The German University in Cairo (GUC)
Faculty of Media Engineering and Technology
Computer Science and Engineering
Embedded System Architecture - CSEN 701

Embedded Smart Factory

Team Members :
Ahmed Amr Abolfadl, #55-1221

Mohamed Amr Shaker, #55-0903

Adam Adham Fayek, #55-3076

Zeina Mohamed Shalaby, #55-1703

Engy Sameh Fouad, #55-1361

Omar Ayman Orensa, #55-0599

Team Number :
Team #51

Under Supervision of :
Dr. Eng. Catherine M. Elias

Winter 2024

Contents

Chapter 1

Project Objectives

The primary objectives of this embedded systems project are:

1. To design and implement a conveyor belt system controlled by a Raspberry Pi Pico microcontroller.
2. To ensure accurate movement of the conveyor belt using a synchronous motor and relay setup with a level shifter for voltage adjustment.
3. To integrate multiple sensors for enhanced system functionality:
   - Touch sensor to start the system.
   - IR sensor to count objects passing on the conveyor belt.
   - Ultrasonic sensor to measure the distance of objects and trigger the servo motor for object removal when necessary.
4. To incorporate a servo motor for automatic removal of objects that are too close to end of conveyor belt.
5. To display real-time system data on an LCD screen, including the count of objects and the motor's operational status (on/off).
6. To ensure safe and reliable operation of the system through proper use of external power sources and component interfacing.

Chapter 2

Introduction

This project focuses on the design and development of an automated conveyor belt system using the Raspberry Pi Pico microcontroller. The system integrates various sensors and actuators to achieve efficient and accurate operation. By leveraging the capabilities of the Raspberry Pi Pico, this project aims to demonstrate a practical implementation of embedded systems in industrial automation.

The conveyor belt system is designed to perform the following tasks:

— Automatically start and operate using a touch sensor as the trigger mechanism.

— Count objects on the conveyor belt through the use of an IR sensor.

— Monitor the proximity of objects using an ultrasonic sensor and activate a servo motor to remove objects that are too close.

— Display system status and operational data, such as object count and motor status, on an LCD screen.

— Utilize a synchronous motor controlled via a relay and level shifter for precise belt movement and external power handling.

This system highlights the integration of hardware and software components to create a reliable and functional embedded system suitable for real-world applications.

Chapter 3

Methodology

The methodology for developing the automated conveyor belt system is as follows:

1. **System Design:** Define the overall architecture of the system, including the placement of sensors, actuators, and microcontroller connections.
2. **Component Selection:** Select appropriate hardware components, such as the Raspberry Pi Pico, synchronous motor, servo motor, sensors, and LCD display, ensuring compatibility and functionality.
3. **Circuit Design:** Create the circuit schematic to integrate all components, including the use of a level shifter for the relay control signal.
4. **Software Development:** Write and test the microcontroller code to:
   - Initialize and configure the sensors, motor, and LCD display.
   - Implement algorithms for object counting, distance measurement, and servo motor control.
   - Display relevant data on the LCD screen.
5. **Integration and Testing:** Assemble the hardware components and upload the software to the Raspberry Pi Pico. Test the system under different conditions to verify functionality and reliability.
6. **Calibration:** Fine-tune the sensor thresholds and motor operation to ensure accurate performance.
7. **Final Assembly:** Secure all components in their designated positions and conduct a final test to validate the complete system.

Chapter 4

Components

4.1 LCD

4.1.1 Datasheet

JHD204A.

4.1.2 Drivers

lcd.h - lcd.c

4.1.3 Connection Guide

Component Pin	Connection
VSS	GND
VCC	5V
VEE	GND (through resistor for contrast)
RS	GPIO Pin
R/W	GND
E	GPIO Pin
DB0	null
DB1	null
DB2	null
DB3	null
DB4	GPIO Pin
DB5	GPIO Pin
DB6	GPIO Pin
DB7	GPIO Pin
LED+	5V
LED-	GND

4.2 Servo Motor

4.2.1 Datasheet

SG90.

