PROJECT LAB REPORT

MT-368L MECHATRONICS SYSTEM DESIGN LAB

Group Members:

Name	Roll No
Faiq Nasir	201075
Waleed Tariq	201111
Anzala Tariq	201132
Muhammad Abubakar	201090
Muzammil Faisal	201180
Saud Mubashir	201147
Juhi Nehab	180909

Semester	Fall 2020
Section	B
Course	Mechatronics System Design Lab
Instructor	Dr. Zareena Kausar
Lab Engineer	Eng. Naqash Ahmed

Project Report

Design of Indigenous Robot

Chapter 1 Introduction

1.1 Objectives:

- To design an indigenous robot which is autonomous robots that can simulate Fruit Harvesting tasks and using mechatronics designing techniques.
- To design an optimal solution using the design procedure.
- To understand and implement various steps of the design process.

1.2 Introduction:

- **What is Indigenous Robot?**

  Indigenous robots that are constructed from scratch. Their mechanical structure, controls etc. are designed by themselves. The electronic control modules including all electronic boards and motor drivers (Unless specified otherwise) etc. There will be a robot that is acting as the harvester robot, its job is to pluck only ripped fruits from the tree. The ripe and unripe fruits are represented with orange and green colored ping pong balls respectively. After collecting the desired fruits, the robot shall drop them at the designated area and shall reach the parking spot.

- **Adopted Solution:**

  We will try to use mechatronics designing procedure to make a design for a robot harvesting robot, of food plucking fruit.

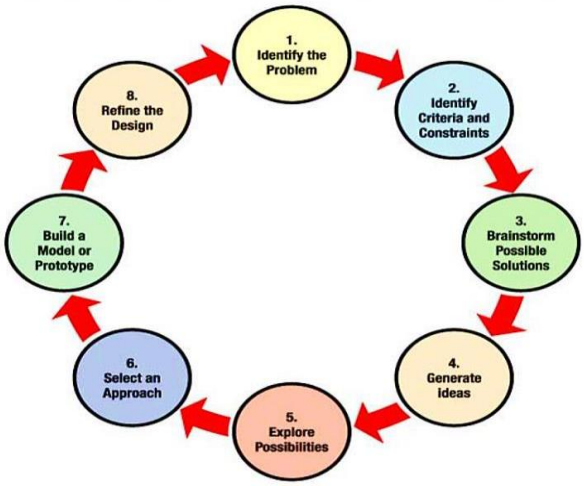


Figure 1 Steps followed during designing a mechatronic System

1.3 Methodology:

- **Identification of Problem:**
- There will be a robot that is acting as the harvester robot, its job is to pluck only riped fruits from the tree.
- The ripe and unripe fruits are represented with orange and green colored ping pong balls respectively.
- After collecting the desired fruits, the robot shall drop them at the designated area and shall reach the parking spot.

- ***Criteria and Constraints:***

- The maximum dimension of the robot is 10x10 inches (LxW).
- The Robot may follow the black line (Reflective Tape) or Wall of the arena to locate the Trees.
- The fruits are considered to be ping pong balls of green and orange colors.
- The entire arena is divided into 8x8 inch grids.
- The fruits are hanged with the branch of trees shown in Annex C.
- . The robot should be an autonomous and indigenously developed robot.
- The robot (R1) will reach the location of the tree and collect the desired ripe fruits from the tree and will put all the collected fruits in the unloading bay(B).
- Should be a mechatronics system
- Should be efficient

- **Brainstorming:**

We will divide the whole system into NERC theme map according to the diagram. Phase 1 will deal with there will be a robot that is acting as the harvester robot, its job is to pluck only ripped fruits from the tree. The ripe and unripe fruits are represented with orange and green colored ping pong balls respectively. After collecting the desired fruits, the robot shall drop them at the designated area and shall reach the parking spot. Robot complete the path can be done through a sensor and mechanical work. The entire arena is divided into 8x8 inch grids. Each grid is assigned a row and a column number. This grid is not represented as lines on the arena. It is for reference positions only, and grid lines number are not marked on fabricated arena. The robot (R1) will start at the intersection of grid lines (4, 5) and (9, 10) facing in the direction of the arrow as marked. The starting position and orientation of the robot (R1) are fixed. The robot must be placed behind the starting line. The red line on the figure adjacent on R1 shows the starting point of the Robot. The Robot will be placed behind the starting line whereas, wheels may be at different position. The line following sensor should also be behind the line. The Robot may follow the black line (Reflective Tape) or Wall of the arena to locate the Trees. The fruits are considered to be ping pong balls of green and orange colors. The fruits are hanged with the branch of trees shown in Annex C. The robot (R1) will reach the location of the tree and collect the desired ripe fruits from the tree and will put all the collected fruits in the unloading bay(B). After successful collection of all the ripe fruits and dropping them in unloading bay(B), the robot(R1) will move towards the Parking spot as marked in the arena. According to NERC Theme.

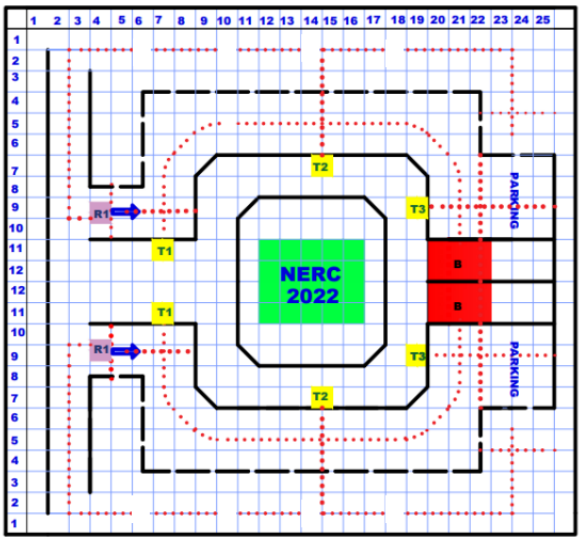
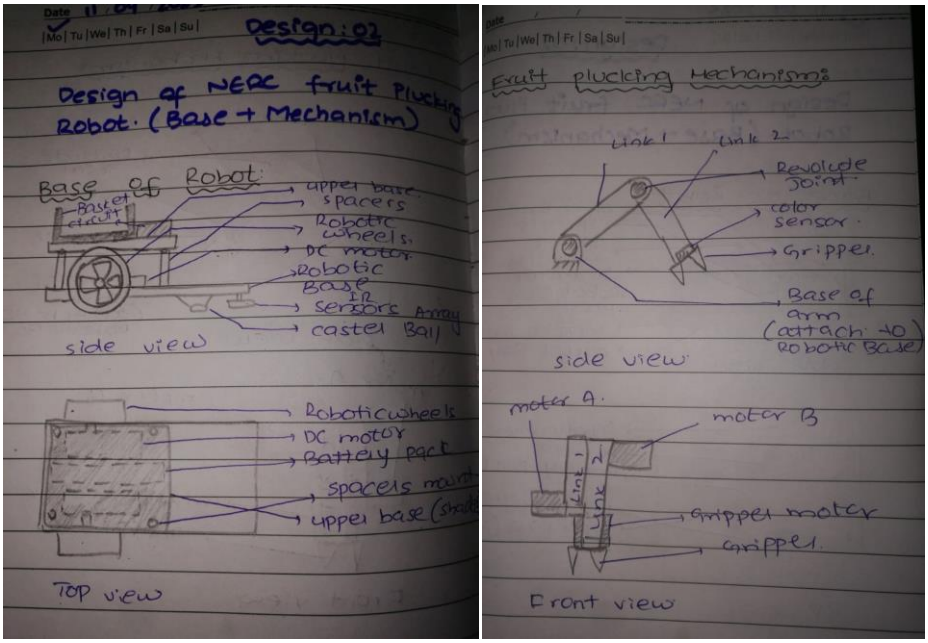


