FACULTY OF ENGINERING
ELELECTERCITY DEP
COMPUTER AND CONTROL
FOURTH YEAR
FIRST SEMESTER

PUBLIC OF YEMEN
HIGH EDUCATION
SANA'A UNIVERSITY

# Embedded Systems

## Automated Production Line
## With Robotic Arm Control

Done by:
Mohammed Arif (202270203)
Supervision by:
*Dr. Farouq Al-Faheedi*
*Eng. Omair Al-Baadani*

Sana'a 2024

# Summary

This project report presents the development of a miniature automated production line model with an industrial robotic arm, controlled through an ESP32 microcontroller. The system is designed to demonstrate an efficient method of transferring items on a conveyor belt to specified locations. The production line integrates a DC motor, relay, and IR sensor to manage the movement and positioning of materials, ensuring precise stops and transfers by detecting item presence. The robotic arm, composed of four micro servo motors, is remotely controlled via a mobile-based application interface, allowing for efficient and user-friendly operation.

The report covers all stages of the system's design, including requirement gathering, system architecture, and implementation procedures, along with the tests conducted to validate the system's functionality. A methodology based on the Waterfall model is utilized, outlining the functional and non-functional requirements. Detailed diagrams, including block and flowcharts, are presented to illustrate the design and operational flow of the system. Experimental results demonstrate the effectiveness of the setup, with recommendations for future improvements provided in the conclusion.

# Contents

# Introduction

In recent years, automated production lines have become essential in modern manufacturing, allowing for greater efficiency, precision, and scalability. These systems often rely on a combination of mechanical components, electronic control systems, and software to coordinate operations and handle various tasks, such as object transportation, sorting, and quality control.

This project aims to build a scaled-down prototype of an automated production line to demonstrate the fundamental principles of automation and control systems. The model includes a conveyor with a robotic arm capable of transferring objects along the line. The entire system is managed by an ESP32 microcontroller, with commands issued through a mobile interface.

In this report, we will discuss the process of developing this automated system, including its design, implementation, and testing. By integrating sensors, servos, and a relay-controlled DC motor, this project seeks to emulate the functionalities of a real-world production line while exploring key challenges in embedded systems and electrical engineering.

# Aims & Objectives

The primary aim of this project is to design and implement a miniature automated production line model with a robotic arm that can effectively transfer objects across a conveyor system. The project aims to showcase the capabilities of using an ESP32 microcontroller for remote control, thereby creating a flexible, user-friendly system that emulates larger industrial processes.

- The key objectives of this project include:

Developing a functional conveyor belt system driven by a DC motor, with an IR sensor to detect items' presence, ensuring precise stopping for item handling.

1. Integrating a robotic arm with multiple servo motors to simulate item transfer, demonstrating real-world automation tasks like picking, placing, and sorting.
2. Creating a mobile application interface for remote control of the robotic arm, enabling users to manage the system’s operations easily.
3. Validating the system through various experiments to assess its performance, reliability, and adaptability for potential applications in industrial automation.

These objectives are intended to demonstrate how smaller systems can model real-life industrial processes, offering insights into how similar designs can be scaled up for larger applications.

# Methodology

The methodology employed in this project follows a structured approach, specifically utilizing the Waterfall Model of software engineering. This model is characterized by a sequential design process, where each phase must be completed before the next begins. The phases include requirement analysis, system design, implementation, testing, and maintenance. This approach is particularly effective for this project due to its clear structure and emphasis on documentation.

- *Software Engineering Model

The Waterfall Model was chosen for its simplicity and effectiveness in managing the project. It allows for thorough documentation at each stage, ensuring that all requirements are met and that the development process is well-defined.

- *Requirements Gathering

The requirements for the automated production line model were gathered through a combination of literature review, analysis of existing systems, and discussions with stakeholders. The process involved identifying both functional and non-functional requirements that would guide the development of the system.

- Functional Requirements

1. The system must be capable of detecting the presence of objects using an IR sensor.
2. The conveyor belt should move items along a predefined path using a DC motor.
3. The robotic arm must be able to pick up items from the conveyor and place them in designated locations.

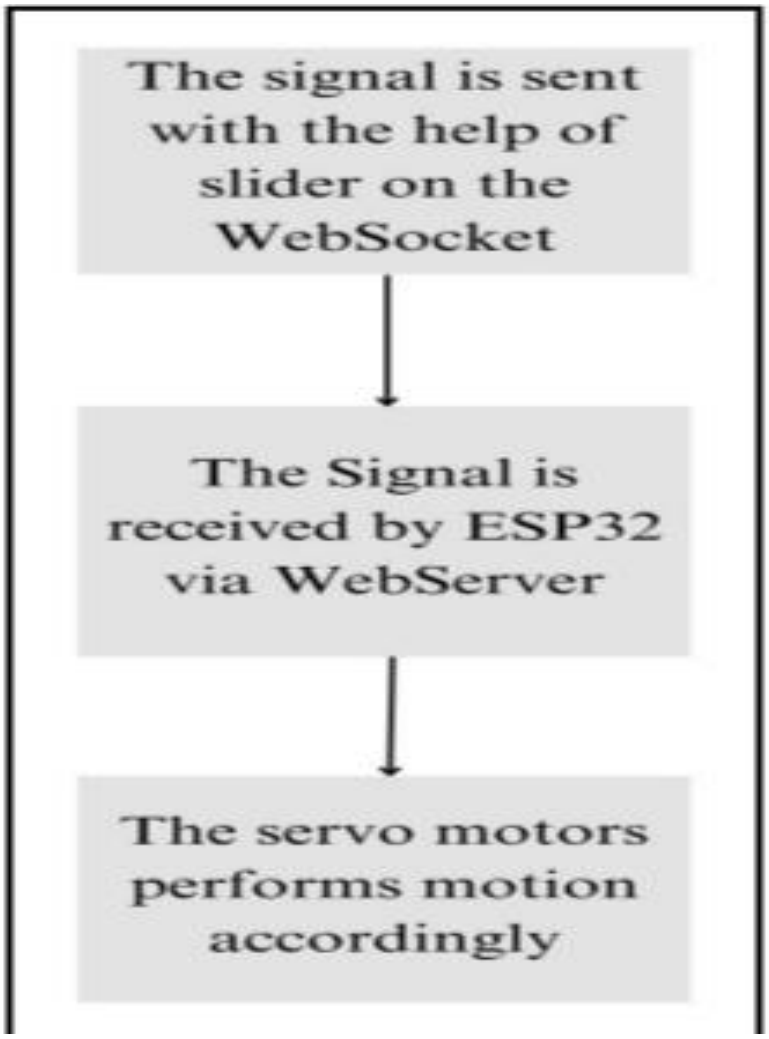
- Non-Functional Requirements

1. The system should be reliable, with minimal downtime during operation.
2. The user interface of the mobile application should be intuitive and easy to navigate.
3. The components used must be durable and suitable for repeated use.