4.2.2 Drivers

servo.h - servo.c

4.2.3 Connection Guide

Component Pin	Connection
VCC	3.3V
GND	GND
IN(PWM)	GPIO pin

4.3 Relay

4.3.1 Datasheet

SRD-05VDC-SL-C.

4.3.2 Drivers

relay.h - relay.c

4.3.3 Connection Guide

Component Pin	Connection
VCC	5V
GND	GND
IN	GPIO pin (through level shifter)

4.4 Infrared Sensor

4.4.1 Datasheet

HW201.

4.4.2 Drivers

infrared.h - infrared.c

4.4.3 Connection Guide

Component Pin	Connection
VCC	3.3V
GND	GND
D0	GPIO pin
A0	null

4.5 Ultrasonic Sensor

4.5.1 Datasheet

HC - SR04.

4.5.2 Drivers

ultrasonic.h - ultrasonic.c

4.5.3 Connection Guide

Component Pin	Connection
VCC	5V
GND	GND
ECHO	GPIO pin
TRIG	GPIO pin

4.6 Touch Sensor

4.6.1 Datasheet

TTP223.

4.6.2 Drivers

touch.h - touch.h

4.6.3 Connection Guide

Component Pin	Connection
VCC	3.3V
GND	GND
OUT	GPIO pin

Chapter 5

Implementation

The following is the code main method implementation for the system:

5.1 Imports

```
#include "pico/multicore.h"

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "infrared.h"
#include "lcd.h"
#include "relay.h"
#include "servo.h"
#include "touch.h"
#include "ultrasonic.h"

#include "hardware/adc.h"
#include "hardware/pwm.h"
```

5.2 GPIO Mapping

```
uint trigPin = 15;
uint echoPin = 14;

int led_pin = 12;
int servo_pin_out = 13;
int touch_pin = 16;
int relay_pin = 17;
int infrared_pin = 18;
```

5.3 Variable Initialization

```
int servo_init_pos = 400;

bool motorOn = false;

int count = 0;
```

```
char str[9] = "Count: ";
char numStr[2];
```

5.4 Subsystem 1

```
void thread1(){
    setupUltrasonicPins(trigPin, echoPin);
    gpio_init(12);
    gpio_set_dir(12,GPIO_OUT);
    gpio_put(12,false);
    while(1) {
        uint64_t distance = getCm( trigPin, echoPin);
        if(distance < 20 && motorOn){
            //gpio_put(12, true);
            open();
            sleep_ms(1000);
            close();
            sleep_ms(1000);
        } else {
            //gpio_put(12, false);
            sleep_ms(100);
        }
    }
}
```

5.5 System Initialization

```
int main() {
    stdio_init_all();

    infrared_init();
    relay_init(relay_pin);
    touch_init(touch_pin);
    servo_init(servo_pin_out, servo_init_pos);

    //LCD block working
    lcdSetup(0, 1, 2, 3, 4, 5);
    lcdInit();
    lcdSetCursor(0, 0);
    lcdString("Welcome To EMBEDDED");
    lcdSetCursor(1, 0);
    lcdString("Team 51");

    multicore_launch_core1(thread1);
    // For debugging
    // gpio_init(12);
    // gpio_set_dir(12,GPIO_OUT);
    // gpio_put(12,false);
```

5.6 Subsystem 2

```
    while (1) {
        //display count
        lcdSetCursor(2, 0);
        size_t len = strlen(str);
        if (len > 8) { //string has 2 digits, remove them before concat.
            str[len - 2] = '\0';
        } else if (len > 7) { //string has 1 digit, remove it before concat.
            str[len - 1] = '\0';
        }
        sprintf(numStr, "%d", count);
        strcat(str, numStr);
        lcdString(str);

        lcdSetCursor(3,0);
        if (motorOn) {
            lcdString("           Motor ON!");
        } else {
            lcdString("           Motor OFF");
        }


        bool touchRead = touch_is_pressed(touch_pin);
        if(touchRead && !motorOn){
            relay_start(relay_pin); //start the motor
            motorOn = true;
            lcdSetCursor(2, 0);
            lcdString("                "); //clear
            count = 0; //reset count
            sleep_ms(500);
        } else if(touchRead && motorOn) {
            relay_stop(relay_pin);  //stop the motor
            motorOn = false;
            sleep_ms(500);
        }

        bool irRead = infrared_read_digital();
        if(!irRead  && motorOn){ //remember IR is negative logic
            count++; //increment count whenever ir senses object
            sleep_ms(1500);
        }

        sleep_ms(100);
    }

    return 0;
}
```

