Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

DIVISIÓN DE TECNOLOGÍAS PARA LA INTEGRACIÓN CIBER-HUMANA

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Proyecto Final

TEMA: Proyecto carrito

NOMBRE DEL ESTUDIANTE:

Padilla Perez Jorge Daray

NOMBRE DE LA MATERIA: Diseño de interfaces

CALENDARIO: 2024-B

NOMBRE DEL PROFESOR: RUBEN ESTRADA MARMOLEJO

Contents

Introducción:

Este proyecto tiene como propósito desarrollar un sistema integrado que combine monitoreo en tiempo real, control autónomo y almacenamiento de datos en una base de datos MariaDB, mediante una interfaz gráfica en Qt y un microcontrolador ESP32. El sistema será capaz de medir y registrar variables como temperatura, humedad, voltaje de batería y distancia, y permitirá la operación de un robot móvil en dos modos: manual y autónomo.

En el modo autónomo, el robot tomará decisiones basadas en los datos de un sensor de distancia, como detenerse, cambiar de dirección o avanzar dependiendo de la proximidad a obstáculos. Por otro lado, en el modo manual, el usuario podrá controlar los movimientos básicos del robot desde la interfaz gráfica. Además, la interfaz incluirá gráficas dinámicas, campos de visualización y botones interactivos para facilitar el monitoreo y el control.

El proyecto integra hardware, programación en ESP32 y desarrollo en Qt para ofrecer una solución funcional y adaptable, con aplicaciones potenciales en robótica y sistemas de control.

Resumen

Introducción del Control del Servomotor:

- Se agregó funcionalidad para controlar un servomotor, moviéndolo a las posiciones 0° (izquierda), 90° (centro) y 180° (derecha).
- El servomotor toma medidas de distancia en cada posición, integrando estos datos en el proceso de toma de decisiones del carrito.

Actualización de la Máquina de Estados:

- La máquina de estados original se modificó para incluir los movimientos del servomotor y las lecturas asociadas.
- Los estados ahora incluyen:
- inicio: Posiciona el servo en el centro (90°).
- evaluar: Toma la distancia central y decide si continuar o finalizar.
- medir_derecha y medir_izquierda: Mueve el servo a la derecha (180°) y a la izquierda (0°) para medir distancias.
- decidir: Compara las tres distancias (centro, derecha, izquierda) para determinar la dirección óptima.
- avanzar: Mueve el carrito hacia la dirección seleccionada.

Modificaciones en la Función enviarComando:

- Adaptada para enviar comandos de ángulo al servomotor (0°, 90°, 180°) a través del WebSocket.
- Se utiliza también para enviar comandos de movimiento al carrito (avanzar, girar, etc.).

Integración con el WebSocket:

- Los comandos del servomotor y los datos de las lecturas de distancia se comunican entre el programa y la ESP32 usando WebSocket.
- Se procesa la información JSON para actualizar las lecturas de distancia.

Mantenimiento de Funcionalidades Previas:

- El resto de las funcionalidades del proyecto, como la lectura de sensores de temperatura, voltaje y humedad, no se modificaron.
- La lógica básica de movimiento del carrito permanece intacta, solo ajustada para integrar el servomotor.

Resultados Esperados:

- El carrito puede evaluar el entorno moviendo el servomotor y tomando decisiones basadas en las distancias medidas.
- Mejora la eficiencia de navegación al considerar diferentes direcciones antes de avanzar

Metodología y materiales

Materiales utilizados

Cantidad	Descripción
1	Controlador de motores L298D (Puente H)
1	Kit de carro (llantas, motores, estructura)
4	Baterías 18650
1	Cargador 18650
1	Portabatería 18650
1	Voltímetro Digital
∞	Cables
1	Madera
1	Sensor DHT22 (temperatura y humedad)
1	Laptop con Qt instalado
1	**ESP32-S3** (para manejar el WebSocket)
1	**Servomotor SG90 o similar** (control de ángulos)
1	**Sensor de distancia (**Ultrasónico HC-SR04 o similar**)**