Figure 2

- **Generation of Ideas:**



- **Exploring Possibilities:**

➢ The robot fits within 10x 10-inch square at the time of measurement.

➢ There is no restriction on the maximum permissible height of the robot.

➢ All sensors mounted on the robot will be counted as part of the robot's total dimensions.

➢ Ultra-Sonic Range detectors (SONARs) or IR based proximity sensors (models specified in the components' list attached) must be used for sensing walls/Line

- **Selection of Approach:**

As we had only single idea that is optimal so we will only select that idea.

- **Build a model:**

Before building a full-scale model, we will build a prototype. On that prototype, all the analysis will be done. If our prototype fails to meet our requirements we will redesign.

1.4 Sustainable Development Goals:

This Robot can contribute to several of the United Nations Sustainable Development Goals (SDGs), including:

- **SDG 3: Good Health and Well-being:**

The Robot can contribute to better health and well-being by reducing wait times and improving the overall customer experience, leading to greater satisfaction and enjoyment of food service.

- **SDG 8: Decent Work and Economic Growth:**

The development and use of the Robot can create new job opportunities and stimulate economic growth in the food service industry.

- **SDG 9: Industry, Innovation and Infrastructure:**

The development of a robot involves cutting-edge technology and innovation, contributing to the advancement of industry and infrastructure.

- **SDG 12: Responsible Consumption and Production:**

By improving the efficiency and sustainability of food service delivery, the Robot can help reduce food waste and promote responsible consumption and production.

Chapter: 2

Select material, derive types (tire), sensor selection, battery, motors, micro controller and dimensions for robot

2.1 Introduction of Fruit Plucking Robot:

A fruit plucking robot is an autonomous machine designed to harvest fruits from trees. The robot typically uses various sensors and algorithms to detect and identify ripe fruits, then uses its robotic arms to pluck them from the tree without damaging them. The goal of the fruit plucking robot is to increase efficiency and reduce labor costs associated with manual fruit picking. Additionally, it can help address labor shortages and increase the quality of the fruits harvested by minimizing damage and bruising. Some fruit plucking robots also have the ability to sort the fruits based on size and ripeness, further streamlining the harvesting process. The development of fruit plucking robots is a promising area of agricultural technology with the potential to revolutionize the industry.

2.2 Material Selection:

When selecting materials for a university-level fruit plucking robot, several factors should be considered, including durability, weight, strength, cost, and ease of machining. Here are some materials that could be considered:

Aluminum: Aluminum is lightweight and has high strength-to-weight ratio, making it an ideal choice for the robot's body and structural components. It is also relatively easy to machine, which can reduce production costs.

Stainless steel: Stainless steel is a strong and durable material that can withstand exposure to the elements, making it a good choice for components that will be exposed to moisture and humidity in orchards. It is also corrosion-resistant and can be easily cleaned.

Carbon fiber: Carbon fiber is lightweight, strong, and has a high stiffness-to-weight ratio. It is commonly used in high-performance applications and can provide excellent structural support for the robot's frame.

Polycarbonate: Polycarbonate is a lightweight, shatter-resistant material that can be used for the robot's protective covers and windows. It is also easy to work with and can be formed into complex shapes.

Nylon: Nylon is a versatile and durable material that can be used for gears, bearings, and other components that need to withstand high stress and friction. It is also lightweight and can be easily machined.

Overall, the material selection for a university-level fruit plucking robot should prioritize the balance between strength, durability, weight, and cost to ensure a reliable and efficient design.

2.3 Drive type:

The choice of tires for a fruit plucking robot will depend on several factors such as the terrain and orchard conditions. Here are some drive type options to consider:

Skid steer: Skid steer drive system is a popular choice for fruit plucking robots, especially for those that operate on uneven terrain. Skid steer systems consist of two independent sets of wheels or tracks that can be operated in different directions, allowing the robot to easily maneuver and turn within tight spaces.

Omni-directional: Omni-directional drive systems can also be used in fruit plucking robots. This type of drive system allows the robot to move in any direction, making it easy to navigate around obstacles in the orchard.

Tracked drive: Tracked drive systems are suitable for fruit plucking robots that operate on soft, uneven terrain or steep inclines. They provide excellent traction and stability, which can be important when operating on slopes or in wet conditions.

Wheeled drive*:* Wheeled drive systems are a simple and cost-effective option for fruit plucking robots that operate on relatively flat terrain. They can be equipped with specialized tires for improved traction, such as knobby tires for off-road use or slick tires for smooth surfaces.

The selection of drive type for a fruit plucking robot will depend on the specific requirements of the application. A combination of factors such as terrain, size of the orchard, type of fruit, and efficiency should be considered to determine the optimal drive system for the fruit plucking mechanism.

2.4 Sensor Selection:

When it comes to selecting sensors for a fruit plucking robot for university competitions, cost-effectiveness, reliability, and ease of use should be considered. Here are some sensor options to consider:

2.5 Battery and Motors:

When it comes to selecting batteries and motors for a fruit plucking robot for university competitions, several factors should be considered, such as the robot's weight, size, and power requirements. Here are some options to consider:

DC Motors: DC motors are affordable and easy to control, making them a popular choice for fruit plucking robots. They can provide the necessary torque to move the robot's arms and pluck the fruits. However, they may require additional gears to provide enough power.

Servo Motors: Servo motors are precise and easy to control, making them an excellent choice for the robot's arms and plucking mechanism. They can be more expensive than DC motors, but their accuracy and ease of use make them a good option for university competitions.