Chapter 6

Drivers

6.1 Actuator 1: LCD

6.1.1 lcd.h

```
#ifndef LCD_H
#define LCD_H

#include <stdint.h>

// Function Prototypes
void lcdSetup
(uint16_t RS, uint16_t En, uint16_t D4, uint16_t D5, uint16_t D6, uint16_t D7);

void lcdEnable(void);
void lcdWrite(uint8_t data);
void lcdCommand(uint8_t command);
void lcdInit(void);
void lcdChar(uint8_t ch);
void lcdString(char *string);
void lcdStringCount(char *string, int count);
void lcdWriteInt(char *format, uint32_t number);
void lcdWriteFloat(char *format, double number);
void lcdSetCursor(uint8_t row, uint8_t col);

#endif // LCD_H
```

6.1.2 lcd.c

```
#include "lcd.h"
#include <stdio.h>
#include "pico/stdlib.h"

//Global variables for LCD
uint16_t lcdRsPin;
uint16_t lcdEnPin;
uint16_t lcdDataPin[4];

void lcdSetup
```

```
(uint16_t RS, uint16_t En, uint16_t D4, uint16_t D5, uint16_t D6, uint16_t D7) {
    // Assign Pin values to global variables
    lcdRsPin = RS;
    lcdEnPin = En;
    lcdDataPin[0] = D4;
    lcdDataPin[1] = D5;
    lcdDataPin[2] = D6;
    lcdDataPin[3] = D7;

    // Initialize GPIO pins
    gpio_init(RS);
    gpio_init(En);
    gpio_set_dir(RS, GPIO_OUT);
    gpio_set_dir(En, GPIO_OUT);
    for (int i = 0; i < 4; i++) {
        gpio_init(lcdDataPin[i]);
        gpio_set_dir(lcdDataPin[i], GPIO_OUT);
    }
}

void lcdEnable() {
    // Give a small 0 to 1 and 1 to 0 pulse on Enable pin to transfer port data
    gpio_put(lcdEnPin, 1);
    sleep_us(1); // Small delay
    gpio_put(lcdEnPin, 0);
    sleep_us(50); // Ensure the command is latched
}

void lcdWrite(uint8_t data) {
    // Send the higher nibble (D7-D4)
    for (int i = 0; i < 4; i++) {
        gpio_put(lcdDataPin[i], (data >> (i + 4)) & 0x01);
    }
    lcdEnable();

    // Send the lower nibble (D3-D0)
    for (int i = 0; i < 4; i++) {
        gpio_put(lcdDataPin[i], (data >> i) & 0x01);
    }
    lcdEnable();
}

void lcdCommand(uint8_t command) {
    // Sends Command data to LCD
    gpio_put(lcdRsPin, 0); // Set RS to 0 for commands
    lcdWrite(command);
}

void lcdInit() {
    // Set to 4-bit mode
```

```
    lcdCommand(0x33); // Initialize LCD for 4-bit mode
    sleep_ms(5);
    lcdCommand(0x32); // Confirm 4-bit mode
    sleep_ms(5);

    lcdCommand(0x28); // Function Set: 4-bit mode, 2-line display, 5x8 dots
    lcdCommand(0x0C); // Display ON, Cursor OFF
    lcdCommand(0x06); // Entry Mode: Increment cursor, no shift
    lcdCommand(0x01); // Clear Display
    sleep_ms(2);      // Wait for the command to complete
}


void lcdChar(uint8_t ch) {
    // Display One Byte of Data to LCD
    gpio_put(lcdRsPin, 1); // Set RS Pin to 1 for Data
    lcdWrite(ch);
}

void lcdString(char *string) {
    // Display a String of characters
    while (*string)
        lcdChar(*string++);
}

void lcdStringCount(char *string, int count) { //TODO
    // Display a String of characters
    while (*string)
        lcdChar(*string++);
}

void lcdWriteInt(char *format, uint32_t number) {
    // Display integer numbers, format is standard C printf() like %d
    char buffer[20];
    sprintf(buffer, format, number);
    lcdString(buffer);
}

void lcdWriteFloat(char *format, double number) {
    // Display a floating point number
    char buffer[20];
    sprintf(buffer, format, number);
    lcdString(buffer);
}

void lcdSetCursor(uint8_t row, uint8_t col) {
    // Sets the cursor position
    gpio_put(lcdRsPin, 0); // RS=0;
    switch (row) {
    case 0:
```

```
        lcdWrite(0x80 + col);
        break;
    case 1:
        lcdWrite(0xC0 + col);
        break;
    case 2:
        lcdWrite(0x80 + 0x14 + col);
        break;
    case 3:
        lcdWrite(0xC0 + 0x14 + col);
        break;
    }
}
```