***The hardware assembly of the robotic arm is done. The software includes WebSocket connections which has major libraries such as (ESPAsyncWebServer) Library, (AsyncTCP) Library, and (ESP32 Servo) Library.

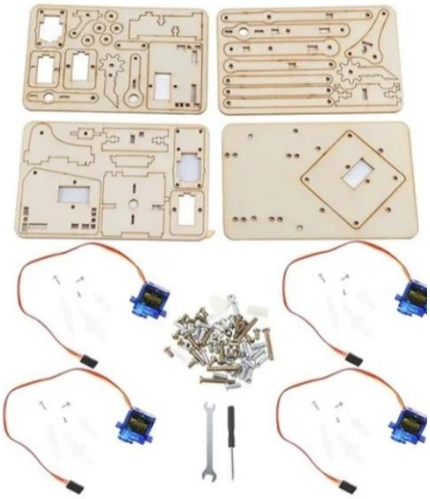
- ESPAsyncWebServer Library: It offers asynchronous handling of HTTP requests, enabling you to create web applications that are responsive and effective.
- Async TCP Library: A networking library for ESP8266 and ESP32 microcontroller boards is called the AsyncTCP library. It offers asynchronous TCP client and server features, enabling you to create TCP connections and engage in asynchronous network communication.

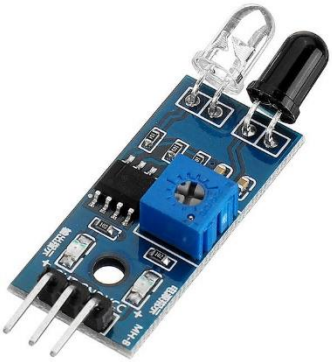
- ESP32 Servo Library: The ESP32 servo library provides an intuitive user interface for precise and fluid servo motor motions, simplifying the process of controlling servo motors with the ESP32 microcontroller.
- In Fig., the working flow of the project is explaine



# Components

- 4 x SG90 servo motors
- ESP-32 microcontroller
- Breadboard and jumper wires
- Power supply(5 v)
- Mechanical arm structure
-DC-motor.
- IR sensor.
- relay (5 VDC)
- automated production line model.



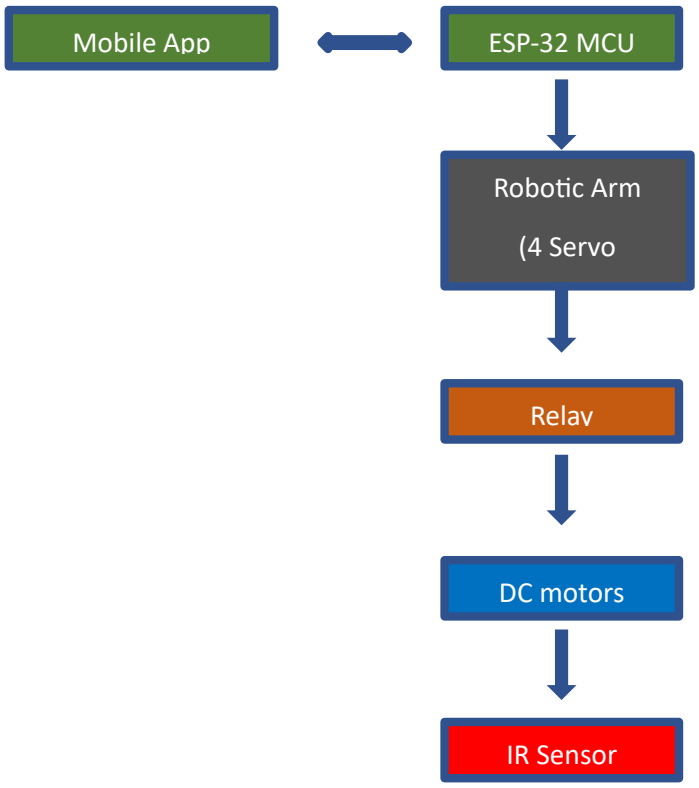


# System Design

In this section, we present the design of the automated production line system, detailing the architecture and the components used. The design phase is critical as it lays the foundation for the implementation and ensures that all requirements are effectively met.

- **System Block Diagram**

The system consists of several interconnected components that work together to achieve the desired functionality. Below is the block diagram that illustrates the major elements of the system



- ## Component Connections and Integration:

1. ESP32 Microcontroller: Acts as the control unit for the robotic arm, receiving commands from the mobile application to manage the movement of the four servo motors.

2. Mobile Application: Provides a user-friendly interface for controlling the robotic arm, allowing users to initiate movements and operations remotely.

3. Relay: Controls the power to the DC motor, enabling the movement of the conveyor belt independently of the ESP32.

4. DC Motor: Drives the conveyor belt to transport items to the designated stopping point.

5. IR Sensor: Detects the presence of items on the conveyor belt and, upon detection, signals the relay to stop the DC motor.
6. Robotic Arm: Composed of four servo motors, the arm is responsible for picking up items from the conveyor and placing them in specified locations, managed exclusively by the ESP-32.