Diagrama de flujo del procesamiento en QT

Máquina de estados

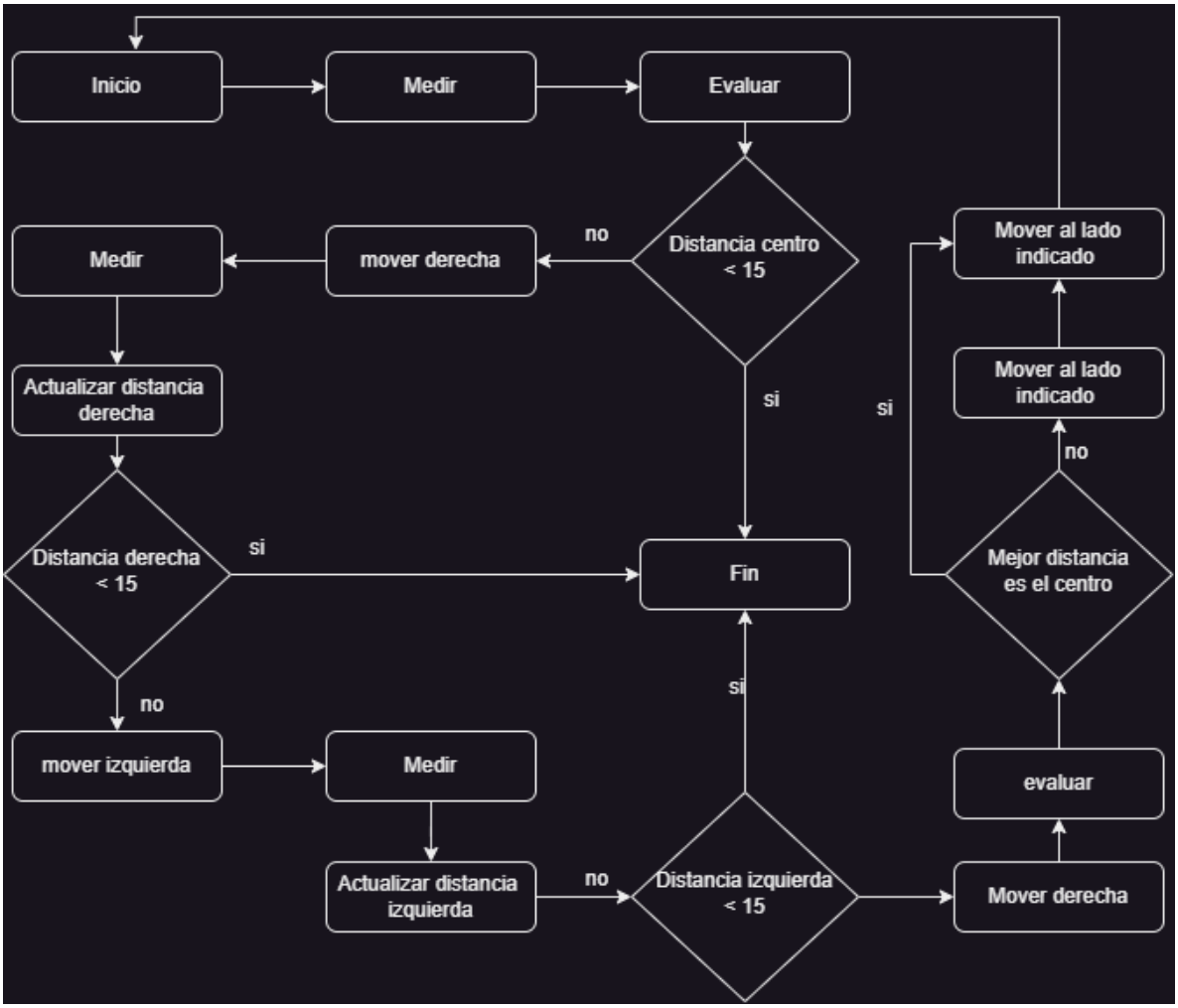

Diagrama 1 Maquina de estados

Conexión web

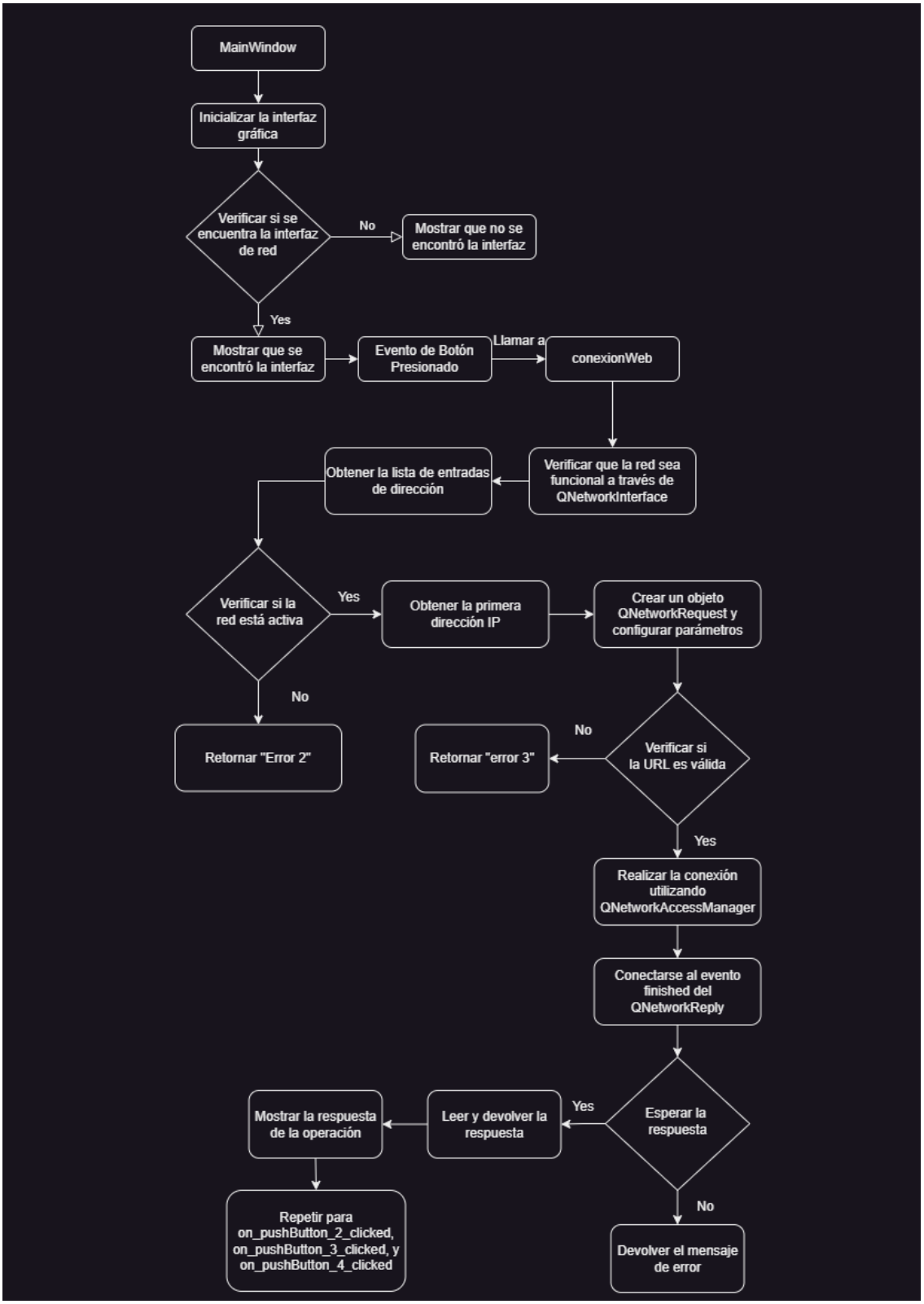

Diagrama 2 Conexion web

Diagramas (charts)

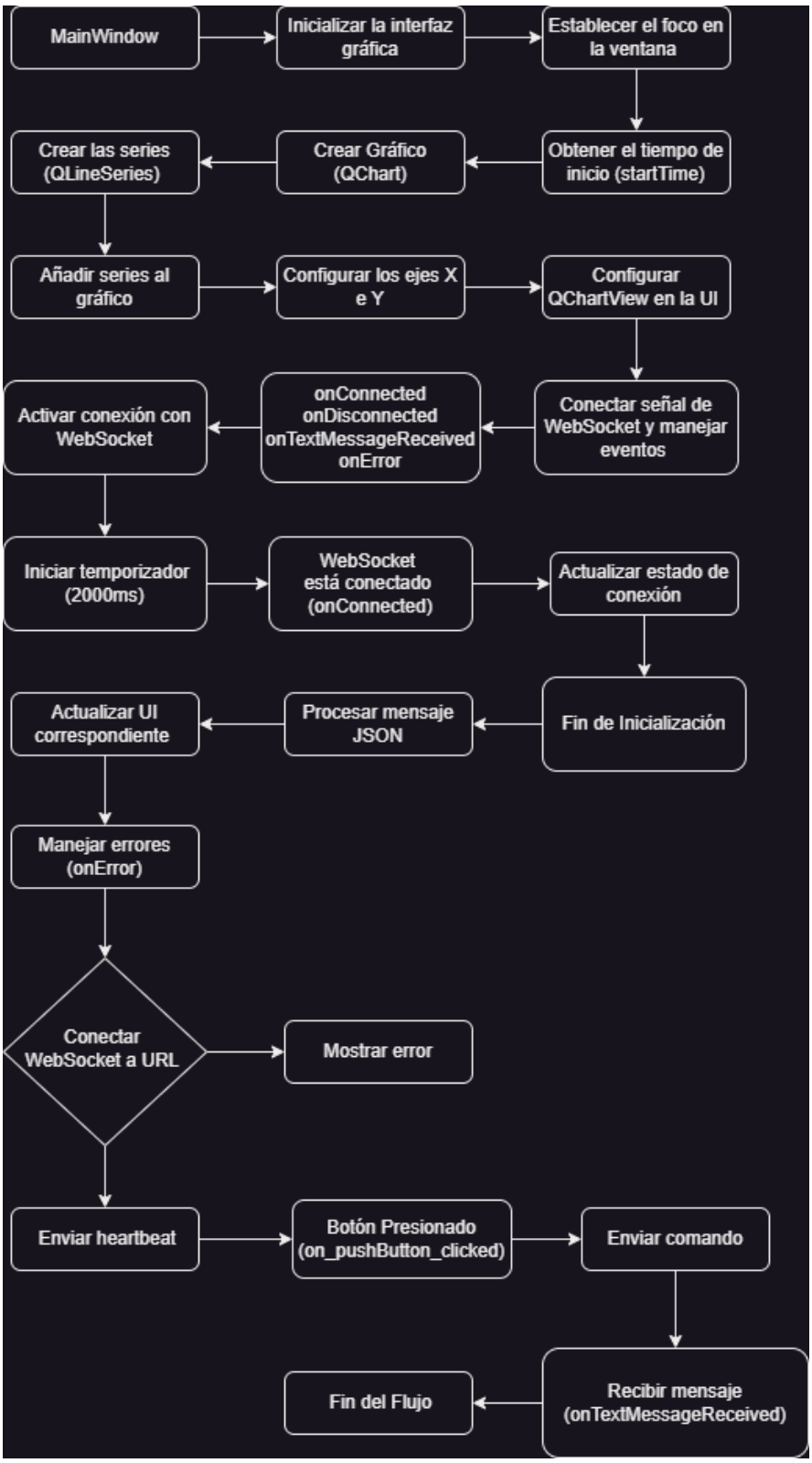

Diagrama 3 Graficas

Base de datos (DB)