6.2 Actuator 2: Relay

6.2.1 relay.h

```
#ifndef relay_h
#define relay_h

#define RELAY_PIN 17

#include "pico/stdlib.h"

void relay_init(int relay_pin);

void relay_start(int relay_pin);

void relay_stop(int relay_pin);


#endif
```

6.2.2 relay.c

```
#include "relay.h"

#include "pico/stdlib.h"
#include <stdio.h>

// Initialize the touch sensor pin
void relay_init(int relay_pin) {
    gpio_init(relay_pin);
    gpio_set_dir(relay_pin, GPIO_OUT);
    //gpio_pull_down(relay_pin);
    gpio_put(relay_pin, true);
}

// Check if the touch sensor is pressed
```

```
void relay_start(int relay_pin) {
    gpio_put(relay_pin, false);
}

void relay_stop(int relay_pin) {
    gpio_put(relay_pin, true);
}
```

6.3 Actuator 3: Servo Motor

6.3.1 servo.h

```
#ifndef servo_h
#define servo_h

#define SERVO_PIN 13

void setMillis(int servoPin, float millis);
void servo_init(int servoPin, float startMillis);
void sweep();
void open();
void close();

#endif
```

6.3.2 servo.c

```
// Control a servo by degrees or millis

#include "pico/stdlib.h"
#include <stdio.h>
#include "hardware/pwm.h"
#include "hardware/clocks.h"
#include "servo.h"

float clockDiv = 64;
float wrap = 39062;
int currentMillis = 400;
bool direction = true;


void setMillis(int servoPin, float millis)
{
    pwm_set_gpio_level(servoPin, (millis/20000.f)*wrap);
}

void servo_init(int servoPin, float startMillis)
{
    gpio_set_function(servoPin, GPIO_FUNC_PWM);
    uint slice_num = pwm_gpio_to_slice_num(servoPin);
```

```
    pwm_config config = pwm_get_default_config();

    uint64_t clockspeed = clock_get_hz(5);
    clockDiv = 64;
    wrap = 39062;

    while (clockspeed/clockDiv/50 > 65535 && clockDiv < 256) clockDiv += 64;
    wrap = clockspeed/clockDiv/50;

    pwm_config_set_clkdiv(&config, clockDiv);
    pwm_config_set_wrap(&config, wrap);

    pwm_init(slice_num, &config, true);

    setMillis(servoPin, startMillis);
}

void sweep() {
    currentMillis += (direction)?300:-300;
    if (currentMillis >= 2400) direction = false;
    if (currentMillis <= 400) direction = true;
    setMillis(SERVO_PIN, currentMillis);
}

void open() {
    setMillis(SERVO_PIN, 2400);
}

void close() {
    setMillis(SERVO_PIN, 400);
}
```