# Implementation & Experiments

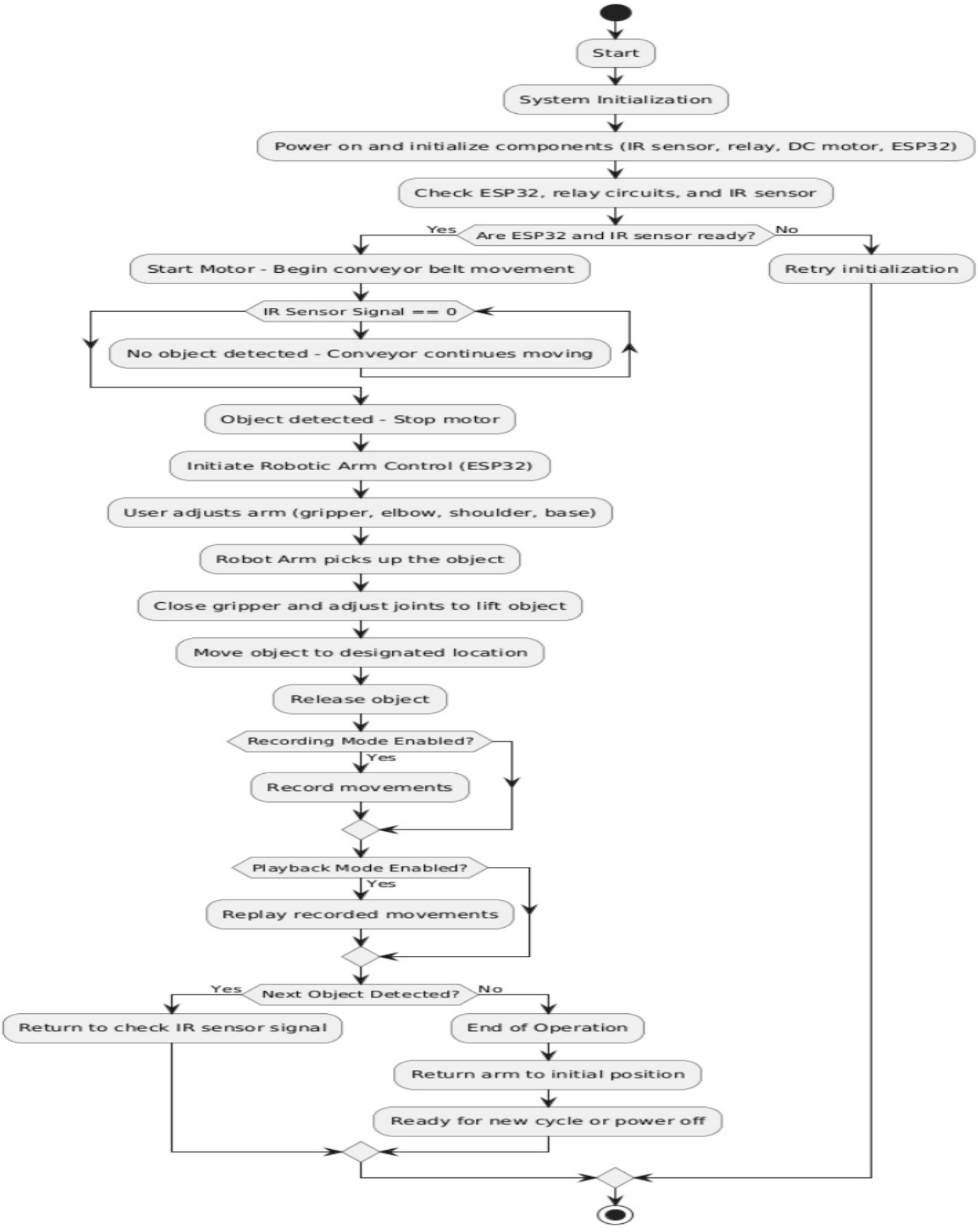
This section outlines the implementation phase of the automated production line model, detailing the development environment, system workflow, and the experimental results obtained during testing.

Development Environment

The project was developed in an integrated development environment (IDE) suitable for programming the ESP32 microcontroller, specifically the Arduino IDE. This platform provides the necessary tools and libraries to facilitate the development and debugging processes. The coding was primarily done in C/C++, leveraging the capabilities of the ESP32 for Wi-Fi connectivity and control of the servo motors.

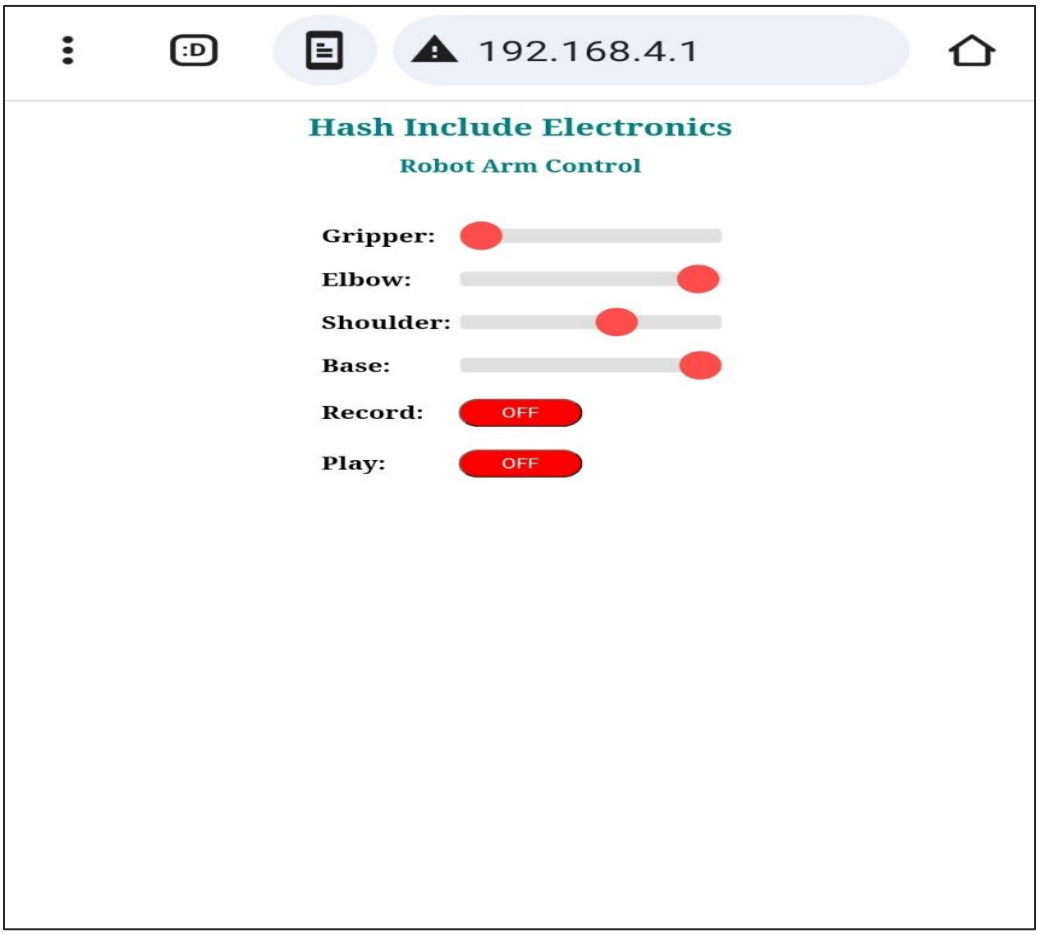
System Workflow (Flowchart Diagram)