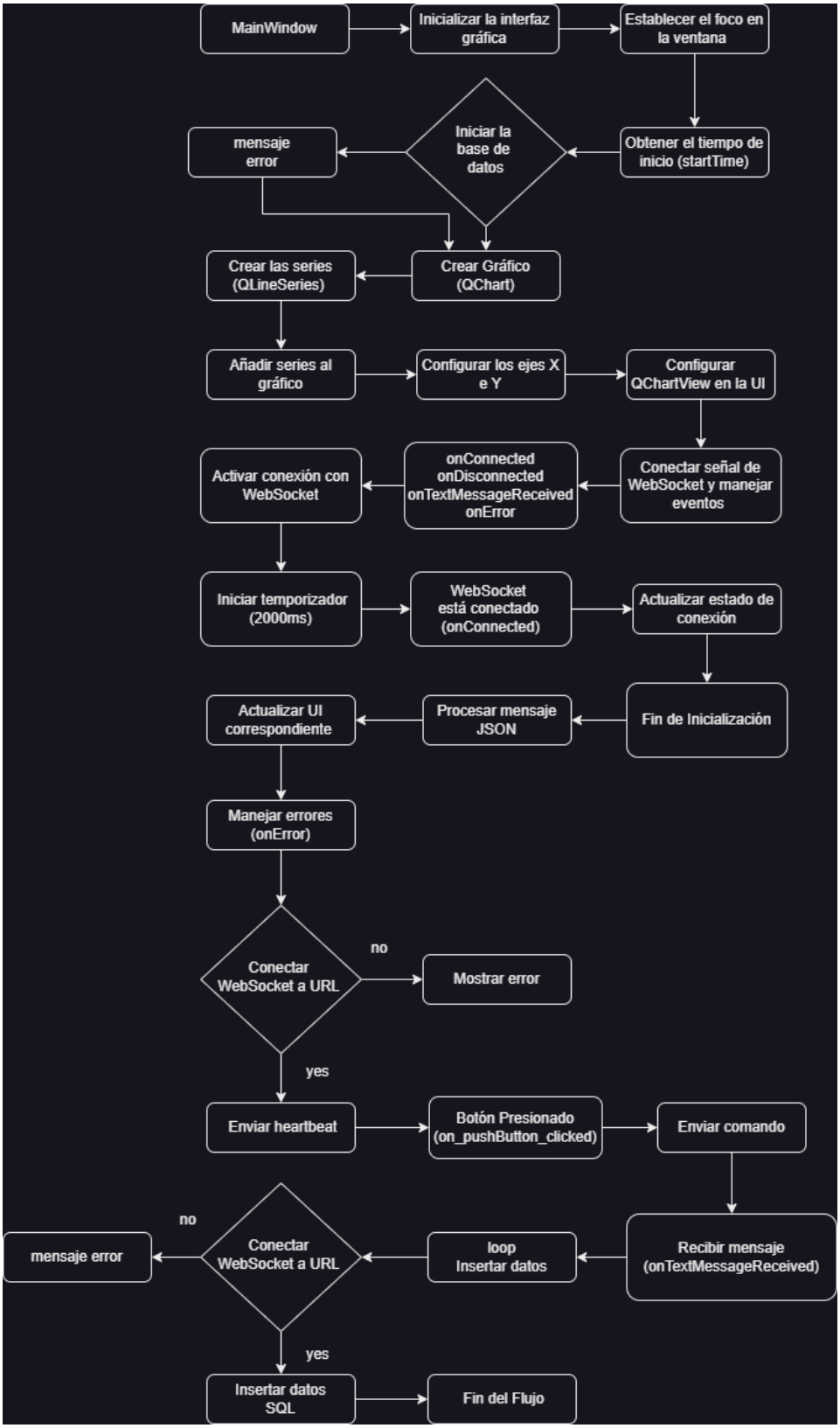

Diagrama 4 Conexion Base de datos

1.3 Hardware del sistema

Conexión Puente H

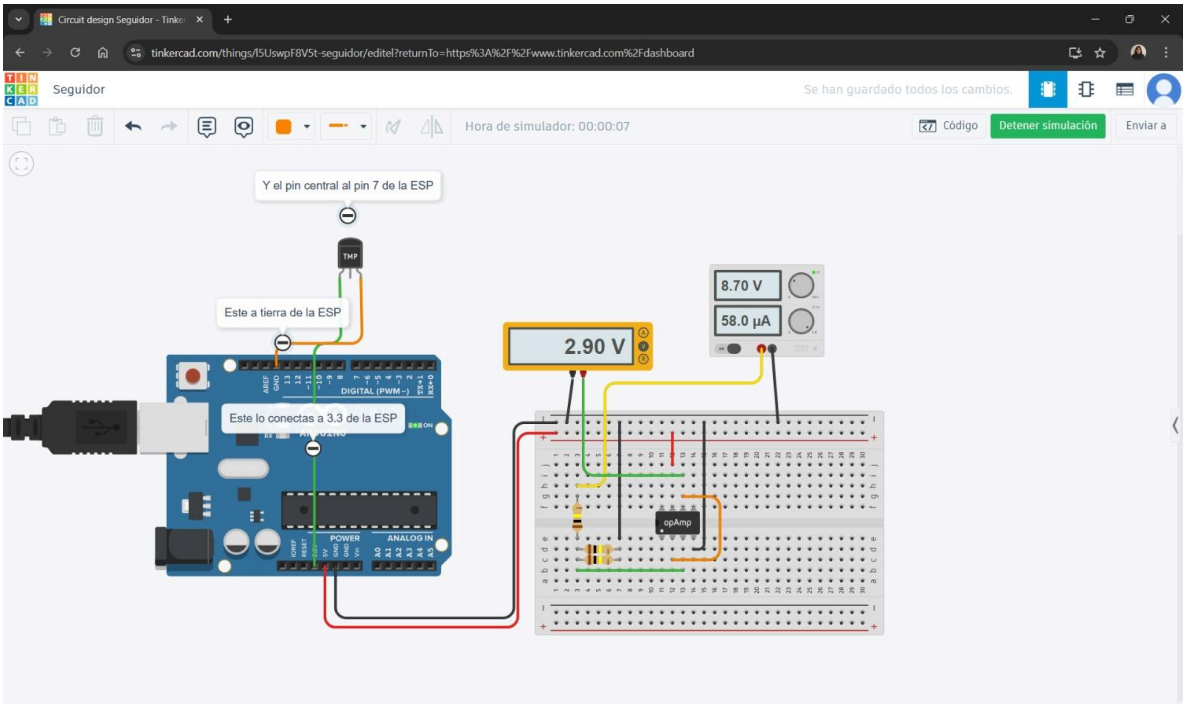

Ilustración 1 Conexion puente H

Conexión de servo con SRO4

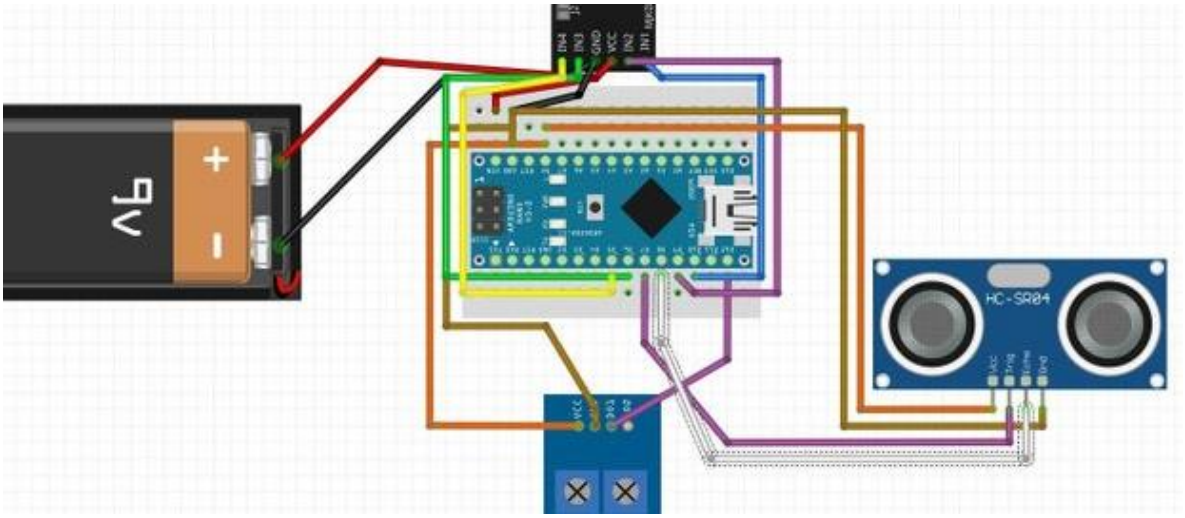

Ilustración 2 Conexion de servo con SRO4

1.4 Guía para crear el proyecto en QT

Guía para Crear el Proyecto en Qt

1. Preparación del Entorno

Requisitos del Software

1. Instala **Qt Creator** y el paquete de desarrollo (compatible con tu sistema operativo). Puedes descargarlo desde Qt.
2. Instala el compilador GCC (Linux), MinGW (Windows), o Xcode (Mac) si no está incluido en tu instalación.
3. Asegúrate de tener instalados los siguientes:
    - **ESP32 Arduino Core** en el IDE de Arduino.
    - Librerías necesarias para el ESP32 (WebSockets, Servo, etc.).

Materiales

Asegúrate de tener todos los materiales listados en la tabla anterior.