Stepper Motors: Stepper motors provide accurate and precise control over the robot's movements, making them a popular choice for robotics applications. They are suitable for applications that require precise positioning and control, such as fruit plucking robots

Chapter: 3
RGB Color sensor discussion and calibration

3.1 Introduction:

Thus, Arduino RGB Color Detector using TCS3200 Color Sensor is a wonderful project for students who want to have fun with too many colors. In this project, we will be "Interfacing TCS3200 Color Sensor with Arduino" for designing a simple color detector. Earlier we learned about APDS9960 Color Sensor.

Experimental Set-Up has been designed specifically for detecting the frequency of RGB color. The LCD panel directly displays the frequency of RGB color. A combination of all these colors gives a different color. There is various software available which directly converts the combination of RGB frequency into the desired color. The setup is absolutely self-contained and requires no other apparatus.

3.2 Materials:

For designing Arduino RGB Color Detector using TCS3200 Color Sensor we use the following components. Each and every component are described below further.

Figure 3

3.3 Color Sensor:

TCS3200 RGB Color Sensor for Arduino is a complete color detector, including a TAOS TCS3200 RGB sensor chip and 4 white LEDs. The TCS3200 can detect and measure a nearly limitless range of visible colors. Applications include test strip reading, sorting by color,

3.4 Specifications & Pin Details:

Pin	Name	Details
1.	S0	Output frequency scaling selection inputs.
2.	S1	Output frequency scaling selection inputs.
3.	OE	Enable for fo (active low).
4.	GND	Power supply ground.
5.	S3	Photodiode type selection inputs
6.	S2	Photodiode type selection inputs
7.	OUT	Output frequency (fo).
8.	V_{DD}	Supply Voltage

Figure 4

3.5 Working Explanation:

The TCS230 senses color light with the help of an 8 x 8 array of photodiodes. Then using a Current to-Frequency Converter the readings from the photodiodes are converted into a square wave with a frequency directly proportional to the light intensity. Finally, using the Arduino Board we can read the square wave output and get the results for the color.

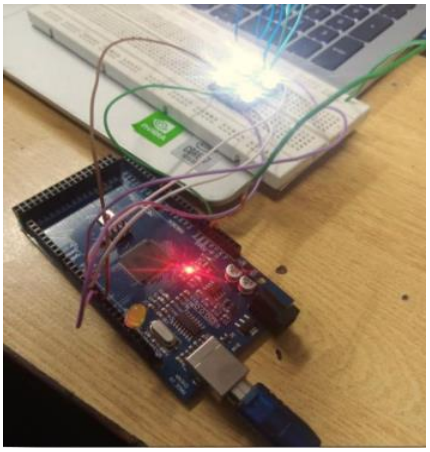
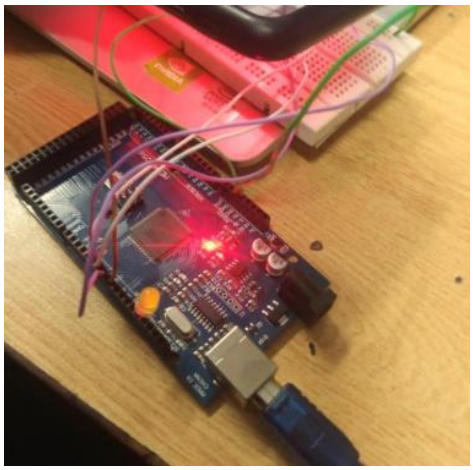


Figure 5



3.6 Results:

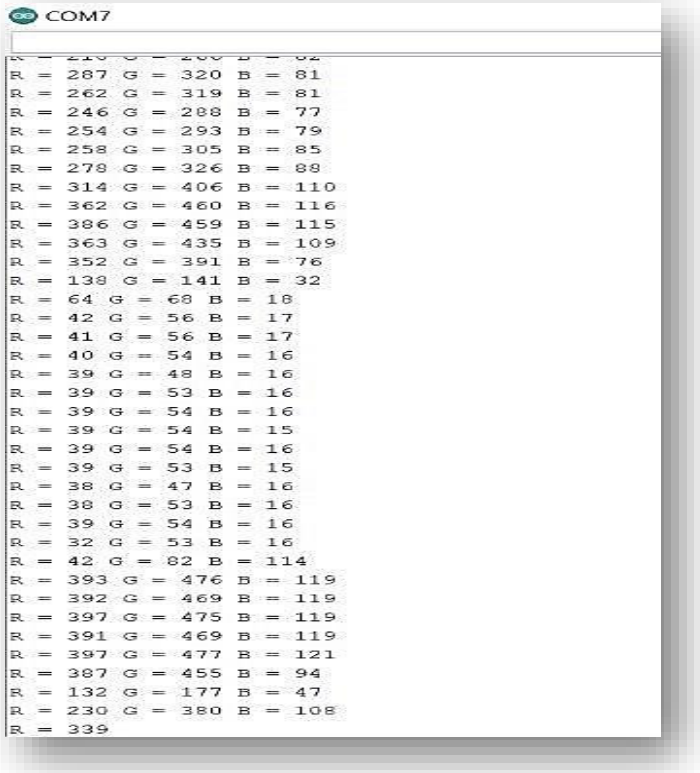


Figure 6

Chapter 4
Solid works of Autonomous Robot

4.1 Introduction:

Indigenous robots that are constructed from scratch. Their mechanical structure, controls etc. are designed by themselves. The electronic control modules including all electronic boards and motor drivers (Unless specified otherwise) etc. There will be a robot that is acting as the harvester robot, its job is to pluck only ripped fruits from the tree. The ripe and unripe fruits are represented with orange and green colored ping pong balls respectively. After collecting the desired fruits, the robot shall drop them at the designated area and shall reach the parking spot.

4.2 CAD Model:

For designing the robot on SW by using the components. as show in figure. Steels sheet for the base. We have made this model on solid works.

- **Arduino Uno**
- **Color sensor**
- **Servo motor**
- **Motor driver L298**
- **Wheels**
- **Motor Drives**

4.3 Arduino Uno:

The Arduino UNO is categorized as a microcontroller that uses the ATmega328 as a controller in it. The Arduino UNO board is used for an electronics project and mostly preferred by the beginners.

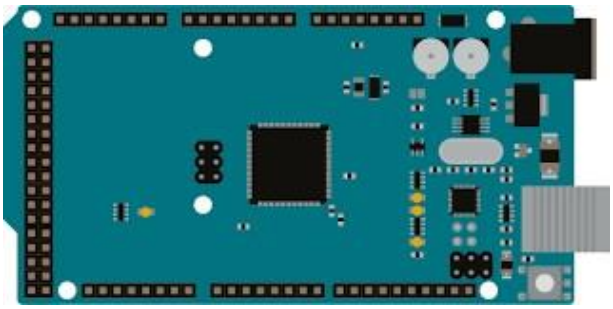


Figure 7

4.4 Color Sensor:

TCS3200 RGB Color Sensor for Arduino is a complete color detector, including a TAOS TCS3200 RGB sensor chip and 4 white LEDs. The TCS3200 can detect and measure a nearly limitless range of visible colors.