The following flowchart illustrates the sequence of operations for the automated production line:


Start
System Initialization
Power on and initialize components (IR sensor, relay, DC motor, ESP32)
Check ESP32, relay circuits, and IR sensor
Are ESP32 and IR sensor ready?
Yes
No
Retry initialization
Start Motor - Begin conveyor belt movement
IR Sensor Signal == 0
No object detected - Conveyor continues moving
Object detected - Stop motor
Initiate Robotic Arm Control (ESP32)
User adjusts arm (gripper, elbow, shoulder, base)
Robot Arm picks up the object
Close gripper and adjust joints to lift object
Move object to designated location
Release object
Recording Mode Enabled?
Yes
Record movements
Playback Mode Enabled?
Yes
Replay recorded movements
Next Object Detected?
Yes
No
Return to check IR sensor signal
End of Operation
Return arm to initial position
Ready for new cycle or power off

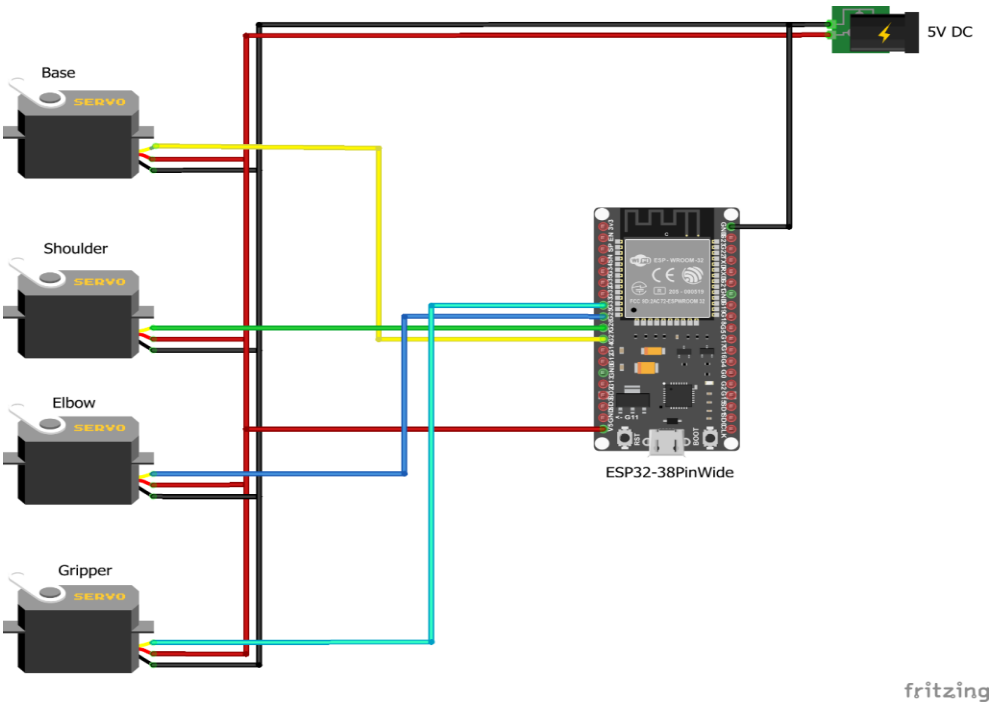
- User Interface and Execution Screens:

A simple and efficient user interface was developed to facilitate operator control of the robotic arm. This interface, accessible through a local Wi-Fi network via a mobile phone, includes several control elements. Each part of the robotic arm, such as the Gripper, Elbow, Shoulder, and Base, is controlled through individual sliders, enabling precise movement control. This setup allows the operator to easily capture and transport objects on the production line. Additionally, there are two buttons—"Record" and "Play"—which allow the user to record a sequence of movements and replay it later with a single click, adding automation and flexibility to the system.



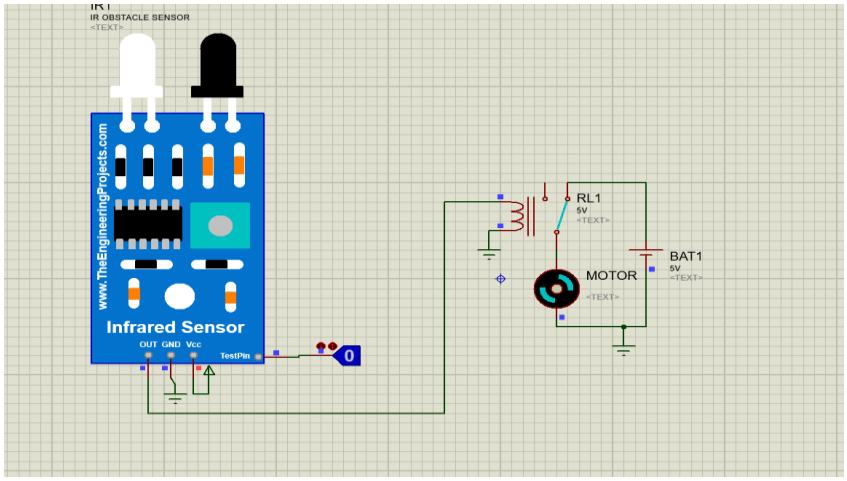
- Execution Screens:

The execution screens provide real-time feedback on the system status, allowing the user to monitor the response of the robotic arm and ensure it accurately follows commands from the interface. The main screen displays the current command states and shows the "ON/OFF" status of the Record and Play features, ensuring easy control and monitoring for the operator.