---

2. Diseño del Proyecto en Qt

a. Crear el Proyecto

1. Abre **Qt Creator**.
2. Selecciona **New Project** > **Application (Qt Widgets)**.
3. Configura:
    - Nombre del proyecto (e.g., *CarritoControl*).
    - Carpeta destino.
    - Framework (elige Qt Widgets para este caso).
4. Selecciona el compilador y las herramientas necesarias para tu sistema operativo.

b. Configura el Entorno

1. Ve al archivo CMakeLists.txt o .pro dependiendo de tu configuración.
2. Agrega la dependencia para usar WebSockets:

cpp

Copiar código

QT += core gui websockets

3. Compila el proyecto inicial para verificar que el entorno funciona correctamente.

---

3. Implementación del Proyecto

a. Configuración de la Interfaz (UI)

1. Usa **Qt Designer** para arrastrar y soltar widgets:
    - **TextEdit**: Muestra mensajes del sistema (log).
    - **Botones**: Opcional, para controles manuales si es necesario.
    - **LCD Display**: Para mostrar valores (opcional).

b. Creación de Clases

1. Abre el archivo mainwindow.h y define:
    - **WebSocket** para la comunicación con el ESP32.
    - **Métodos** para manejar comandos, estados, y mensajes.

c. Implementación del Código

1. Configura el WebSocket para conectarte al ESP32:

cpp

Copiar código

```
QWebSocket *m_webSocket;

QString esp32IP = "192.168.x.x";  // Reemplaza con la IP asignada al ESP32
```

2. Crea métodos clave:
    - **onConnected()**: Configura qué hacer al conectarte.
    - **onTextMessageReceived()**: Procesa los mensajes recibidos del ESP32.
    - **enviarComando()**: Envía comandos al ESP32.

---

4. Máquina de Estados

1. Implementa una máquina de estados para controlar las decisiones del carrito en función de las mediciones de distancia.
    - Actualiza los estados basándote en el flujo presentado:

cpp

Copiar código

```
void MainWindow::maquina_estados() {
   if (estado_actual == "inicio") {
      enviarComando(6); // Colocar sensor en 90°
      estado_sig = "evaluar";
```

```
  } else if (estado_actual == "evaluar") {
    // Procesar la distancia medida
    if (distancia_centro < 15) estado_sig = "fin";
    else {
      enviarComando(3); // Gira el sensor a la derecha (0°)
      estado_sig = "medir_derecha";
    }
  }
  // Continúa con los estados...
}
```

---

5. Comunicación con el ESP32

a. Código en el ESP32

1. Usa el IDE de Arduino para programar el ESP32.
2. Configura:
    - Servomotor para moverse entre 0°, 90°, y 180°.
    - Sensor ultrasónico para medir la distancia.
    - WebSocket para enviar datos a Qt y recibir comandos.

b. Formato de Mensajes

1. Los mensajes enviados desde el ESP32 al Qt deben ser en formato JSON:

json

Copiar código

```
{
  "distancia": 25.0,
  "timestamp": 1698015600
}
```

2. Qt debe interpretar y procesar este JSON en la función onTextMessageReceived():

cpp

Copiar código

```
void MainWindow::onTextMessageReceived(const QString &message) {
   QJsonDocument doc = QJsonDocument::fromJson(message.toUtf8());
   QJsonObject jsonObj = doc.object();
   distancia_actual = jsonObj["distancia"].toDouble();
}
```

---

6. Pruebas

1. Prueba la conexión WebSocket entre Qt y ESP32.
2. Asegúrate de que el servomotor se mueva a las posiciones correctas (0°, 90°, 180°).
3. Verifica que las distancias medidas se envíen y reciban correctamente.
4. Simula diferentes distancias para observar la lógica de decisiones del carrito.

---

7. Opcionales

- **Interfaz Gráfica**: Mejora la presentación con gráficos que representen los movimientos y las mediciones.
- **Control Manual**: Agrega botones para controlar manualmente el movimiento del carrito.
- **Persistencia**: Guarda los registros de las mediciones y estados.

---

8. Ensamblaje del Carrito

1. Monta los componentes: servomotor, sensor ultrasónico, ESP32, controlador de motores, y baterías.
2. Asegúrate de que los motores, servomotor, y sensores estén correctamente conectados.

---

9. Implementación Final

1. Combina la lógica de Qt y el ESP32.
2. Ejecuta el proyecto completo y verifica que el carrito:
    - Mida distancias correctamente.
    - Tome decisiones basadas en los valores.
    - Avance, se detenga, o gire según lo programado.

1.4.1 Archivos de cabeceras

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QWebSocket>
#include <QDebug>
#include <QJsonObject>
#include <QJsonDocument>
#include <QTimer>
#include <QDateTime>
#include <QChartView>
#include <QLineSeries>
#include <QList>
#include <QChart>
#include <QValueAxis>
#include <QLayout>
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void onConnected();
    void onTextMessageReceived(const QString &message);
    void onDisconnected();
    void onError(QAbstractSocket::SocketError error);
    void activarConexion(); // Función para el botón de enviar
    void sendHeartbeat();
    void connectWebSocket(const QUrl &url);
    void enviarComando(int comando);
    void keyPressEvent(QKeyEvent *event);
```

```
    void maquina_estados();
    void updateDistancia(double temp, qint64 timestamp);
    void updateHumidity(double humidity, qint64 timestamp);
    void updateVoltage(double voltage, qint64 timestamp);
    void updateTemperature(double temp, qint64 timestamp);
    void loop();
    bool insertarDatos(double temperatura, double voltaje, double humedad);
    void on_pushButton_8_clicked();

    void on_pushButton_9_clicked();

    void on_pushButton_4_clicked();

    void on_pushButton_5_clicked();

    void on_pushButton_6_clicked();

    void on_pushButton_7_clicked();

private:
    Ui::MainWindow *ui;

    QWebSocket *m_webSocket;
    bool m_connected;
    QString esp32IP = "192.168.0.160";

    QChart *chartTemp, *chartVolt, *chartHum, *chartDis;
    QChartView *chartViewTemp, *chartViewVolt, *chartViewHum;
    QLineSeries *seriesTemp, *seriesVolt, *seriesHum, *seriesdistancia;
    QValueAxis *axisXTemp, *axisXVolt, *axisXHum, *axisXDis;
    QValueAxis *axisYTemp, *axisYVolt, *axisYHum, *axisYDis;