6.4 Sensor 1: Infrared Sensor

6.4.1 infrared.h

```
#ifndef INFRARED_H
#define INFRARED_H

#include <stdint.h>
#include <stdbool.h>
#include <hardware/adc.h>

// Define pins
#define INFRARED_DO_PIN 18       // GPIO pin for digital output (DO)
#define INFRARED_AO_PIN 26      // ADC pin for analog output (AO, ADC0)

// Function prototypes
void infrared_init(void);
bool infrared_read_digital(void);
uint16_t infrared_read_analog(void);
```

```
#endif // IR_SENSOR_H
```

6.4.2 infrared.c

```
#include "infrared.h"
#include "pico/stdlib.h"
#include "hardware/adc.h"

// #include "hardware/adc.h"

// Initialize the IR sensor
void infrared_init(void) {
    // Initialize the digital output (DO) pin
    gpio_init(INFRARED_DO_PIN);
    gpio_set_dir(INFRARED_DO_PIN, GPIO_IN);

    // Initialize the analog input (AO) pin by enabling the ADC
    // adc_init();
    // adc_gpio_init(INFRARED_AO_PIN); // Initialize GPIO pin for ADC
}

// Read the digital value from the IR sensor's DO pin
bool infrared_read_digital(void) {
    return gpio_get(INFRARED_DO_PIN); // Returns true if high, false if low
}

// Read the analog value from the IR sensor's AO pin (0-4095 for 12-bit ADC)
uint16_t infrared_read_analog(void) {
    adc_select_input(0);              // Select ADC0 (assuming AO is connected to
    return adc_read();                 // Read and return ADC value (0 to 4095)
}
```

6.5 Sensor 2: Ultrasonic Sensor

6.5.1 ultrasonic.h

```
#ifndef ULTRASONIC_H
#define ULTRASONIC_H

void setupUltrasonicPins(uint trigPin, uint echoPin);
int getCm(uint trigPin, uint echoPin);
int getInch(uint trigPin, uint echoPin);

#endif // ULTRASONIC_H
```

6.5.2 ultrasonic.c

```
//Get readings from ultrasonic sensor
```

```
#include "pico/stdlib.h"
#include <stdio.h>
#include "hardware/gpio.h"
#include "hardware/timer.h"

int timeout = 26100;

void setupUltrasonicPins(uint trigPin, uint echoPin)
{
    gpio_init(trigPin);
    gpio_init(echoPin);
    gpio_set_dir(trigPin, GPIO_OUT);
    gpio_set_dir(echoPin, GPIO_IN);
}

uint64_t getPulse(uint trigPin, uint echoPin)
{
    gpio_put(trigPin, 1);
    sleep_us(10);
    gpio_put(trigPin, 0);

    uint64_t width = 0;

    while (gpio_get(echoPin) == 0); //tight_loop_contents(); // the echo pin is s
    gpio_put(6,1);

    absolute_time_t startTime = get_absolute_time();
    while (gpio_get(echoPin) == 1)
    {
        width++;
        sleep_us(1);
        if (width > timeout) return 0;
    }
    absolute_time_t endTime = get_absolute_time();

    return absolute_time_diff_us(startTime, endTime);
}

uint64_t getCm(uint trigPin, uint echoPin)
{
    uint64_t pulseLength = getPulse(trigPin, echoPin);
    return pulseLength / 29 / 2;
}

uint64_t getInch(uint trigPin, uint echoPin)
{
    uint64_t pulseLength = getPulse(trigPin, echoPin);
    return pulseLength / 74 / 2;
}
```

6.6 Sensor 3: Touch Sensor

6.6.1 touch.h

```
#ifndef touch_h
#define touch_h

#define TOUCH_PIN 16

#include "pico/stdlib.h"

void touch_init(int touch_pin);

bool touch_is_pressed(int touch_pin);

#endif
```