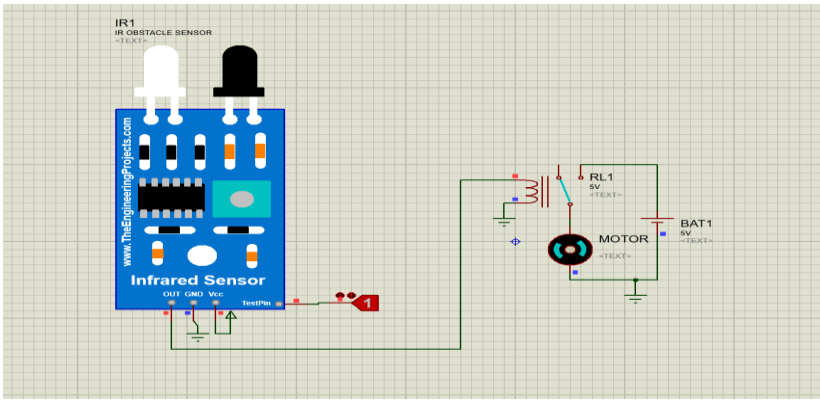


- System Operation and Testing:

The system was implemented and tested successfully, showing high efficiency in performing the required tasks. Upon system activation, the process begins with the transport of objects along the production line using a DC motor, controlled by a relay and an IR sensor. When the IR sensor detects a signal of "0," the motor continues running, allowing the production line to move forward. Once the sensor detects an object (indicated by a "1" signal), the motor stops, halting the production line.




After the production line stops, the robotic arm is activated via the user interface, allowing the operator to command it to pick up the object and move it to the desired location. Experimental results showed that the robotic arm responds accurately and quickly to commands sent through the interface, successfully picking up and relocating objects without issues.




Moreover, the "Record" and "Play" features were tested, demonstrating successful recording of specific arm movements and their automatic playback. This capability enhances operational efficiency and reduces the need for continuous human intervention.

The integration between the user interface, production line control system, and robotic arm proved effective in handling moving objects on the production line with flexibility and speed. This system is suitable for industrial applications requiring automation and rapid performance, providing a seamless solution for object detection, transportation, and placement.

# Conclusions

1. System Requirements Identification
    - The system requirements were carefully identified through a comprehensive analysis of the automated production line needs, including functional requirements such as motor control and sensor monitoring, as well as non-functional requirements like reliability and maintainability.

2. Design and Implementation
    - A system was designed integrating a DC motor and an IR sensor to control the production line's movement: the line moves when no object is detected and stops upon object detection.
3. A robotic arm connected to an ESP32 controller was developed, capable of picking up objects and transporting them to designated locations, with full control enabled through a mobile interface.

4. Integration of Control Structure
    - Appropriate control structures were utilized to create a logically organized and sequenced system, following the Waterfall Model for system development. This approach supported the orderly arrangement of design, development, and control processes.

5. Execution Environment
    - The system was built within a local environment using ESP32, along with manually assembled and tested control circuits, ensuring optimal performance of the motors and sensors.

6. Experiments and Results
    - Practical experiments demonstrated the system's efficiency, with the robotic arm successfully picking up and transporting objects with

precision. The experiments confirmed the effectiveness of the sensor and motor in managing the production line's movement as required.

7. Record and Playback Functionality
   - The design incorporated a feature for recording and playback of movements, allowing the system to autonomously repeat operations, thereby reducing the need for manual intervention and enhancing automation levels.

7. System Effectiveness

- Practical tests proved the system’s effectiveness and efficiency in achieving its intended functions, demonstrating reliable automated control of the production line and the robotic arm.

# REFERENCES:

[1] A. N. Reganti, S. Ananthapalli, B. C. V. Mohan, R. Mounica, and V. K. Mittal, "Mobile Quad-Controlled wireless Robotic Arm," 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), Delhi, India, 2016, pp. 1-5, doi: 10.1109/ICPEICES.2016.7853105.

[2] S. Gushi, H. Higa, H. Uehara and T. Soken, "A mobile robotic arm for people with severe disabilities: Evaluation of scooping foods," 2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), Okinawa, Japan, 2017, pp. 152-153, doi: 10.1109/ICIIBMS.2017.8279740.

[3] K. Belda and O. Rovný, "Predictive control of 5 DOF robot arm of autonomous mobile robotic system motion control employing mathematical model of the robot arm dynamics," 2017 21st International Conference on Process Control (PC), Strbske Pleso, Slovakia, 2017, pp. 339-344, doi: 10.1109/PC.2017.7976237.

[4] M. Farag, A. N. Abd Ghafar and M. H. Alsibai, "Grasping and Positioning Tasks for Selective Compliant Articulated Robotic Arm Using Object Detection and Localization: Preliminary Results," 2019 6th International Conference on Electrical and Electronics Engineering (ICEEE), Istanbul, Turkey, 2019, pp. 284-288, doi: 10.1109/ICEEE2019.2019.00061.