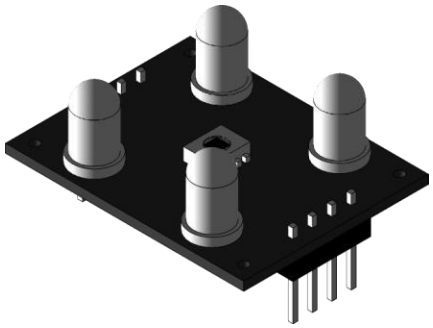


Figure 8

4.5 Motor Driver L298:

This **L298N Motor Driver Module** is a high-power motor driver module for driving DC and Stepper Motors. This module consists of an L298 motor driver IC and a 78M05 5V regulator. **L298N Module** can control up to 4 DC motors, or 2 DC motors with directional and speed control.

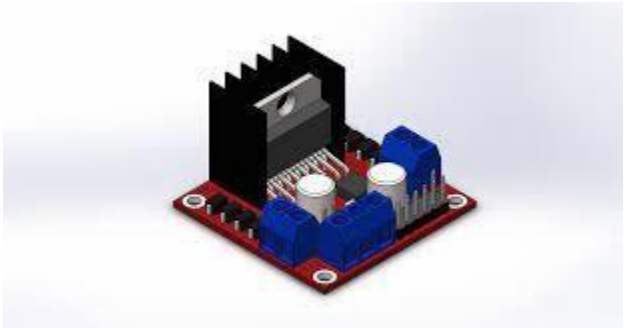


Figure 9

4.6 Robotic Wheel:

Wheeled robots are robots that navigate around the ground using motorized wheels to properly themselves.

Figure 10

4.7 Development on Solid works:

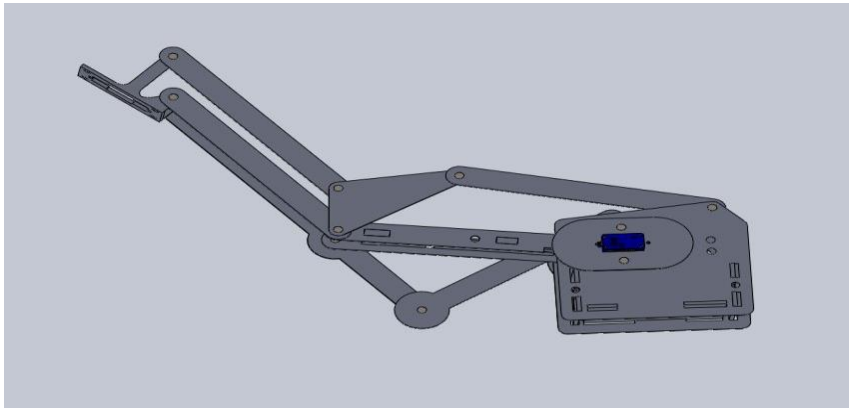


Figure 11

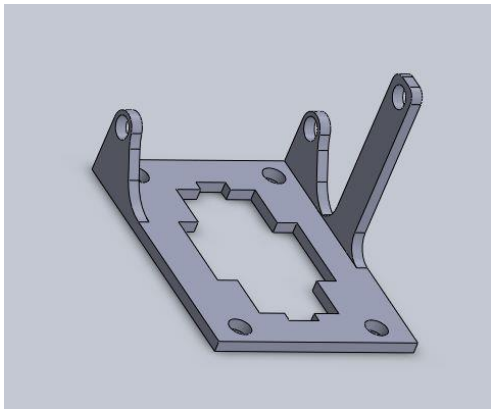


Figure 12

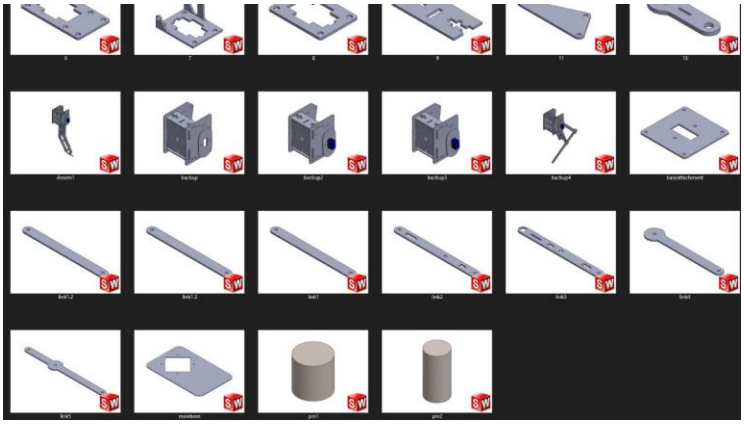


Figure 13

4.8 Complete Model:

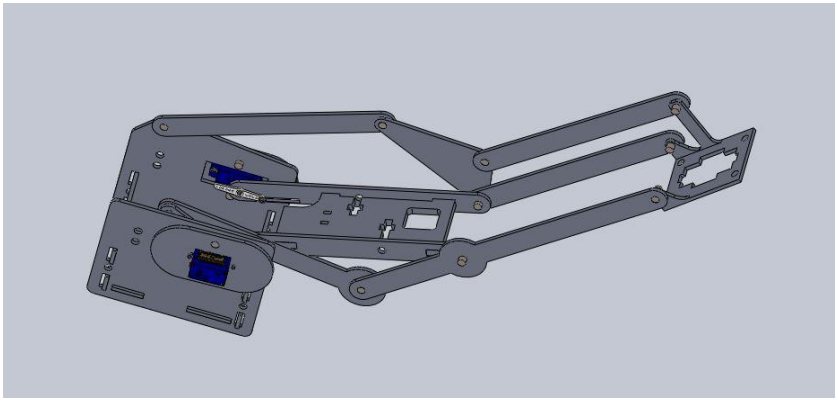


Figure 14

Chapter: 5
Motor Driver Module L298N

5.1 Introduction:

- **What is L298 Motor Driver?**

A motor driver module is a simple circuit used for controlling a DC motor. It is commonly used in autonomous robots and RC cars (L298N and L293D are the most regularly utilized motor driver chips). A motor driver module takes the low voltage input from a controller like Arduino. This input logic controls the direction of DC motors connected to the driver. To put it in simple words, you can control the direction of DC motors by giving appropriate logic to the motor driver module. The motor driver module consists of a motor driver IC, which is the heart of the module. The IC alone can control the DC motor, but using the module makes interfacing with Arduino easy.

5.2 Motor Driver:

This **L298N Motor Driver Module** is a high-power motor driver module for driving DC and Stepper Motors. This module consists of an L298 motor driver IC and a 78M05 5V regulator. **L298N Module** can control up to 4 DC motors, or 2 DC motors with directional and speed control.