    QList<QPointF> dataPointsTemp, dataPointsVolt, dataPointsHum,
dataPointsDis;
    qint64 startTime;
    double temperatura;
    double voltaje;
    double humedad;
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    QTimer *cronos = new QTimer(this);
};
#endif // MAINWINDOW_H
```

Código Principal

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

double distancia_actual = -1;
double distancia_centro = -1;
double distancia_derecha = -1;
double distancia_izquierda = -1;
QString estado_actual = "inicio";
QString estado_sig;

void MainWindow::maquina_estados() {
    qDebug() << estado_actual;

    if (estado_actual == "inicio") {
        enviarComando(90); // Posición inicial del servo al centro (90°)
        estado_sig = "evaluar";
    }
    else if (estado_actual == "evaluar") {
        distancia_centro = distancia_actual;
        qDebug() << "Distancia centro: " << distancia_centro;
        if (distancia_centro < 15)
            estado_sig = "fin";
        else {
            enviarComando(180); // Mover servo a la derecha (180°)
            estado_sig = "medir_derecha";
        }
    }
    else if (estado_actual == "medir_derecha") {
        distancia_derecha = distancia_actual;
        qDebug() << "Distancia derecha: " << distancia_derecha;
        if (distancia_derecha < 15)
            estado_sig = "fin";
        else {
            enviarComando(8); // Mover servo a la izquierda (0°)
            estado_sig = "medir_izquierda";
        }
    }
    else if (estado_actual == "medir_izquierda") {
        distancia_izquierda = distancia_actual;
        qDebug() << "Distancia izquierda: " << distancia_izquierda;
        if (distancia_izquierda < 15)
```

```
            estado_sig = "fin";
        else {
            enviarComando(90); // Volver servo al centro (90°)
            estado_sig = "decidir";
        }
    }
    else if (estado_actual == "decidir") {
        if ((distancia_centro < distancia_derecha) && (distancia_centro <
distancia_izquierda)) {
            qDebug() << "Gano centro";
            estado_sig = "avanzar";
        } else if ((distancia_derecha < distancia_centro) &&
(distancia_derecha < distancia_izquierda)) {
            qDebug() << "Gano derecha";
            enviarComando(3); // Comando para girar a la derecha
            estado_sig = "avanzar";
        } else if ((distancia_izquierda < distancia_centro) &&
(distancia_izquierda < distancia_derecha)) {
            qDebug() << "Gano izquierda";
            enviarComando(4); // Comando para girar a la izquierda
            estado_sig = "avanzar";
        } else {
            estado_sig = "avanzar";
        }
    }
    else if (estado_actual == "avanzar") {
        enviarComando(2); // Comando para avanzar
        estado_sig = "inicio";
    }
    else if (estado_actual == "fin") {
        qDebug() << "Fin del programa";
    }
    else {
        qDebug() << "Error en los estados";
    }
    estado_actual = estado_sig;
}

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    ui->setupUi(this);
```

```
    this->setFocus();  // Establecer el foco en la ventana principal

    startTime = QDateTime::currentSecsSinceEpoch();

    // Configuración para gráfica de temperatura
    chartTemp = new QChart();
    seriesTemp = new QLineSeries();
    chartTemp->addSeries(seriesTemp);
    chartTemp->setTitle("Temperatura");

    axisXTemp = new QValueAxis();
    axisXTemp->setTitleText("Tiempo (s)");
    axisXTemp->setRange(0, 40);
    chartTemp->addAxis(axisXTemp, Qt::AlignBottom);
    seriesTemp->attachAxis(axisXTemp);

    // Inicializar el eje Y para temperatura y configurarlo
    axisYTemp = new QValueAxis();
    axisYTemp->setTitleText("Temperatura (°C)");
    axisYTemp->setRange(0, 50);  // Ajusta según el rango esperado de
temperatura
    chartTemp->addAxis(axisYTemp, Qt::AlignLeft);
    seriesTemp->attachAxis(axisYTemp);

    ui->widget->setChart(chartTemp);

    // Configuración para gráfica de voltaje
    chartVolt = new QChart();
    seriesVolt = new QLineSeries();
    chartVolt->addSeries(seriesVolt);
    chartVolt->setTitle("Voltaje");

    axisXVolt = new QValueAxis();
    axisXVolt->setTitleText("Tiempo (s)");
    axisXVolt->setRange(0, 40);
    chartVolt->addAxis(axisXVolt, Qt::AlignBottom);
    seriesVolt->attachAxis(axisXVolt);

    // Inicializar el eje Y para voltaje y configurarlo
    axisYVolt = new QValueAxis();
    axisYVolt->setTitleText("Voltaje (V)");
    axisYVolt->setRange(0, 5);  // Ajusta según el rango esperado de voltaje
    chartVolt->addAxis(axisYVolt, Qt::AlignLeft);
    seriesVolt->attachAxis(axisYVolt);
```

```
    ui->widget_2->setChart(chartVolt);

    // Configuración para gráfica de humedad
    chartHum = new QChart();
    seriesHum = new QLineSeries();
    chartHum->addSeries(seriesHum);
    chartHum->setTitle("Humedad");

    axisXHum = new QValueAxis();
    axisXHum->setTitleText("Tiempo (s)");
    axisXHum->setRange(0, 40);
    chartHum->addAxis(axisXHum, Qt::AlignBottom);
    seriesHum->attachAxis(axisXHum);

    // Inicializar el eje Y para humedad y configurarlo
    axisYHum = new QValueAxis();
    axisYHum->setTitleText("Humedad (%)");
    axisYHum->setRange(0, 100);  // Ajusta según el rango esperado de
humedad
    chartHum->addAxis(axisYHum, Qt::AlignLeft);
    seriesHum->attachAxis(axisYHum);

    ui->widget_3->setChart(chartHum);

    // Configuración para gráfica de distancia
    chartDis = new QChart();
    seriesdistancia = new QLineSeries();
    chartDis->addSeries(seriesdistancia);
    chartDis->setTitle("distancia");

    axisXDis = new QValueAxis();
    axisXDis->setTitleText("Tiempo (s)");
    axisXDis->setRange(0, 40);
    chartDis->addAxis(axisXDis, Qt::AlignBottom);
    seriesdistancia->attachAxis(axisXDis);

    // Inicializar el eje Y para distancia y configurarlo
    axisYDis = new QValueAxis();
    axisYDis->setTitleText("distancia");
    axisYDis->setRange(0, 100);  // Ajusta según el rango esperado de
distancia
    chartDis->addAxis(axisYDis, Qt::AlignLeft);
    seriesdistancia->attachAxis(axisYDis);

    ui->widget_4->setChart(chartDis);
```

```
    db.setHostName("localhost");       // Cambia a la IP de tu servidor
MySQL si no es local
    db.setDatabaseName("interfaces");   // Reemplaza con el nombre de tu
base de datos
    db.setUserName("admin");         // Reemplaza con tu usuario de MySQL
    db.setPassword("changeme");      // Reemplaza con tu contraseña de MySQL
    if (!db.open()) {
        qDebug() << "Error al conectar con la base de datos:" <<
db.lastError().text();
    } else {
        qDebug() << "Conexión exitosa con la base de datos";
    }

    m_webSocket = new QWebSocket;
    m_connected = false;
    // Conectar WebSocket
    connect(m_webSocket, &QWebSocket::connected, this,
&MainWindow::onConnected);
    connect(m_webSocket, &QWebSocket::disconnected, this,
&MainWindow::onDisconnected);
    connect(m_webSocket, &QWebSocket::textMessageReceived, this,
&MainWindow::onTextMessageReceived);
    connect(m_webSocket, &QWebSocket::errorOccurred, this,
&MainWindow::onError);

    activarConexion();

    // Enviar un heartbeat cada 2 segundos
    connect(cronos, SIGNAL(timeout()), this, SLOT(maquina_estados()));
    cronos->start(1000);

    // Enviar un heartbeat cada 2 segundos
    QTimer *cronometro = new QTimer(this);
    connect(cronometro, SIGNAL(timeout()), this, SLOT(sendHeartbeat()));
    connect(cronometro,SIGNAL(timeout()),this,SLOT(loop()));
    cronometro->start(2000);

}

MainWindow::~MainWindow()
{
    delete ui;
}
```

```
void MainWindow::onDisconnected() {
    qDebug() << "WebSocket disconnected!";
    ui->textEdit->append("Desconectado del WebSocket");
    m_connected = false;  // Actualizar el estado de la conexión
}

void MainWindow::onError(QAbstractSocket::SocketError errores) {
    QString errorMsg;
    switch (errores) {
    case QAbstractSocket::HostNotFoundError:
        errorMsg = "Host no encontrado.";
        break;
    case QAbstractSocket::ConnectionRefusedError:
        errorMsg = "Conexión rechazada.";
        break;
    case QAbstractSocket::RemoteHostClosedError:
        errorMsg = "El host remoto cerró la conexión.";
        break;
    default:
        errorMsg = "Error desconocido.";
        break;
    }
    qDebug() << "Error de WebSocket: " << errorMsg;
    ui->textEdit->append("Error de WebSocket: " + errorMsg);
}

void MainWindow::connectWebSocket(const QUrl &url) {
    if (!m_connected) {
        m_webSocket->open(url);
        qDebug() << "Conectando a WebSocket en" << url;
        ui->textEdit->append("Conectando a WebSocket en: " +