https://youtu.be/cVSvg6VQhGU?si=duwHh_IBaeN6JUvS

28/10/2024

https://www.espressif.com/en/products/socs/esp32

20/10/2024

https://youtu.be/yFj9W0iFBgI?si=O0dH7ASCLbxHZPEp

30/10/2024

# Appendixes:

This is the code that the arm and interface are programmed to ***

```
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

#include <ESP32Servo.h>
#include <iostream>
#include <sstream>

struct ServoPins
{
  Servo servo;
  int servoPin;
  String servoName;
  int initialPosition;
};
std::vector<ServoPins> servoPins =
{
  { Servo(), 27 , "Base", 90},
  { Servo(), 26 , "Shoulder", 90},
  { Servo(), 25 , "Elbow", 90},
  { Servo(), 33 , "Gripper", 90},
```

```
};

struct RecordedStep
{
  int servoIndex;
  int value;
  int delayInStep;
};
std::vector<RecordedStep> recordedSteps;

bool recordSteps = false;
bool playRecordedSteps = false;

unsigned long previousTimeInMilli = millis();

const char* ssid     = "RobotArm";
const char* password = "12345678";

AsyncWebServer server(80);
AsyncWebSocket wsRobotArmInput("/RobotArmInput");

const char* htmlHomePage PROGMEM = R"HTMLHOMEPAGE(
<!DOCTYPE html>
<html>
  <head>
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
    <style>
```

```
input[type=button]
{
 background-color:red;color:white;border-radius:30px;width:100%;height:40px;font-size:20px;text-align:center;
}

.noselect {
  -webkit-touch-callout: none; /* iOS Safari */
    -webkit-user-select: none; /* Safari */
     -khtml-user-select: none; /* Konqueror HTML */
       -moz-user-select: none; /* Firefox */
        -ms-user-select: none; /* Internet Explorer/Edge */
            user-select: none; /* Non-prefixed version, currently
                                  supported by Chrome and Opera */
}

.slidecontainer {
  width: 100%;
}

.slider {
  -webkit-appearance: none;
  width: 100%;
  height: 20px;
  border-radius: 5px;
  background: #d3d3d3;
  outline: none;
```

```
  opacity: 0.7;
  -webkit-transition: .2s;
  transition: opacity .2s;
}

.slider:hover {
  opacity: 1;
}

.slider::-webkit-slider-thumb {
  -webkit-appearance: none;
  appearance: none;
  width: 40px;
  height: 40px;
  border-radius: 50%;
  background: red;
  cursor: pointer;
}

.slider::-moz-range-thumb {
  width: 40px;
  height: 40px;
  border-radius: 50%;
  background: red;
  cursor: pointer;
}
```

```
  </style>


 </head>
 <body class="noselect" align="center" style="background-color:white">


  <h1 style="color: teal;text-align:center;">Hash Include Electronics</h1>
  <h2 style="color: teal;text-align:center;">Robot Arm Control</h2>


  <table id="mainTable" style="width:400px;margin:auto;table-layout:fixed" CELLSPACING=10>
    <tr/><tr/>
    <tr/><tr/>
    <tr>
     <td style="text-align:left;font-size:25px"><b>Gripper:</b></td>
     <td colspan=2>
      <div class="slidecontainer">
        <input type="range" min="0" max="180" value="90" class="slider" id="Gripper" oninput='sendButtonInput("Gripper",value)'>
      </div>
     </td>
    </tr>
    <tr/><tr/>
    <tr>
     <td style="text-align:left;font-size:25px"><b>Elbow:</b></td>
     <td colspan=2>
      <div class="slidecontainer">
        <input type="range" min="0" max="180" value="90" class="slider" id="Elbow" oninput='sendButtonInput("Elbow",value)'>
```

```
      </div>
     </td>
    </tr>
    <tr/><tr/>
    <tr>
     <td style="text-align:left;font-size:25px"><b>Shoulder:</b></td>
     <td colspan=2>
      <div class="slidecontainer">
        <input type="range" min="0" max="180" value="90" class="slider" id="Shoulder" oninput='sendButtonInput("Shoulder",value)'>
      </div>
     </td>
    </tr>
    <tr/><tr/>
    <tr>
     <td style="text-align:left;font-size:25px"><b>Base:</b></td>
     <td colspan=2>
      <div class="slidecontainer">
        <input type="range" min="0" max="180" value="90" class="slider" id="Base" oninput='sendButtonInput("Base",value)'>
      </div>
     </td>
    </tr>
    <tr/><tr/>
    <tr>
     <td style="text-align:left;font-size:25px"><b>Record:</b></td>
     <td><input type="button" id="Record" value="OFF" ontouchend='onclickButton(this)'></td>
```

```
      <td></td>
    </tr>
    <tr/><tr/>
    <tr>
      <td style="text-align:left;font-size:25px"><b>Play:</b></td>
      <td><input type="button" id="Play" value="OFF" ontouchend='onclickButton(this)'></td>
      <td></td>
    </tr>
  </table>

  <script>
    var webSocketRobotArmInputUrl = "ws:\/\/" + window.location.hostname + "/RobotArmInput";
    var websocketRobotArmInput;

    function initRobotArmInputWebSocket()
    {
      websocketRobotArmInput = new WebSocket(webSocketRobotArmInputUrl);
      websocketRobotArmInput.onopen    = function(event){};
      websocketRobotArmInput.onclose   = function(event){setTimeout(initRobotArmInputWebSocket, 2000);};
      websocketRobotArmInput.onmessage    = function(event)
      {
        var keyValue = event.data.split(",");
        var button = document.getElementById(keyValue[0]);
        button.value = keyValue[1];
        if (button.id == "Record" || button.id == "Play")
        {
```

```
    button.style.backgroundColor = (button.value == "ON" ? "green" : "red");
    enableDisableButtonsSliders(button);
   }
  };
}

function sendButtonInput(key, value)
{
  var data = key + "," + value;
  websocketRobotArmInput.send(data);
}

function onclickButton(button)
{
  button.value = (button.value == "ON") ? "OFF" : "ON" ;
  button.style.backgroundColor = (button.value == "ON" ? "green" : "red");
  var value = (button.value == "ON") ? 1 : 0 ;
  sendButtonInput(button.id, value);
  enableDisableButtonsSliders(button);
}

function enableDisableButtonsSliders(button)
{
  if(button.id == "Play")
  {
    var disabled = "auto";
    if (button.value == "ON")
```

```
        {
          disabled = "none";
        }
        document.getElementById("Gripper").style.pointerEvents = disabled;
        document.getElementById("Elbow").style.pointerEvents = disabled;
        document.getElementById("Shoulder").style.pointerEvents = disabled;
        document.getElementById("Base").style.pointerEvents = disabled;
        document.getElementById("Record").style.pointerEvents = disabled;
      }
      if(button.id == "Record")
      {
        var disabled = "auto";
        if (button.value == "ON")
        {
          disabled = "none";
        }
        document.getElementById("Play").style.pointerEvents = disabled;
      }
    }

    window.onload = initRobotArmInputWebSocket;
    document.getElementById("mainTable").addEventListener("touchend", function(event){
      event.preventDefault()
    });
  </script>
 </body>
</html>
```

```
)HTMLHOMEPAGE";

void handleRoot(AsyncWebServerRequest *request)
{
  request->send_P(200, "text/html", htmlHomePage);
}

void handleNotFound(AsyncWebServerRequest *request)
{
   request->send(404, "text/plain", "File Not Found");
}

void onRobotArmInputWebSocketEvent(AsyncWebSocket *server,
                   AsyncWebSocketClient *client,
                   AwsEventType type,
                   void *arg,
                   uint8_t *data,
                   size_t len)
{
  switch (type)
  {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n", client->id(), client->remoteIP().toString().c_str());
      sendCurrentRobotArmState();
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n", client->id());
```