Figure 15

5.3 Proteus Diagram:

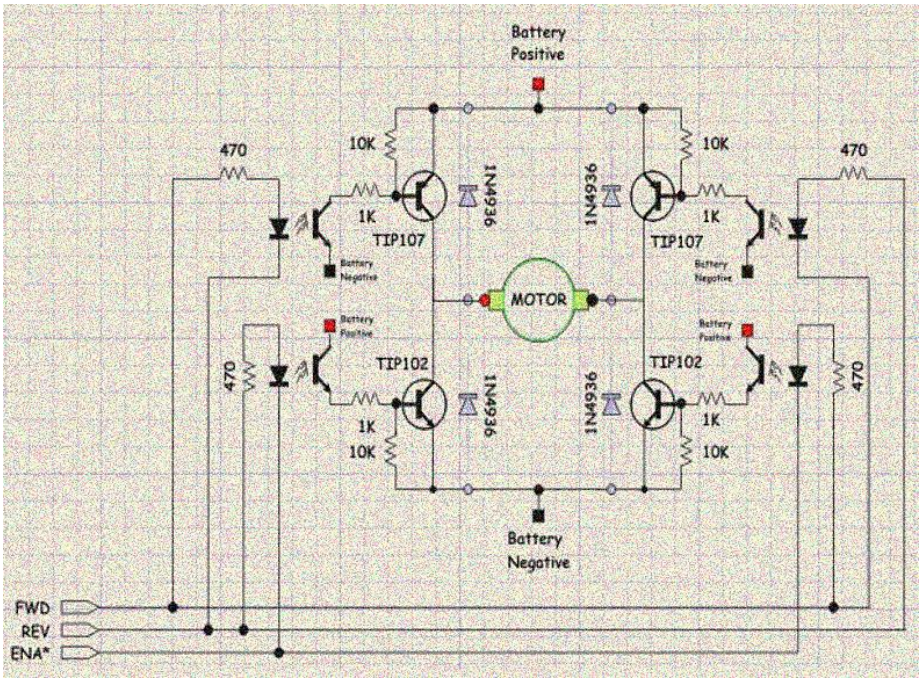


Figure 16

5.4 Proteus Diagram:

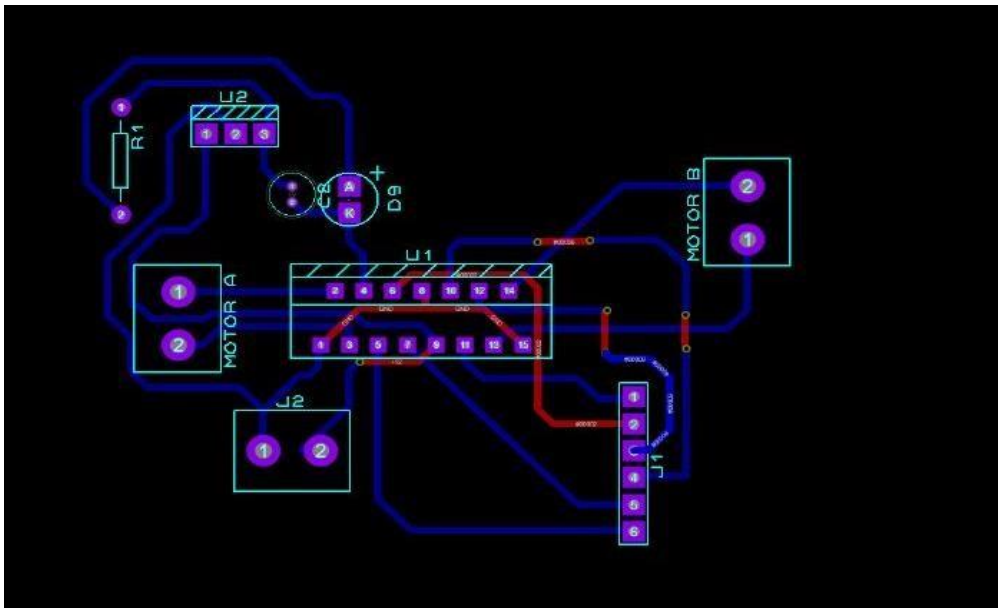


Figure 17

Chapter 6
Microcontroller Code for RGB Color sensor

6.1 Coding with Arduino:

```
#define S0 4
#define S1 5
#define S2 6
#define S3 7
#define sensorOut 8

int frequency = 0;
int R=0;
int B=0;
int G=0;

void setup() {
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(sensorOut, INPUT);

  // Setting frequency-scaling to 20%
  digitalWrite(S0,HIGH);
  digitalWrite(S1,LOW);

  Serial.begin(9600);
}
```

```
void loop() {
  // Setting R filt R photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,LOW);
  // Reading the output frequency
  R = pulseIn(sensorOut, LOW);
  // Printing the value on the serial monitor
  Serial.print("R= ");//printing name
  Serial.print (R);//printing R color frequency
  Serial.print("  ");
  delay(100);

  // Setting Green filt R photodiodes to be read
  digitalWrite(S2,HIGH);
  digitalWrite(S3,HIGH);
  // Reading the output frequency
  G = pulseIn(sensorOut, LOW);
  // Printing the value on the serial monitor
  Serial.print("G= ");//printing name
  Serial.print(G);//printing R color frequency
  Serial.print("  ");
  delay(100);

  // Setting B filt R photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,HIGH);
  // Reading the output frequency
  B = pulseIn(sensorOut, LOW);
  // Printing the value on the serial monitor
  Serial.print("B= ");//printing name
  Serial.print(B);//printing R color frequency
```

```
Serial.println("  ");
delay(100);




if(R<=212 && G<=181 && B<=210)
{
}


}
```

