreter .
- To use tkinter, you don't need to write Tcl code, As Tkinter calls are translated into Tcl commands, which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

- Tkinter isn't the only GUI library for python, but it is the one that comes standard. Additional GUI libraries that can be used with python include: wxPython, PyQt, PySide, Pygame, Pyglet, PyGTK, and kivy.
- Tkinter's greatest strength is its ubiquity and simplicity. It works out of the box on most platforms (linux, OSX, Windows), and comes complete with a wide range of widgets necessary for most common tasks (buttons, labels, drawing canvas, multiline text, etc). As a learning tool, tkinter has some features that are unique among GUI toolkits, such as named fonts, bind tags, and variable tracing.

tkinter — Python interface to Tcl/Tk

Running python -m tkinter from the command line should open a window demonstrating a simple Tk interface, letting you know that tkinter is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

Tkinter supports a range of Tcl/Tk versions, built either with or without thread support. The official Python binary release bundles Tcl/Tk 8.6 threaded. See the source code for the _tkinter module for more information about supported versions.

Tkinter is not a thin wrapper, but adds a fair amount of its own logic to make the experience more pythonic. This documentation will concentrate on these additions and changes, and refer to the official Tcl/Tk documentation for details that are unchanged.

Architecture:

Tcl/Tk is not a single library but rather consists of a few distinct modules, each with separate functionality and its own official documentation. Python's binary releases also ship an add-on module together with it.

- Tcl:

  Tcl is a dynamic interpreted programming language, just like Python. Though it can be used on its own as a general-purpose programming language, it is most commonly embedded into C applications as a scripting engine or an interface to the Tk toolkit. The Tcl library has a C interface to create and manage one or more instances of a Tcl interpreter, run Tcl commands and scripts in those instances, and add custom commands implemented in either Tcl or C. Each interpreter has an event queue, and there are facilities to send events to it and process them. Unlike Python, Tcl's execution model is designed around cooperative multitasking, and Tkinter bridges this difference (see Threading model for details).

- Tk:

  Tk is a Tcl package implemented in C that adds custom commands to create and manipulate GUI widgets. Each Tk object embeds its own Tcl interpreter instance with Tk loaded into it. Tk's widgets are very customizable, though at the cost of a dated appearance. Tk uses Tcl's event queue to generate and process GUI events.

- Ttk:

> Themed Tk (Ttk) is a newer family of Tk widgets that provide a much better appearance on different platforms than many of the classic Tk widgets. Ttk is distributed as part of Tk, starting with Tk version 8.5. Python bindings are provided in a separate module, `tkinter.ttk`.

Internally, Tk and Ttk use facilities of the underlying operating system, i.e., Xlib on Unix/X11, Cocoa on macOS, GDI on Windows.

When your Python application uses a class in Tkinter, e.g., to create a widget, the `tkinter` module first assembles a Tcl/Tk command string. It passes that Tcl command string to an internal _tkinter binary module, which then calls the Tcl interpreter to evaluate it. The Tcl interpreter will then call into the Tk and/or Ttk packages, which will in turn make calls to Xlib, Cocoa, or GDI.

Tkinter Modules

Support for Tkinter is spread across several modules. Most applications will need the main `tkinter` module, as well as the `tkinter.ttk` module, which provides the modern themed widget set and API:

```
from tkinter import *
from tkinter import ttk
```

Differences of Tkinter in python 2 and python 3:

Tkinter is largely unchanged between python 2 and python 3, with the major difference being that the tkinter package and modules were renamed.

- Importing in python 2.x:

In python 2.x, the tkinter package is named Tkinter, and related packages have their own names. For example, the following shows a typical set of import statements for python 2.x:

```
import Tkinter as tk
import tkFileDialog as filedialog
import ttk
```

- Importing in python 3.x:

Although functionality did not change much between python 2 and 3, the names of all of the tkinter modules have changed. The following is a typical set of import statements for python 3.x:

```
import tkinter as tk
from tkinter import filedialog
from tkinter import ttk
```

The modules that provide Tk support include:

- `tkinter`: Main Tkinter module.
- `tkinter.colorchooser`: Dialog to let the user choose a color.
- `tkinter.commondialog`: Base class for the dialogs defined in the other modules listed here.
- `tkinter.filedialog`: Common dialogs to allow the user to specify a file to open or save.
- `tkinter.font`: Utilities to help work with fonts.
- `tkinter.messagebox`: Access to standard Tk dialog boxes.
- `tkinter.scrolledtext`: Text widget with a vertical scroll bar built in.