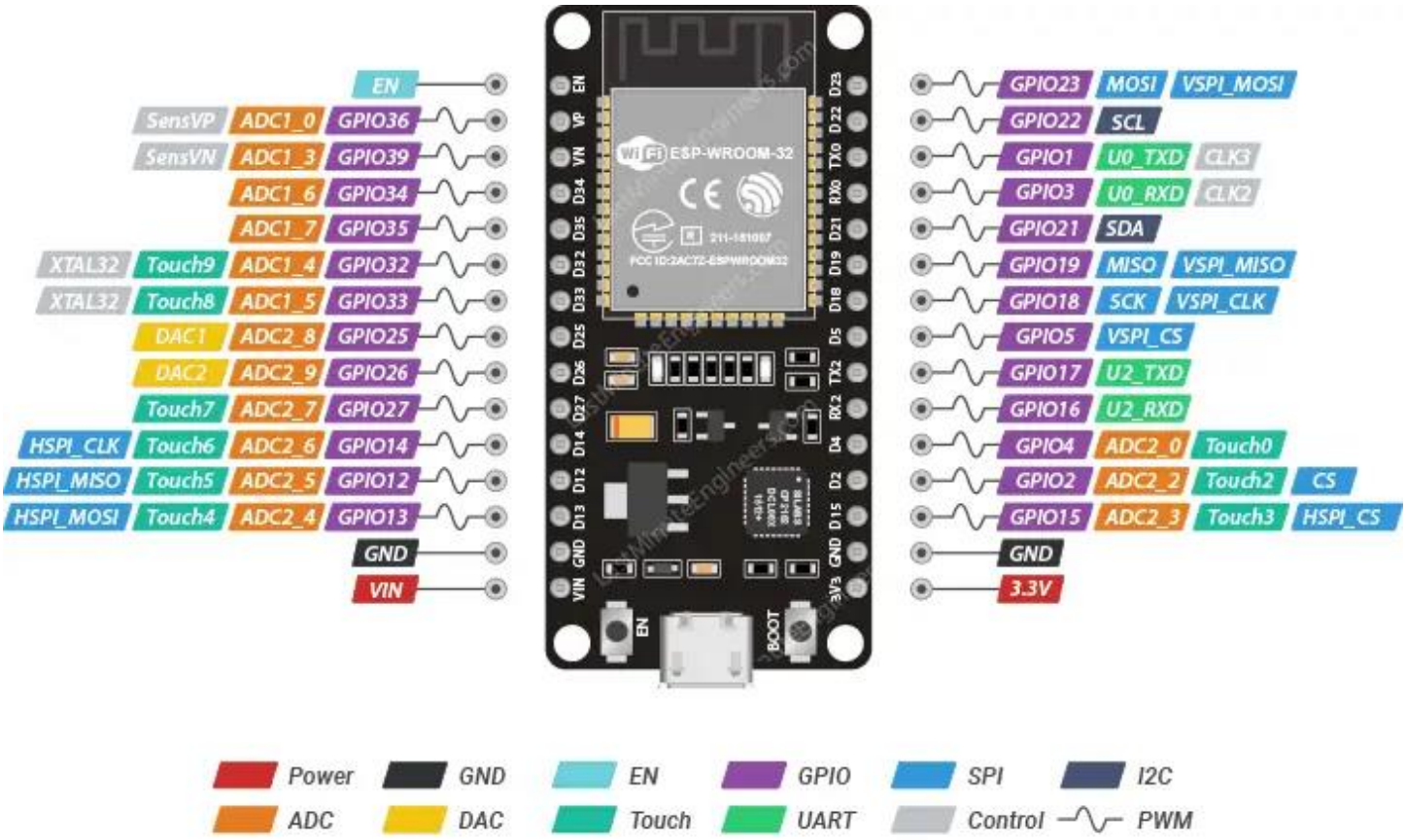
```
  break;
case WS_EVT_DATA:
  AwsFrameInfo *info;
  info = (AwsFrameInfo*)arg;
  if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT)
  {
   std::string myData = "";
   myData.assign((char *)data, len);
   std::istringstream ss(myData);
   std::string key, value;
   std::getline(ss, key, ',');
   std::getline(ss, value, ',');
   Serial.printf("Key [%s] Value[%s]\n", key.c_str(), value.c_str());
   int valueInt = atoi(value.c_str());

   if (key == "Record")
   {
    recordSteps = valueInt;
    if (recordSteps)
    {
     recordedSteps.clear();
     previousTimeInMilli = millis();
    }
   }
   else if (key == "Play")
   {
    playRecordedSteps = valueInt;
```

```
      }
      else if (key == "Base")
      {
        writeServoValues(0, valueInt);
      }
      else if (key == "Shoulder")
      {
        writeServoValues(1, valueInt);
      }
      else if (key == "Elbow")
      {
        writeServoValues(2, valueInt);
      }
      else if (key == "Gripper")
      {
        writeServoValues(3, valueInt);
      }

    }
    break;
  case WS_EVT_PONG:
  case WS_EVT_ERROR:
    break;
  default:
    break;
 }
}
```

```
void sendCurrentRobotArmState()
{
  for (int i = 0; i < servoPins.size(); i++)
  {
    wsRobotArmInput.textAll(servoPins[i].servoName + "," + servoPins[i].servo.read());
  }
  wsRobotArmInput.textAll(String("Record,") + (recordSteps ? "ON" : "OFF"));
  wsRobotArmInput.textAll(String("Play,") + (playRecordedSteps ? "ON" : "OFF"));
}

void writeServoValues(int servoIndex, int value)
{
  if (recordSteps)
  {
    RecordedStep recordedStep;
    if (recordedSteps.size() == 0) // We will first record initial position of all servos.
    {
      for (int i = 0; i < servoPins.size(); i++)
      {
        recordedStep.servoIndex = i;
        recordedStep.value = servoPins[i].servo.read();
        recordedStep.delayInStep = 0;
        recordedSteps.push_back(recordedStep);
      }
    }
    unsigned long currentTime = millis();
```