6.2 Hardware Design:

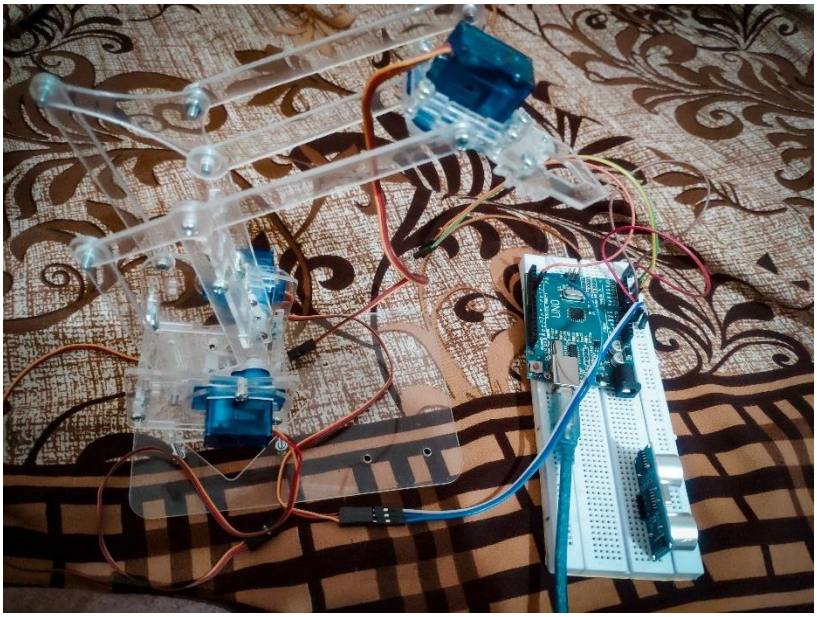


Figure 18

Chapter: 7

Design of Autonomous Robot on Hardware

7.1 Introduction:

Autonomous robots that can simulate Fruit Harvesting tasks. In this theme, there will be a robot that is acting as the harvester robot, its job is to pluck only ripped fruits from the tree. The ripe and unripe fruits are represented with orange and green colored ping pong balls respectively. After collecting the desired fruits, the robot shall drop them at the designated area and shall reach the parking spot. The first team to successfully collect all the desired fruits and reach the parking spot will be declared the winner.

7.2 Materials:

- **Steels sheet for the base.**
- **Arduino Uno**
- **Color sensor**
- **Servo motor**
- **Motor driver L298**
- **Wheels**
- **Motor Drives**

7.3 Steels Sheet for the Base:

Robot arms are typically made of raw materials like steel, aluminum and cast iron. Some special robots, like those used in clean room applications, are made of titanium. Beginning at the base, these industrial robots are assembled of several components, including motors, cylinders, cables and bearings.

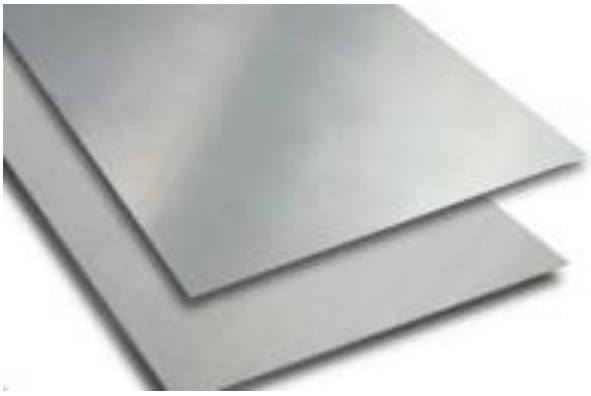


Figure 19

7.4 Arduino Uno:

The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc and initially released in 2010. The board is equipped with sets of digital and analog input/output pins that may be interfaced to various expansion boards and other circuits.

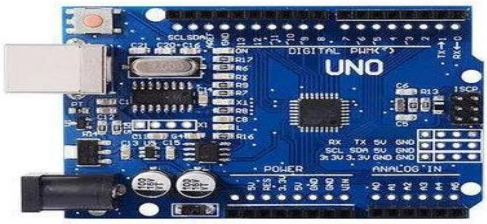


Figure 20

7.5 Color Sensor:

Figure 21

7.6 Motor Drives:

Motor driver IC is an integrated circuit chip which is usually used to control motors in autonomous robots. Motor driver ICs act as an interface between microprocessors in robots and the motors in the robot.

Figure 22

7.7 Servo Motor:

Servo motors are commonly used in applications such as robotics, CNC machines, and automation systems due to their high accuracy, speed and responsiveness.

Figure 23

7.8 Robotic Wheels:

Wheeled robots are robots that navigate around the ground using motorized wheels to properly themselves.