- `tkinter.simpledialog`: Basic dialogs and convenience functions.
- `tkinter.ttk`: Themed widget set introduced in Tk 8.5, providing modern alternatives for many of the classic widgets in the main `tkinter` module.

Additional modules:

- _tkinter:

  A binary module that contains the low-level interface to Tcl/Tk. It is automatically imported by the main `tkinter` module, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

- Idlelib: Python's Integrated Development and Learning Environment (IDLE). Based on `tkinter`.
- tkinter.constants: Symbolic constants that can be used in place of strings when passing various parameters to Tkinter calls. Automatically imported by the main `tkinter` module.
- `tkinter.dnd`: (experimental) Drag-and-drop support for `tkinter`. This will become deprecated when it is replaced with the Tk DND.

- `tkinter.tix`: (deprecated) An older third-party Tcl/Tk package that adds several new widgets. Better alternatives for most can be found in `tkinter.ttk`.
- `turtle`: Turtle graphics in a Tk window.

Tkinter Life Preserver:

This section is not designed to be an exhaustive tutorial on either Tk or Tkinter. For that, refer to one of the external resources noted earlier. Instead, this section provides a very quick orientation to what a Tkinter application looks like, identifies foundational Tk concepts, and explains how the Tkinter wrapper is structured.

The remainder of this section will help you to identify the classes, methods, and options you'll need in your Tkinter application, and where to find more detailed documentation on them, including in the official Tcl/Tk reference manual.

How Tkinter Wraps Tcl/Tk:

When your application uses Tkinter's classes and methods, internally Tkinter is assembling strings representing Tcl/Tk commands, and executing those commands in the Tcl interpreter attached to your applicaton's `Tk` instance.

Whether it's trying to navigate reference documentation, trying to find the right method or option, adapting some existing code, or debugging your Tkinter application, there are times that it will be useful to understand what those underlying Tcl/Tk commands look like.

To illustrate, here is the Tcl/Tk equivalent of the main part of the Tkinter script above.

```
ttk::frame .frm -padding 10
grid .frm
grid [ttk::label .frm.lbl -text "Hello World!"] -column 0 -row
0
grid [ttk::button .frm.btn -text "Quit" -command "destroy ."]
-column 1 -row 0
```

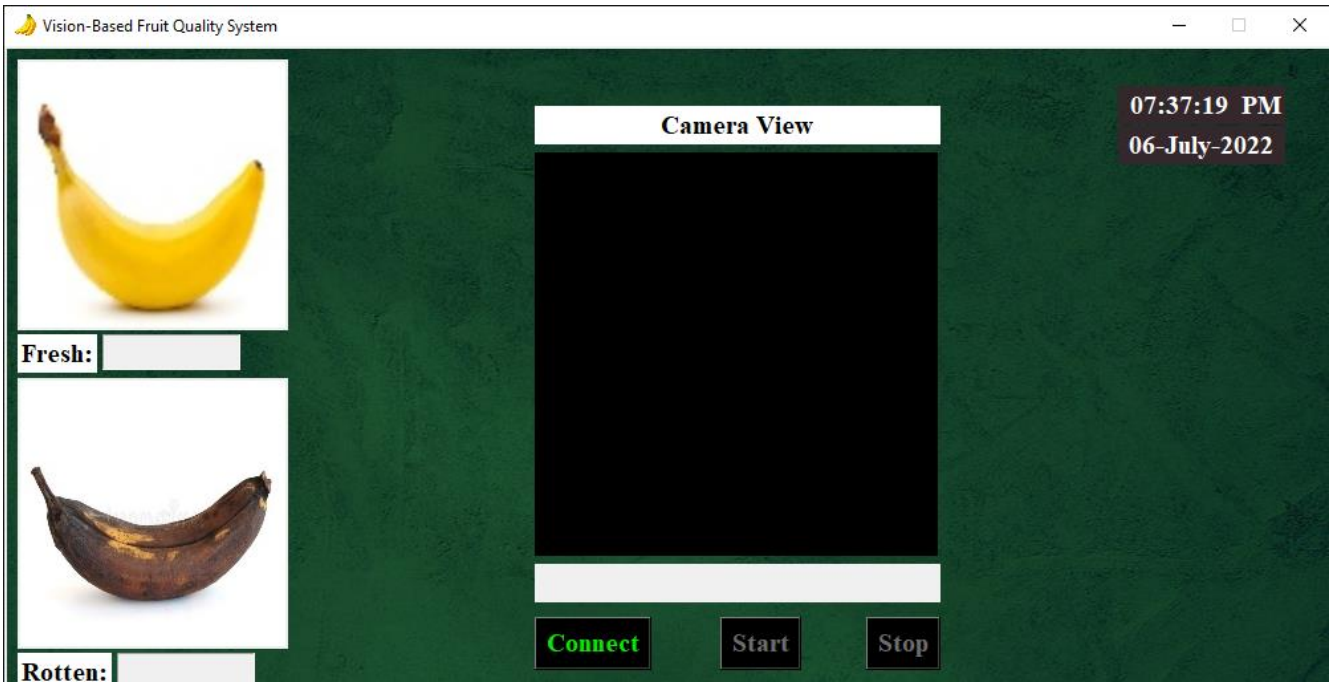
Tcl's syntax is similar to many shell languages, where the first word is the command to be executed, with arguments to that command following it, separated by spaces. Without getting into too many details, notice the following:

- The commands used to create widgets (like `ttk::frame`) correspond to widget classes in Tkinter.
- Tcl widget options (like `-text`) correspond to keyword arguments in Tkinter.
- Widgets are referred to by a *pathname* in Tcl (like `.frm.btn`), whereas Tkinter doesn't use names but object references.

- A widget's place in the widget hierarchy is encoded in its (hierarchical) pathname, which uses a `.` (dot) as a path separator. The pathname for the root window is just `.` (dot). In Tkinter, the hierarchy is defined not by pathname but by specifying the parent widget when creating each child widget.
- Operations which are implemented as separate *commands* in Tcl (like `grid` or `destroy`) are represented as *methods* on Tkinter widget objects. As you'll see shortly, at other times Tcl uses what appear to be method calls on widget objects, which more closely mirror what would is used in Tkinter.

System GUI

- **Design**:



- **Description:**
  - Use connect button to initialize the system and open the serial port to communicate with the arm system.
  - Once connection is finished, the start, and stop button enabled to control the system.
  - Start button is used to run the system main process to define the fruit quality based on computer vision technology.
  - Main function sequence is:
    - Capture a fruit image.
    - Display the image in the camera view window.
    - Use deep-learning to detect the fruit state.
    - Display the state under the camera view.
    - Send the fruit state to the arm system serially to put it in the right place.
- Increase the entries based on the fruit state to keep track the results.
- Stop button is used to close the port and connection to end the system process.

GUI importance:

- Provide the ability to monitor the system flow.
- make it easy to nonprofessional users to use the system.

Chapter 9:

APPENDICES

Abbreviations:

Abbreviations	Description
ARFQS	Automated Recognition Fruit Quality System
MCU	Microcontroller Unit
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
UART	Universal Asynchronous Receiver/Transmitter
ML	Machine Earning
IR	Infrared
DOF	Degree Of Freedom
PWM	Pulse Width Modulation
DC	Direct Current
GND	Ground
POT	Potentiometer (Variable Resistor)
TX	Transmitter
RX	Receiver
LED	Light Emitting Diode
DIO	Digital Input Output
MUX	Multiplexer
DEMUX	Demultiplexer
USB	Universal Serial Bus
TTL	Transistor Transistor Logic
DTR	Data Terminal Ready
INT2	Interrupt 2

REFERENCES

1. https://subscription.packtpub.com/book/data/9781789537147/1/ch01lvl1sec09/object-detection-using-color-in-hsv
2. https://www.theseus.fi/bitstream/handle/10024/452552/Do_Thuan.pdf?sequence=2&isAllowed=y
3. https://viso.ai/deep-learning/object-detection/
4. https://www.fritz.ai/object-detection/#part-basics
5. https://www.madrasresearch.org/post/step-by-step-yolo-algorithm
6. https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/
7. https://www.visiongeek.io/2019/10/preparing-custom-dataset-for-training-yolo-object-detector.html
8. file:///D:/download/12254-35180-1-PB.pdf
9. https://docs.python.org/3/library/tkinter.html
10. https://datasheetspdf.com/pdf/942981/ETC/MG996R/1
11. https://datasheet.octopart.com/LM7805CT-Fairchild-datasheet-5337151.pdf
12. https://lastminuteengineers.com/stepper-motor-l298n-arduino-tutorial/

13. https://www.researchgate.net/publication/353117321

14. https://library.iugaza.edu.ps/thesis/92207.pdf

15. https://doi.org/10.1007/s40435-019-00558-1