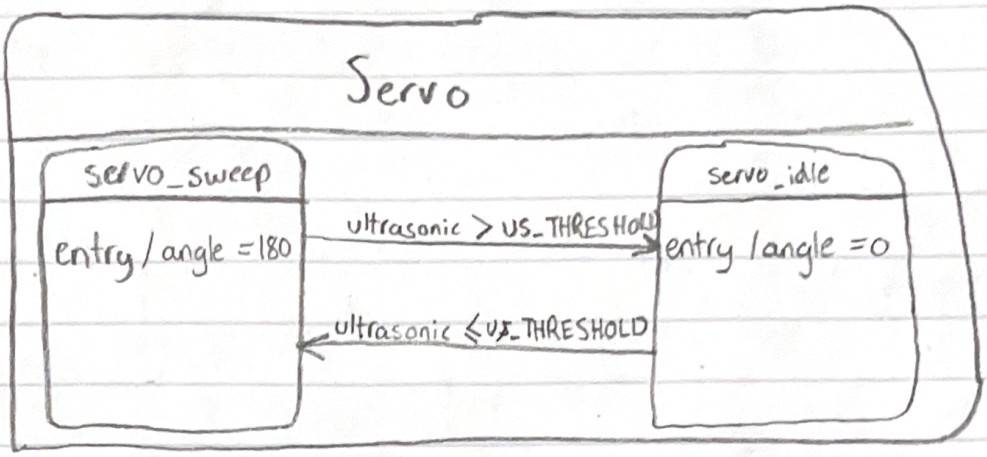
6.6.2 touch.c

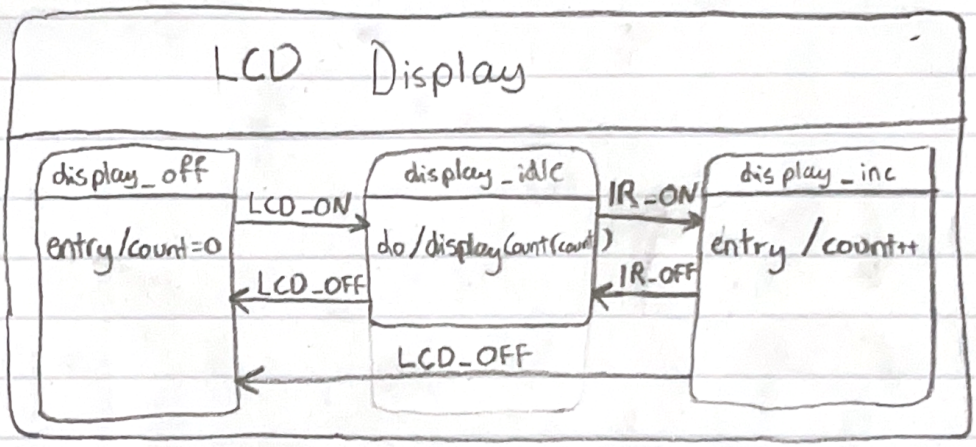
```
#include "touch.h"

#include "pico/stdlib.h"
#include <stdio.h>

// Initialize the touch sensor pin
void touch_init(int touch_pin) {
    gpio_init(touch_pin);
    gpio_set_dir(touch_pin, GPIO_IN);
    gpio_pull_down(touch_pin);
}

// Check if the touch sensor is pressed
bool touch_is_pressed(int touch_pin) {
    return gpio_get(touch_pin);
}
```

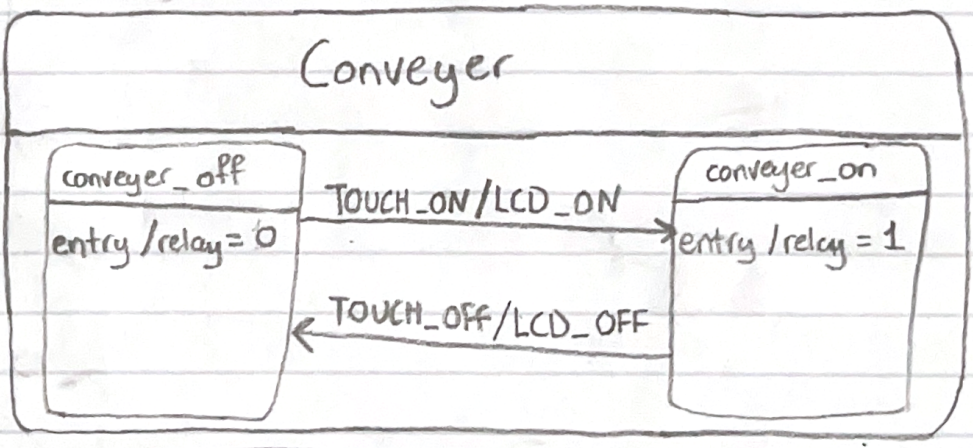
Chapter 7

State Flow Chart

Smart Embedded Factory







Team 51 :