url.toString());
    } else {
        qDebug() << "WebSocket ya está conectado";
        ui->textEdit->append("WebSocket ya está conectado");
    }
}

void MainWindow::activarConexion() {

    if (!esp32IP.isEmpty()) {
        QUrl url(QString("ws://") + esp32IP + "/ws");
        if (!m_connected) {
            connectWebSocket(url);
        }
```

```
    } else {
        ui->textEdit->append("Error: Debes ingresar la IP y el mensaje.");
    }
}

void MainWindow::sendHeartbeat() {
    QJsonObject heartbeatMessage;
    heartbeatMessage["type"] = "heartbeat";
    heartbeatMessage["timestamp"] = QDateTime::currentSecsSinceEpoch();  //
Unix timestamp

    QJsonDocument doc(heartbeatMessage);
    QString jsonString = doc.toJson(QJsonDocument::Compact);

    if (m_connected) {
        m_webSocket->sendTextMessage(jsonString);
        qDebug() << "Enviando heartbeat: " << jsonString;
    }
}

void MainWindow::onConnected() {
    qDebug() << "WebSocket connected!";
    ui->textEdit->append("Conectado a WebSocket");
    m_connected = true;  // Actualizar el estado de la conexión
}


void MainWindow::onTextMessageReceived(const QString &message) {
    //qDebug() << "Mensaje recibido:" << message;  // Muestra el mensaje
JSON completo recibido

    QJsonDocument doc = QJsonDocument::fromJson(message.toUtf8());
    if (!doc.isNull() && doc.isObject()) {
        QJsonObject jsonObj = doc.object();

        // Si el mensaje no contiene un timestamp, lo asigna desde Qt
        qint64 timestamp;
        if (jsonObj.contains("timestamp")) {
            timestamp = jsonObj["timestamp"].toVariant().toLongLong();
            //qDebug() << "Timestamp recibido:" << timestamp;
        } else {
            timestamp = QDateTime::currentSecsSinceEpoch();  // Asigna el
timestamp actual
            //qDebug() << "El mensaje JSON no contiene un campo de
'timestamp', se usa timestamp actual:" << timestamp;
```

```
        }

        // Procesar temperatura
        if (jsonObj.contains("temperature")) {
            double tempValue = jsonObj["temperature"].toDouble();
            //qDebug() << "Temperatura recibida:" << tempValue;
            updateTemperature(tempValue, timestamp);
            ui->lcdNumber->display(tempValue);
        } else {
            //qDebug() << "No se recibió campo de 'temperature' en el
mensaje JSON.";
        }

        // Procesar voltaje
        if (jsonObj.contains("adc_value")) {
            double voltValue = jsonObj["adc_value"].toDouble();
            //qDebug() << "Voltaje recibido:" << voltValue;
            updateVoltage(voltValue, timestamp);
            ui->lcdNumber_2->display(voltValue);
        } else {
            //qDebug() << "No se recibió campo de 'voltage' en el mensaje
JSON.";
        }

        // Procesar humedad
        if (jsonObj.contains("humidity")) {
            double humValue = jsonObj["humidity"].toDouble();
            //qDebug() << "Humedad recibida:" << humValue;
            updateHumidity(humValue, timestamp);
            ui->lcdNumber_3->display(humValue);
        } else {
            //qDebug() << "No se recibió campo de 'humidity' en el mensaje
JSON.";
        }
        // Procesar distancia
        if (jsonObj.contains("distancia")) {
            distancia_actual = jsonObj["distancia"].toDouble();
            qDebug() << "distancia recibida:" << distancia_actual;
            updateDistancia(distancia_actual, timestamp);
            ui->lcdNumber_4->display(distancia_actual);
        } else {
            qDebug() << "No se recibió campo de 'distancia_actual' en el
mensaje JSON.";
        }
    } else {
```

```
        qDebug() << "Error: mensaje JSON no válido.";
    }
}

void MainWindow::keyPressEvent(QKeyEvent *event)
{
    switch(event->key()) {
    case Qt::Key_Up:
        enviarComando(1);  // Comando para avanzar
        break;
    case Qt::Key_Down:
        enviarComando(2);  // Comando para retroceder
        break;
    case Qt::Key_Left:
        enviarComando(4);  // Comando para girar a la izquierda
        break;
    case Qt::Key_Right:
        enviarComando(3);  // Comando para girar a la derecha
        break;
    default:
        QMainWindow::keyPressEvent(event);
    }
}

void MainWindow::enviarComando(int comando) {
    // Crear el objeto JSON
    QJsonObject jsonObject;
    jsonObject["type"] = "comando";  // Tipo de mensaje
    jsonObject["comando"] = comando;  // Enviar el comando como entero

    // Convertir el objeto JSON a una cadena
    QJsonDocument jsonDoc(jsonObject);
    QString jsonString =
QString::fromUtf8(jsonDoc.toJson(QJsonDocument::Compact));

    // Enviar el mensaje a través del WebSocket
    m_webSocket->sendTextMessage(jsonString);
}

void MainWindow::updateTemperature(double temp, qint64 timestamp) {
    qint64 currentTime = timestamp - startTime;
    dataPointsTemp.append(QPointF(currentTime, temp));
    while (!dataPointsTemp.isEmpty() && currentTime -
dataPointsTemp.first().x() > 60) {
        dataPointsTemp.removeFirst();
```

```
    }
    seriesTemp->replace(dataPointsTemp);
    axisXTemp->setRange(currentTime - 60, currentTime);
    axisYTemp->setRange(0, 50);
}

void MainWindow::updateVoltage(double voltage, qint64 timestamp) {
    qint64 currentTime = timestamp - startTime;
    dataPointsVolt.append(QPointF(currentTime, voltage));
    while (!dataPointsVolt.isEmpty() && currentTime -
dataPointsVolt.first().x() > 60) {
        dataPointsVolt.removeFirst();
    }
    seriesVolt->replace(dataPointsVolt);
    axisXVolt->setRange(currentTime - 60, currentTime);
    axisYVolt->setRange(0, 9);
}

void MainWindow::updateHumidity(double humidity, qint64 timestamp) {
    qint64 currentTime = timestamp - startTime;
    dataPointsHum.append(QPointF(currentTime, humidity));
    while (!dataPointsHum.isEmpty() && currentTime -
dataPointsHum.first().x() > 60) {
        dataPointsHum.removeFirst();
    }
    seriesHum->replace(dataPointsHum);
    axisXHum->setRange(currentTime - 60, currentTime);
    axisYHum->setRange(0, 100);
}

void MainWindow::updateDistancia(double temp, qint64 timestamp) {
    qint64 currentTime = timestamp - startTime;
    dataPointsDis.append(QPointF(currentTime, temp));
    while (!dataPointsDis.isEmpty() && currentTime -
dataPointsDis.first().x() > 60) {
        dataPointsDis.removeFirst();
    }
    seriesdistancia->replace(dataPointsDis);
    axisXDis->setRange(currentTime - 60, currentTime);
    axisYDis->setRange(0, 100);
}

void MainWindow::loop(){
    // Verificar si los datos fueron recibidos correctamente
    if (temperatura != -1 && voltaje != -1 && humedad != -1 ) {
```

```
        // Insertar los datos en la base de datos
        insertarDatos(temperatura, voltaje, humedad);
    } else {
        qDebug() << "No se han recibido todos los datos necesarios.";
    }
}

bool MainWindow::insertarDatos(double temperatura, double voltaje, double
humedad) {
    // Obtener el timestamp en formato epoch (segundos desde 1970)
    qint64 timestamp = QDateTime::currentSecsSinceEpoch();

    // Crear la consulta SQL para insertar datos (temperatura, voltaje y
humedad)
    QSqlQuery query;
    query.prepare("INSERT INTO robot_data (timestamp, temperatura, voltaje,
humedad, distancia) VALUES (:timestamp, :temperatura, :voltaje, :humedad,
:distancia)");

    // Asignar los valores a los placeholders
    query.bindValue(":timestamp", timestamp);
    query.bindValue(":temperatura", temperatura);
    query.bindValue(":humedad", humedad);
    query.bindValue(":voltaje", voltaje);
    query.bindValue(":distancia", distancia_actual);

    // Ejecutar la consulta e imprimir el resultado
    if (!query.exec()) {
        qDebug() << "Error al insertar en la tabla robot_data:" <<
query.lastError().text();
        return false;  // Si hubo un error, regresa false
    }

    qDebug() << "Datos insertados exitosamente en robot_data:"
             << "Timestamp:" << timestamp
             << ", Temperatura:" << temperatura
             << ", Voltaje:" << voltaje
             << ", Humedad:" << humedad
             << ", Distancia: " << distancia_actual;
    return true;  // Regresa true si la inserción fue exitosa
}

void MainWindow::on_pushButton_8_clicked()
{
    cronos->stop();
```

```
}


void MainWindow::on_pushButton_9_clicked()
{
    cronos->start(1000);
}


void MainWindow::on_pushButton_4_clicked()
{
    enviarComando(1);
}


void MainWindow::on_pushButton_5_clicked()
{
    enviarComando(2);
}


void MainWindow::on_pushButton_6_clicked()
{
    enviarComando(3);
}


void MainWindow::on_pushButton_7_clicked()
{
    enviarComando(4);
}
```