Figure 24

Chapter 8
Code Of Autonomous Robots for Running

8.1 Coding with Arduino of For Running Robot:

```
/*   Servo trims are chosen, so that 0°, 0°, 0° (left/right, back/front,
up/down) results in the neutral position of the robot arm.
   Neutral means: pivot direction forward, and both arms rectangular up and
forward
   Neutral position: X=0
                   Y=(LIFT_ARM_LENGTH_MILLIMETER + CLAW_LENGTH_MILLIMETER)
- Here claw length is from wrist to hand PLUS base center to shoulder
                    Z=HORIZONTAL_ARM_LENGTH_MILLIMETER


    Vertical plane schematic of inverse kinematic values
              LIFT_ARM
    _________/_____________________________
    | <-AngleHorizontalLift                /
    |                                     /
    |<-HORIZONTAL_ARM                    /
     |                                 /
     |                               /
     |                             /
      |                          /
      |                        /
      |   RadiusVertical->   /
       |                   /
       |                 /
       |               /
        |           /
        |         /
        | AngleClawHorizontal
         | |  /
         |  /
         |/ <-VerticalAngleToClaw
    ________________________________
*/

#include <Arduino.h>     // for PI etc.
#include "RobotArmKinematics.h"
#include <Servo.h>
```

```
#if __has_include("RobotArmServoConfiguration.h")
#include "RobotArmServoConfiguration.h"
#else
#define HORIZONTAL_ARM_LENGTH_MILLIMETER    80
#define LIFT_ARM_LENGTH_MILLIMETER          80
#define CLAW_LENGTH_MILLIMETER              68 // Length from wrist to hand
PLUS base center to shoulder
#define BALL_Height_MILLIMETER          80
#endif

const int S0= 4;
const int S1= 5;
const int S2= 6;
const int S3= 7;
const int sensorOut= 8;
const int trigPin=2;
const int echoPin=3;
const int DUmotor=9;
const int BFmotor=10;
const int CLAWmotor=11;
Servo DU, BF, Claw;
//#define LOCAL_TRACE

/**
   Get angle from a triangle using the cosine rule
   All sides are given
*/
bool  getAngleOfTriangle(float  aOppositeSide,  float  aAdjacentSide1,  float
aAdjacentSide2, float &aComputedAngle) {
  /*
     Cosine rule:
     C*C = A*A + B*B - 2*A*B*cos(angle_AB)
     C is opposite side
     A, B are adjacent sides
  */
  float tDenominator = 2 * aAdjacentSide1 * aAdjacentSide2; // (2*A*B)

  if (tDenominator == 0) { // @suppress("Direct float comparison")
    return false; //Singularity
  }
   float  tCosinusAB  =  (aAdjacentSide1  *  aAdjacentSide1  +  aAdjacentSide2  *
aAdjacentSide2 - aOppositeSide * aOppositeSide)
                      / tDenominator;

  if (tCosinusAB > 1 || tCosinusAB < -1) {
```

```
    return false;  //Not Correct value as range is between -1 --- 1
  }
  aComputedAngle = acos(tCosinusAB);

  return true;
}

/*
   Convert Cartesian to polar coordinates
   returns 0 to PI (0 to 180 degree)
   returns 0 to -PI (o to -180 degree) if aYValue < 0
*/
void cartesianToPolar(float aXValue, float aYValue, float &aRadius, float
&aAngleRadiant) {
  // Determine magnitude of Cartesian coordinates
  aRadius = sqrt(aXValue * aXValue + aYValue * aYValue);

  // Don't try to calculate zero-magnitude vectors' angles
  if (aRadius == 0) { // @suppress("Direct float comparison")
    aAngleRadiant = 0;
    return;
  }

  float tNormalizedXValue = aXValue / aRadius;

  // Calculate angle in 0..PI
  // use cos since for negative y values it needs only inverting the sign
of radiant
  aAngleRadiant = acos(tNormalizedXValue);

  // Convert to full range
  if (aYValue < 0) {
    aAngleRadiant *= -1;
  }

#if defined(LOCAL_TRACE)
  Serial.print(F("XValue="));
  Serial.print(aXValue);
  Serial.print(F(" YValue="));
  Serial.print(aYValue);
  Serial.print(F(" Radius="));
  Serial.print(aRadius);
  Serial.print(F(" AngleDegree="));
  Serial.println(aAngleRadiant * RAD_TO_DEG);
#endif
```

```
}

/**
   Inverse kinematics: X,Y,Z -> servo angle
   Reads the Cartesian positions from aPositionStruct and fills the angles
of aPositionStruct with the computed value
   @param aPositionStruct structure holding input position and output angles
   returns true if solving was successful, false if solving is not possible
*/
bool doInverseKinematics(struct ArmPosition *aPositionStruct) {
  bool tReturnValue = true;
  // Get horizontal degree for servo (0-180 degree) and get radius for next
computations
  float tRadiusHorizontalMillimeter, tHorizontalAngleRadiant;
  /*
        First   doInverseKinematics   the   horizontal   (X/Y)   plane.   Get
LeftRightDegree and radius from X and Y.
  */
   cartesianToPolar(aPositionStruct->LeftRight, aPositionStruct->BackFront,
tRadiusHorizontalMillimeter, tHorizontalAngleRadiant);
  aPositionStruct->LeftRightDegree = (tHorizontalAngleRadiant * RAD_TO_DEG)
- 90; // tLeftRightDegree: -90 to 90 degree, -90 is right, 0 degree is
neutral, 90 is left

  /*
     Now we have the base rotation for horizontal (X/Y) plane.
     Because we cannot move claw behind the base axis, we must check if the
radius in horizontal plane is smaller than the virtual claw length!
  */
  if (tRadiusHorizontalMillimeter < CLAW_LENGTH_MILLIMETER) {
    Serial.print(F("For "));
    printPositionCartesian(aPositionStruct);
    Serial.print(F(" horizontal radius "));
    Serial.print(tRadiusHorizontalMillimeter);
    Serial.println(F("mm is < claw length. -> Take claw length now."));
    tRadiusHorizontalMillimeter = 0; // fallback
    tReturnValue = false;
  } else {
    tRadiusHorizontalMillimeter -= CLAW_LENGTH_MILLIMETER;
  }

  /*
     Now doInverseKinematics the vertical plane
     Get vertical angle and radius/distance to claw - angle is negative for
claw below horizontal zero plane
```

```
  */
  float tVerticalAngleToClawRadiant, tRadiusVerticalMillimeter;
    cartesianToPolar(tRadiusHorizontalMillimeter,   aPositionStruct->DownUp,
tRadiusVerticalMillimeter, tVerticalAngleToClawRadiant);
#if defined(LOCAL_TRACE)
  Serial.print(F("RadiusHorizontal="));
  Serial.print(tRadiusHorizontalMillimeter);
  Serial.print(F(" VerticalAngleToClawDegee="));
  Serial.print(tVerticalAngleToClawRadiant * RAD_TO_DEG);
  Serial.print(F(" RadiusVertical="));
  Serial.print(tRadiusVerticalMillimeter);
#endif

  // Solve arm inner angles
   float tAngleClawHorizontalRadiant;       // angle between vertical angle
to claw and horizontal arm
             if            (!getAngleOfTriangle(LIFT_ARM_LENGTH_MILLIMETER,
HORIZONTAL_ARM_LENGTH_MILLIMETER, tRadiusVerticalMillimeter,
                           tAngleClawHorizontalRadiant)) {
    Serial.print(F("Cannot solve angle between claw base and horizontal arm
: "));
    printPositionCartesianWithLinefeed(aPositionStruct);
    return false;
  }
#if defined(LOCAL_TRACE)
  Serial.print(F(" AngleClawHorizontalDegree="));
  Serial.print(tAngleClawHorizontalRadiant * RAD_TO_DEG);
#endif

  float tAngleHorizontalLiftRadiant;  // angle between horizontal and lift
arm
             if            (!getAngleOfTriangle(tRadiusVerticalMillimeter,
HORIZONTAL_ARM_LENGTH_MILLIMETER, LIFT_ARM_LENGTH_MILLIMETER,
                           tAngleHorizontalLiftRadiant)) {
    Serial.print(F("Cannot solve angle between horizontal and lift arm: "));
    printPositionCartesianWithLinefeed(aPositionStruct);
    return false;