55-0599	Omar Orensa	T-16
55-1221	Ahmed Amr Abolfadl	T-18
55-3076	Adam Adham Fayek	T-18
55-0903	Mohamed Amr Shaker	T-10
55-1703	Zeina Mohamed Shalaby	T-26
55-1361	Engy Sameh	T-26

Chapter 8

Concurrency

Utilizing Pico SDK's multicore.h we have divided our system into 2 subsystems each on one of the cores. The first subsystem is Th Ultrasonic sensor and Servo motor subsystem which was loaded on the "1" core of the microcontroller since it was in a function. The second subsystem is the Infrared sensor, Relay and Touch sensor which was loaded onto the "0" core of the microcontroller since it was on the main method.

Chapter 9

Hardware Design

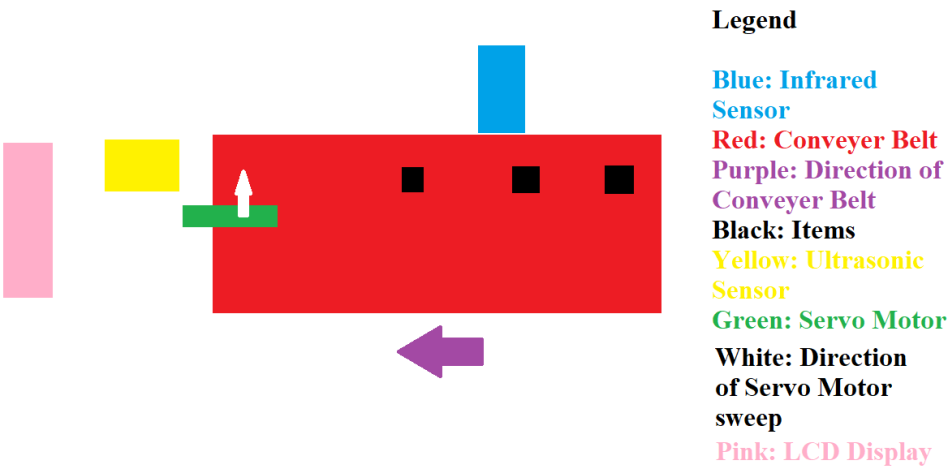


Figure 9.1 – Hardware Design

Chapter 10

Repository

Find the source code on abolfadlinho/embedded$_{factory}$.

Chapter 11

Results

The implemented conveyor belt system successfully met the project objectives. The following outcomes were observed:

1. The touch sensor reliably started the system, initiating the conveyor belt operation.
2. The IR sensor accurately detected and counted objects passing on the conveyor belt.
3. The ultrasonic sensor effectively measured the proximity of objects and triggered the servo motor for object removal when the set threshold distance was breached.
4. The synchronous motor provided smooth and consistent movement of the conveyor belt, controlled through the relay and level shifter.
5. The LCD screen displayed real-time information, including the count of objects and the motor's operational status (on/off), ensuring user-friendly interaction.
6. The system operated reliably under various test conditions, with accurate sensor readings and responsive actuator actions.

Chapter 12

Conclusion

In conclusion, the automated conveyor belt system developed in this project successfully demonstrated the integration of hardware and software components in an embedded system. The Raspberry Pi Pico microcontroller, along with the selected sensors and actuators, provided reliable and accurate performance. The system effectively achieved its objectives, including object detection, counting, and removal, as well as real-time status display.

This project showcases the potential of embedded systems in industrial automation, emphasizing the importance of proper design, component selection, and calibration. Future improvements could include the addition of advanced features, such as wireless monitoring or adaptive speed control, to enhance the system's functionality and scalability.