Arduino código:

Practica4:

```
//Esp32 3.0.5
//Arduino 1.8.19


#include <WiFi.h>
#include <DHT.h>  //By adafruit 1.4.6 mas dependencias
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
//By lacemra 3.1.0 con modificaciones
#include <AsyncWebSocket.h>
#include <ArduinoJson.h>
//BenoitBlanchon
#include <time.h>
#include <ESP32Servo.h> //By Kevin Harrington v3.0.5

#define DHTPIN 7        // Pin donde está conectado el sensor
#define DHTTYPE DHT22  // Cambia a DHT11 si usas ese modelo
#define ADC_PIN 5
```

```
// Configura tus credenciales de WiFi
//const char* ssid = "MEGACABLE-C9A2";        // Reemplaza con tu SSID
//const char* password = "mMAFOE7rE@f$uDq";

// Configura tus credenciales de WiFi
const char* ssid = "GWN571D04";
const char* password = "ESP32CUCEI$$";
DHT dht(DHTPIN, DHTTYPE);
Servo miServo;
const int pinServo = 8; //probar


/*
const char* ssid = "GWN571D04";
const char* password = "ESP32CUCEI$$";
*/
// Configuración del servidor y WebSocket
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");

// Pin del ADC sugerido
const int adcPin = 34;  // GPIO 34 es un pin ADC libre
const int blinkPin = 13; // GPIO 13 para el LED
const int numParts = 500;  // Número de partes de la señal
int i = 0;  // Contador para las 50 partes de la señal senoidal
int tiempoMuestreo = 1000;
volatile int comando = -1;
volatile int velocidad = 0;

 bool motorActivado = false;
 long int tiempoMotorActivado = 0;
 long int tiempoMotor = 0;
 #define tiempoAdelante 500
 #define tiempoAtras    400
 #define tiempoDerecha  500
 #define tiempoIzquierda 500


// Variables para manejar el envío periódico de datos
bool heartbeatReceived = false;
// Indica si ya se ha recibido el primer heartbeat
unsigned long lastAdcSendTime = 0;  // Tiempo de la última lectura del ADC

// Variables para simulación de señal
bool simulateSineWave = true;
// Bandera para activar la señal senoidal
float sineFrequency = 2.0;
// Frecuencia de la señal senoidal
unsigned long startTime = 0;
// Tiempo de inicio para la señal senoidal
volatile long tiempoBlink = 1000;
// Función para actualizar el tiempo de la ESP32
void updateEsp32Time(unsigned long timestamp) {
    timeval tv;
    tv.tv_sec = timestamp;
    tv.tv_usec = 0;
    settimeofday(&tv, nullptr);
// Actualizar el tiempo de la ESP32
    Serial.println("Tiempo de la ESP32 actualizado.");
}

// Función para manejar mensajes JSON
void onWebSocketMessage(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType type, void *arg, uint8_t *data, size_t len) {
    if (type == WS_EVT_DATA) {
        Serial.print("Message length: ");
        Serial.println(len);  // Mostrar la longitud del mensaje

        if (len > 0) {
            // Convertir los datos recibidos en una cadena
            String message = String((char*)data);
```

```
Serial.println("Mensaje recibido: " + message);

            // Crear un objeto DynamicJsonDocument para almacenar el JSON
            DynamicJsonDocument doc(1024);

            // Intentar deserializar el mensaje como JSON
            DeserializationError error = deserializeJson(doc, message);

            // Si el mensaje es un JSON válido
            if (!error) {

Serial.println("Mensaje JSON válido recibido");

                // Comprobar si es un heartbeat
                const char* tipo = doc["type"];
                if (strcmp(tipo, "heartbeat") == 0) {
                    // Obtener el timestamp del heartbeat
                    unsigned long timestamp = doc["timestamp"];

Serial.print("Heartbeat recibido con timestamp: ");

Serial.println(timestamp);

                    // Actualizar el tiempo de la ESP32

updateEsp32Time(timestamp);

                    // Marcar que se ha recibido el primer heartbeat

heartbeatReceived = true;
                }
                else if(strcmp(tipo, "comando") == 0) {
                    comando = doc["comando"];
                    if(comando == 1)      { Serial.println("Robot derecho");     moverAdelante(); tiempoMotorActivado = millis();}
                    else if(comando == 2) {Serial.println("Robot atras"); moverAtras(); tiempoMotorActivado = millis(); }
                    else if(comando == 3) {Serial.println("Robot derecha");      girarDerecha(); tiempoMotorActivado = millis(); }
                    else if(comando == 4) {Serial.println("Izquierda"); girarIzquierda(); tiempoMotorActivado = millis(); }
                    else if(comando == 5){

frenarMotores();

Serial.println("Paro");

                    }
                    else if(comando == 6){
                    float distancia = medicion();

DynamicJsonDocument doc(256);

doc["distancia"] = distancia;
                    // Serializar el JSON y enviarlo
                    String jsonString;

serializeJson(doc, jsonString);
                    client->text(jsonString);
                    }
                    else if(comando == 7){
```

```
                    int
anguloServomotor  =
doc["angulo"];

miServo.write(anguloServomotor);

Serial.print("Angulo: ");

Serial.println(anguloServomotor)
;
                  }
                  // Nuevo
comando para evaluar distancias
con el servomotor
                  else if
(comando == 8) {

Serial.println("Comando
recibido: evaluar distancias");
                    int posicion
= 0;

miServo.write(posicion);  //
Mover servomotor a la posición

//delay(500);
// Esperar para que el servo se
estabilice
                    float
distancia = medicion();

DynamicJsonDocument doc(256);

doc["distancia"] = distancia;
                    //
Serializar el JSON y enviarlo
                    String
jsonString;

serializeJson(doc, jsonString);
                    client-
>text(jsonString);
                  }
                  else
if(comando == 11) tiempoBlink =
1000;
                  else
if(comando == 12) tiempoBlink =
500;
                  else
if(comando == 13) tiempoBlink =
100;
                  // Nuevo
comando para evaluar distancias
con el servomotor
                  else if
(comando == 90) {

Serial.println("Comando
recibido: evaluar distancias");
                    int posicion
= 90;

miServo.write(posicion);  //
Mover servomotor a la posición

//delay(500);
// Esperar para que el servo se
estabilice
                    float
distancia = medicion();

DynamicJsonDocument doc(256);

doc["distancia"] = distancia;
                    //
Serializar el JSON y enviarlo
                    String
jsonString;

serializeJson(doc, jsonString);
                    client-
>text(jsonString);
                  }
                  // Nuevo
comando para evaluar distancias
con el servomotor
                  else if
(comando == 180) {

Serial.println("Comando
recibido: evaluar distancias");
                    int posicion
= 180;

miServo.write(posicion);  //
Mover servomotor a la posición

//delay(500);
// Esperar para que el servo se
estabilice
                    float
distancia = medicion();

DynamicJsonDocument doc(256);
```

```
doc["distancia"] = distancia;
                    //
Serializar el JSON y enviarlo
                    String
jsonString;

serializeJson(doc, jsonString);
                    client-
>text(jsonString);
                  }
                }



            } else {
            // Si el mensaje no
es un JSON válido

Serial.println("Mensaje no es un
JSON válido, ignorando.");
             client-
>text("Error: Mensaje no es un
JSON válido.");
            }
        } else {
            Serial.println("No
data received.");
        }
    } else if (type ==
WS_EVT_CONNECT) {
        Serial.println("Client
connected");
    } else if (type ==
WS_EVT_DISCONNECT) {
        Serial.println("Client
disconnected");
    }
}

// Función para enviar los datos
del ADC o la señal senoidal en
formato JSON
void
sendAdcReading(AsyncWebSocketCli
ent *client) {
    if (heartbeatReceived) {
        float adcVoltage;
        float humedad =
dht.readHumidity();
        float temperatura =
dht.readTemperature();
        float distancia =
medicion();
        if (isnan(humedad) ||
isnan(temperatura)) {

Serial.println("Error al leer
del sensor DHT!");
            return;  // Salir de
la función si hay un error en el
sensor
        }

        // Leer el valor del ADC
y convertir a voltaje
        int adcValue =
analogRead(ADC_PIN);
        adcVoltage = (adcValue /
4095.0) * 3.3;  // Convertir a
voltaje (0-3.3V)

        // Crear un objeto JSON
para enviar todas las lecturas
        DynamicJsonDocument
doc(256);
        doc["type"] =
"adc_reading";
        doc["temperature"] =
temperatura;
        doc["humidity"] =
humedad;
        doc["adc_value"] =
adcVoltage;
        doc["distancia"] =
distancia;
        // Serializar el JSON y
enviarlo
        String jsonString;
        serializeJson(doc,
jsonString);
        client-
>text(jsonString);

        // Log para ver los
datos enviados
        Serial.print("Enviando
datos: ");

Serial.println(jsonString);
    }
}

// Task para el parpadeo del LED
en el core 1
void blinkTask(void *param) {
    long int ta = millis();
```

```
    long int tb = 0;
    bool estadoLed = false;
    //aqui
    pinMode(blinkPin, OUTPUT);
    while (1) {
        tb = millis();
        if((tb-ta)>tiempoBlink){
          ta = millis();
          digitalWrite(blinkPin,
estadoLed);
          estadoLed =
!estadoLed;
        }
        if(comando >= 0 ){
          tiempoMotor =
millis();
            //Hay un comando
para procesar.
            if(comando == 1){
                if((tiempoMotor-
tiempoMotorActivado)>tiempoAdela
nte){

frenarMotores();
                  comando = -1;
                }
            }
            else if(comando ==
2){
              if((tiempoMotor-
tiempoMotorActivado)>tiempoAtras
){

frenarMotores();
                  comando = -1;
                }
            }
            else if(comando ==
3){
              if((tiempoMotor-
tiempoMotorActivado)>tiempoDerec
ha){

frenarMotores();
                  comando = -1;
                }
            }
            else if(comando ==
4){
              if((tiempoMotor-
tiempoMotorActivado)>tiempoIzqui
erda){

frenarMotores();
                  comando = -1;
                }
            }



        }

    }
}

// Task para manejar el loop del
WebSocket y ADC en el core 0
void wsAdcTask(void *param) {
    while (1) {
        // Mantén limpios los
clientes inactivos del WebSocket
        ws.cleanupClients();

        // Verificar si ya se ha
recibido el primer heartbeat y
enviar lecturas cada 100 ms
        if (heartbeatReceived &&
(millis() - lastAdcSendTime >=
tiempoMuestreo)) {
            lastAdcSendTime =
millis();  // Actualizar el
tiempo de la última lectura
            // Enviar lectura a
los clientes conectados
            for
(AsyncWebSocketClient *client :
ws.getClients()) {
                if (client-
>status() == WS_CONNECTED) {

sendAdcReading(client);
                }
            }
        }
        vTaskDelay(10 /
portTICK_PERIOD_MS);  // Pequeño
retardo para no saturar el
procesador
    }
}

void setup() {
    // Inicializa el puerto
serial
    Serial.begin(115200);

    // Conéctate a la red WiFi
```

```
    WiFi.begin(ssid, password);
    while (WiFi.status() !=
WL_CONNECTED) {
        delay(1000);

Serial.println("Conectando a
WiFi...");
    }
    Serial.println("Conectado a
WiFi");

    // Imprime la IP asignada
por el router
    Serial.print("IP asignada:
");

Serial.println(WiFi.localIP());

    // Configura el WebSocket y
asigna el manejador de eventos

ws.onEvent(onWebSocketMessage);
    server.addHandler(&ws);

    // Inicia el servidor
    server.begin();
    Serial.println("Servidor
WebSocket iniciado");

    // Configurar el pin ADC
    pinMode(adcPin, INPUT);
    startTime = millis();  //
Inicializa el tiempo para la
señal senoidal

    // Crear la tarea para el
WebSocket y ADC en Core 0

    xTaskCreatePinnedToCore(
        wsAdcTask,        //
Función de la tarea
        "WS ADC Task",    //
Nombre de la tarea
        4096,             //
Tamaño de la pila
        NULL,             //
Parámetro de entrada (null en
este caso)
        1,                //
Prioridad
        NULL,             //
Puntero a la tarea (no lo
usamos)
        0);               //
Ejecutar en el core 0

    // Crear la tarea para el
blink en Core 1
    xTaskCreatePinnedToCore(
        blinkTask,        //
Función de la tarea
        "Blink Task",     //
Nombre de la tarea
        2048,             //
Tamaño de la pila
        NULL,             //
Parámetro de entrada (null en
este caso)
        1,                //
Prioridad
        NULL,             //
Puntero a la tarea (no lo
usamos)
        1);               //
Ejecutar en el core 1

    setupMotores();
    frenarMotores();
    dht.begin();
    SensorDistanciaSetup();
    miServo.attach(pinServo);


}

void loop() {
    // No hacemos nada en el
loop, ya que todas las tareas
corren en FreeRTOS
}
```