  }

  /*
     Vertical plane schematic of inverse kinematic values
               LIFT_ARM
     _________/_____________________________
     | <-AngleHorizontalLift                 /
     |                                      /
```

```
     |<-HORIZONTAL_ARM                  /
      |                               /
      |                             /
      |                           /
       |                        /
       |                      /
       |   RadiusVertical->  /
        |                  /
        |                /
        |              /
         |           /
         |         /
         | AngleClawHorizontal
          | |  /
          |  /
          |/ <-VerticalAngleToClaw
     ___________________________________
  */
  aPositionStruct->BackFrontDegree = (HALF_PI - (tVerticalAngleToClawRadiant
+ tAngleClawHorizontalRadiant)) * RAD_TO_DEG;
  // Now BackFrontDegree == 0 is vertical, front > 0, back < 0
#if defined(LOCAL_TRACE)
  Serial.print(F(" BackFrontDegree="));
  Serial.print(aPositionStruct->BackFrontDegree);
#endif

    aPositionStruct->DownUpDegree    =    (tVerticalAngleToClawRadiant    +
tAngleClawHorizontalRadiant + tAngleHorizontalLiftRadiant - PI)
                                  * RAD_TO_DEG;
  // Now DownUpDegree == 0 is horizontal, up > 0, down < 0
#if defined(LOCAL_TRACE)
  Serial.print(F(" DownUpDegree="));
  Serial.println(aPositionStruct->DownUpDegree);
#endif
  return tReturnValue;
}

/*
   Forward kinematics: servo angle -> X,Y,Z
*/
void polarToCartesian(float aRadius, float aAngleRadiant, float &aXValue,
float &aYValue) {
  aXValue = aRadius * cos(aAngleRadiant);
  aYValue = aRadius * sin(aAngleRadiant);
}
```

```
/**
   Fill X,Y,Z in aPositionStruct according to angles of aPositionStruct.
   always solvable :-)
*/
void doForwardKinematics(struct ArmPosition *aPositionStruct) {
  float tInputAngle;
      float     tHeightOfHorizontalArm,     tHorizontalArmHorizontalShift,
tLiftHorizontal, tLiftHeight, tInputAngleRad;

  // input angle: horizontal = 0 vertical = 90
  tInputAngle = aPositionStruct->BackFrontDegree;
  // here we have:  0 is vertical, front > 0, back < 0
  tInputAngle *= -1;
  // here we have:  0 is vertical, front < 0, back > 0
  tInputAngleRad = (tInputAngle + 90) * DEG_TO_RAD;

     polarToCartesian(HORIZONTAL_ARM_LENGTH_MILLIMETER,    tInputAngleRad,
tHorizontalArmHorizontalShift, tHeightOfHorizontalArm);

  // input angle: horizontal = 0, up > 0, down < 0
  tInputAngle = aPositionStruct->DownUpDegree;
  tInputAngleRad = tInputAngle * DEG_TO_RAD;
        polarToCartesian(LIFT_ARM_LENGTH_MILLIMETER,       tInputAngleRad,
tLiftHorizontal, tLiftHeight);
  aPositionStruct->DownUp = tHeightOfHorizontalArm + tLiftHeight;

  // Add horizontal length
   float  tHorizontalArmAndClawShift  =  tHorizontalArmHorizontalShift  +
tLiftHorizontal + CLAW_LENGTH_MILLIMETER;

  // input is horizontal plane angle: forward = 0 right > 0 left < 0
  tInputAngle = aPositionStruct->LeftRightDegree;
  // here we have horizontal plane angle: forward = 0 left > 0 right < 0
  tInputAngle *= -1;
  tInputAngleRad = tInputAngle * DEG_TO_RAD;
        polarToCartesian(tHorizontalArmAndClawShift,       tInputAngleRad,
aPositionStruct->BackFront, aPositionStruct->LeftRight);
}

void printPositionCartesian(struct ArmPosition *aPositionStruct) {
  Serial.print(F("X="));
  Serial.print(aPositionStruct->LeftRight);
  Serial.print(F("mm, Y="));
  Serial.print(aPositionStruct->BackFront);
```

```
  Serial.print(F("mm, Z="));
  Serial.print(aPositionStruct->DownUp);
  Serial.print(F("mm"));
}

void printPosition(struct ArmPosition *aPositionStruct) {
  Serial.print(F("LeftRight="));
  Serial.print(aPositionStruct->LeftRight);
  Serial.print("|");
  Serial.print(aPositionStruct->LeftRightDegree);
  Serial.print(F(" BackFront="));
  Serial.print(aPositionStruct->BackFront);
  Serial.print("|");
  Serial.print(aPositionStruct->BackFrontDegree);
  Serial.print(F(" DownUp="));
  Serial.print(aPositionStruct->DownUp);
  Serial.print("|");
  Serial.println(aPositionStruct->DownUpDegree);
}

void printPositionShortWithUnits(struct ArmPosition *aPositionStruct) {
  printPositionCartesian(aPositionStruct);
  Serial.print(F(" <-> "));
  Serial.print(aPositionStruct->LeftRightDegree);
  Serial.print(F(", "));
  Serial.print(aPositionStruct->BackFrontDegree);
  Serial.print(F(", "));
  Serial.print(aPositionStruct->DownUpDegree);
  Serial.println(F(" degree"));
}

float Distance_Sensing(){
  // Clears the trigPin
  long duration;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  float distance = (duration * 0.034 / 2) * 10;
  // Prints the distance on the Serial Monitor
```

```
  /*Serial.print("Distance(mm): ");
  Serial.println(distance);*/
}

bool Color_Sensing(){
  int frequency = 0;
  int R=0;
  int B=0;
  int G=0;
  // Setting R filt R photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,LOW);
  // Reading the output frequency
  R = pulseIn(sensorOut, LOW);
  delay(100);

  // Setting Green filt R photodiodes to be read
  digitalWrite(S2,HIGH);
  digitalWrite(S3,HIGH);
  // Reading the output frequency
  G = pulseIn(sensorOut, LOW);
  delay(100);

  // Setting B filt R photodiodes to be read
  digitalWrite(S2,LOW);
  digitalWrite(S3,HIGH);
  // Reading the output frequency
  B = pulseIn(sensorOut, LOW);
  delay(100);

  /*
  Serial.print("R= ");//printing name
  Serial.print (R);//printing R color frequency
  Serial.print("  ");
  Serial.print("G= ");//printing name
  Serial.print(G);//printing R color frequency
  Serial.print("  ");Serial.print("B= ");//printing name
  Serial.print(B);//printing R color frequency
  Serial.println("  ");
  */

  if(R<=255 && G<=165 && B<=0)
  {
    return true;
  }
```

```
  else {
    return false;
  }
}

#if defined(LOCAL_TRACE)
#undef LOCAL_TRACE
#endif

void setup() {
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(sensorOut, INPUT);
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  DU.attach(DUmotor);
  BF.attach(BFmotor);
  Claw.attach(CLAWmotor);

  // Setting frequency-scaling to 20%
  digitalWrite(S0,HIGH);
  digitalWrite(S1,LOW);
  Serial.begin(115200);
}

void loop() {
  if (Color_Sensing()){
    struct ArmPosition BallPosition
    float backfront= Distance_Sensing();
    BallPosition.BackFront=backfront;
    BallPosition.DownUp=BALL_Height_MILLIMETER;
    doInverseKinematics(&BallPosition);
    DU.write(BallPosition.DownUpDegree);
    BF.write(BallPosition.BackFrontDegree);
    delay(5000);
    printPositionShortWithUnits(&BallPosition);
    if (Distance_Sensing()<=30){
      Claw.write(60);
      Serial.println("Ball has been picked");
    }
  }
}
```

- ## Conclusion:

An agricultural robot is defined as any robotic device that can improve agricultural processes, by taking over many of the farmer's duties that are slow or intensive. Using robots in agriculture makes many tasks simpler, faster, and more effective. Agriculture robots can reduce the cost of cultivation by controlling the high cost of labor, efficient use of fertilizers, pesticides. The jobs in agriculture are a drag, dangerous, require intelligence and quick, though highly repetitive decisions hence robots can be rightly substituted with human operator.

Word Count: 4627

Page Count: 36