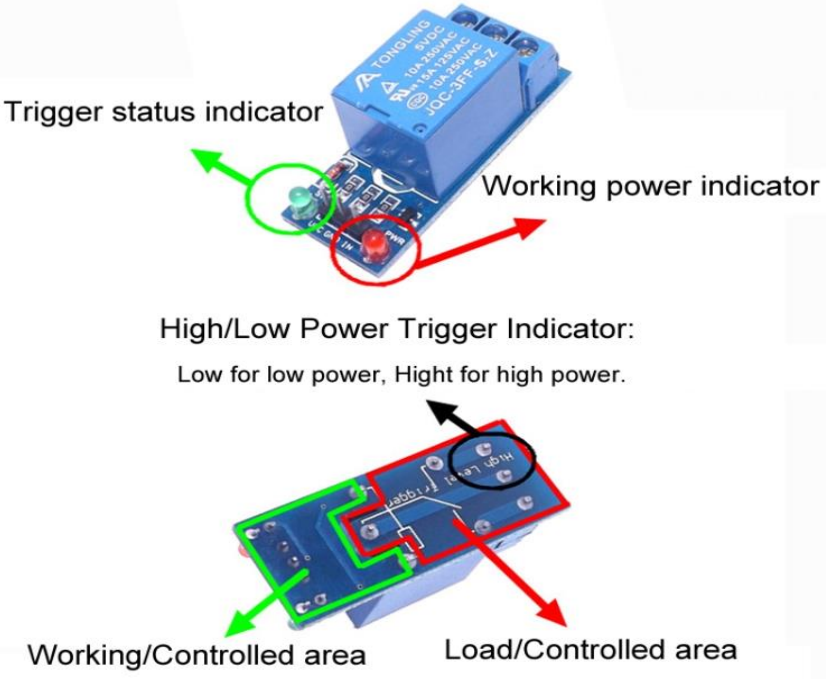
```
    recordedStep.servoIndex = servoIndex;
    recordedStep.value = value;
    recordedStep.delayInStep = currentTime - previousTimeInMilli;
    recordedSteps.push_back(recordedStep);
    previousTimeInMilli = currentTime;
  }
  servoPins[servoIndex].servo.write(value);
}

void playRecordedRobotArmSteps()
{
  if (recordedSteps.size() == 0)
  {
    return;
  }
  //This is to move servo to initial position slowly. First 4 steps are initial position
  for (int i = 0; i < 4 && playRecordedSteps; i++)
  {
    RecordedStep &recordedStep = recordedSteps[i];
    int currentServoPosition = servoPins[recordedStep.servoIndex].servo.read();
    while (currentServoPosition != recordedStep.value && playRecordedSteps)
    {
      currentServoPosition = (currentServoPosition > recordedStep.value ? currentServoPosition -
1 : currentServoPosition + 1);
      servoPins[recordedStep.servoIndex].servo.write(currentServoPosition);
      wsRobotArmInput.textAll(servoPins[recordedStep.servoIndex].servoName + "," +
currentServoPosition);
      delay(50);
```

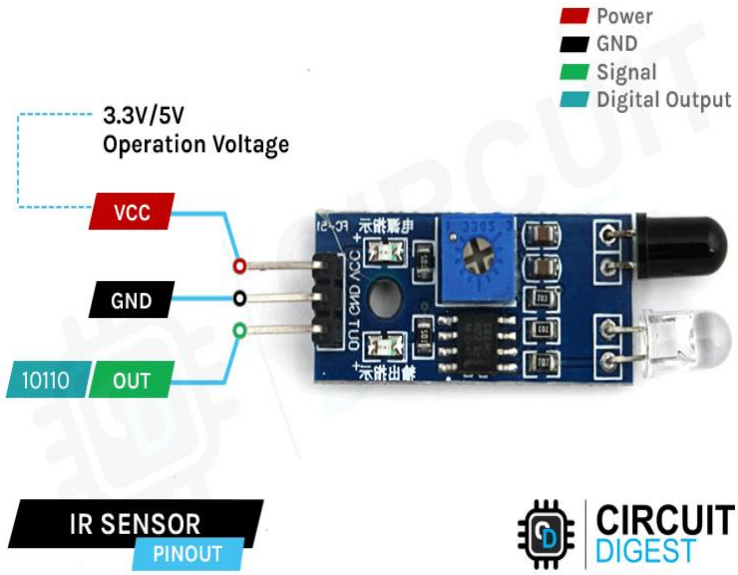
```
    }
  }
  delay(2000); // Delay before starting the actual steps.

  for (int i = 4; i < recordedSteps.size() && playRecordedSteps ; i++)
  {
    RecordedStep &recordedStep = recordedSteps[i];
    delay(recordedStep.delayInStep);
    servoPins[recordedStep.servoIndex].servo.write(recordedStep.value);
    wsRobotArmInput.textAll(servoPins[recordedStep.servoIndex].servoName + "," +
recordedStep.value);
  }
}

void setUpPinModes()
{
  for (int i = 0; i < servoPins.size(); i++)
  {
    servoPins[i].servo.attach(servoPins[i].servoPin);
    servoPins[i].servo.write(servoPins[i].initialPosition);
  }
}
void setup(void)
{
  setUpPinModes()
```




## ESP32 Dev. Board Pinout

Last Minute ENGINEERS.com




CIRCUIT DIGEST

- Specifications of Tower Pro SG90 Servo Motor?

Here are the specifications of a standard SG90 Servo we should know before getting using a servo motor for your project:

| | |
|---|---|
| Operating Voltage | 4.8 V - 6 V |
| Stall Torque | 1.6 kg-cm |
| Stall Current | 650 mA |
| Weight | 9g |
| Operating temperature | - 30 to +60 degree Celsius |

Operating Voltage: This servo motor operates within a voltage range of 4.8V to 6V. Ensure the power supply falls within this range to prevent damage or inefficient operation.

Stall Torque: This indicates the maximum torque output of the motor when it's unable to rotate further. It's the force the motor can exert at a given voltage, crucial for applications that require the motor to overcome resistance or hold positions against external forces.

Stall Current: Stall current refers to the amount of current drawn by a motor when it's operating at stall or maximum load condition, and the shaft is prevented from rotating. It signifies the maximum current which the motor can draw.

Weight: The weight of the servo motor is essential in applications where size and weight constraints are critical, such as in drones, small robotic arms, or RC vehicles.

Operating Temperature: Operating temperature refers to the range of temperatures within which a device or component can safely and effectively operate without risking damage or performance degradation.


**The TG9e boasts the same performance as other servo's 10x the price with a .10sec travel time and up to 1.5kg in torque and an ultra narrow dead bandwidth!**

**The TG9e performance is on par with the famous HXT900, however the TG9e isn't as resistant to crashes or over-loading.**

**Please always ensure your control surfaces are bind free.**