Controles motores

```
// Definir pines de control para los motores
const int motor1PinA = 12; // Pin de control 1 para Motor 1
const int motor1PinB = 11; // Pin de control 2 para Motor 1
const int motor1PWM = 4;   // Pin PWM para Motor 1

const int motor2PinA = 10; // Pin de control 1 para Motor 2
const int motor2PinB = 9;  // Pin de control 2 para Motor 2
const int motor2PWM = 5;   // Pin PWM para Motor 2



// Función para configurar los pines de control de un motor
void establecerDireccionMotor(int pinA, int pinB, bool adelante) {
  if (adelante) {
    digitalWrite(pinA, HIGH);
    digitalWrite(pinB, LOW);
  } else {
    digitalWrite(pinA, LOW);
    digitalWrite(pinB, HIGH);
  }
}

// Función para establecer la dirección de ambos motores hacia
adelante
void moverAdelante() {
  Serial.println("OK adelante");
  establecerDireccionMotor(motor1PinA, motor1PinB, true); // Motor 1
adelante
  establecerDireccionMotor(motor2PinA, motor2PinB, true); // Motor 2
adelante
}

// Función para establecer la dirección de ambos motores hacia atrás
void moverAtras() {
  Serial.println("OK atras");
  establecerDireccionMotor(motor1PinA, motor1PinB, false); // Motor 1
atrás
  establecerDireccionMotor(motor2PinA, motor2PinB, false); // Motor 2
atrás
}

// Función para girar a la izquierda
void girarIzquierda() {
  Serial.println("OK izquierda");
  establecerDireccionMotor(motor1PinA, motor1PinB, false); // Motor 1
atrás
  establecerDireccionMotor(motor2PinA, motor2PinB, true);  // Motor 2
adelante
}

// Función para girar a la derecha
void girarDerecha() {
  Serial.println("OK derecha");
```

```
  establecerDireccionMotor(motor1PinA, motor1PinB, true);  // Motor 1
adelante
  establecerDireccionMotor(motor2PinA, motor2PinB, false); // Motor 2
atrás
}

// Función independiente para controlar la velocidad de ambos motores
void establecerVelocidad(int velocidad) {
  // Convertir el valor de 0-100 a 0-255 (resolución de 8 bits)
  int valorPWM = map(velocidad, 0, 100, 0, 255);

  analogWrite(motor1PWM, valorPWM);
  analogWrite(motor1PWM, valorPWM);
}

// Función para detener/frenar los motores
void frenarMotores() {
  Serial.println("Freno");
  // Para frenar, establecer ambos pines A y B en HIGH
  digitalWrite(motor1PinA, HIGH);
  digitalWrite(motor1PinB, HIGH);
  digitalWrite(motor2PinA, HIGH);
  digitalWrite(motor2PinB, HIGH);

  // Detener el PWM usando ledcWrite en los canales correctos
  analogWrite(motor1PWM, 0);
  analogWrite(motor1PWM, 0);
}

void setupMotores() {
  // Inicializar los pines de los motores como salida
  pinMode(motor1PinA, OUTPUT);
  pinMode(motor1PinB, OUTPUT);
  pinMode(motor1PWM, OUTPUT);

  pinMode(motor2PinA, OUTPUT);
  pinMode(motor2PinB, OUTPUT);
  pinMode(motor2PWM, OUTPUT);

  pinMode(motor1PWM, OUTPUT);
  pinMode(motor2PWM, OUTPUT);
  analogWrite(motor1PWM, 0);
  analogWrite(motor1PWM, 0);

}
```

Ultrasonico

```
#define TRIG_PIN 13 // Pin de TRIG conectado al HC-SR04
#define ECHO_PIN 14 // Pin de ECHO conectado al HC-SR04

void SensorDistanciaSetup() {
  pinMode(TRIG_PIN, OUTPUT); // Configura el pin TRIG como salida
```

```
  pinMode(ECHO_PIN, INPUT);  // Configura el pin ECHO como entrada
}

float medicion(){
  long duration;
  float distance;

  // Envía un pulso ultrasónico
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Mide el tiempo del pulso de regreso
  duration = pulseIn(ECHO_PIN, HIGH);

  // Calcula la distancia en centímetros
  distance = (duration * 0.034) / 2;
  return distance;
}
```

Fotos:

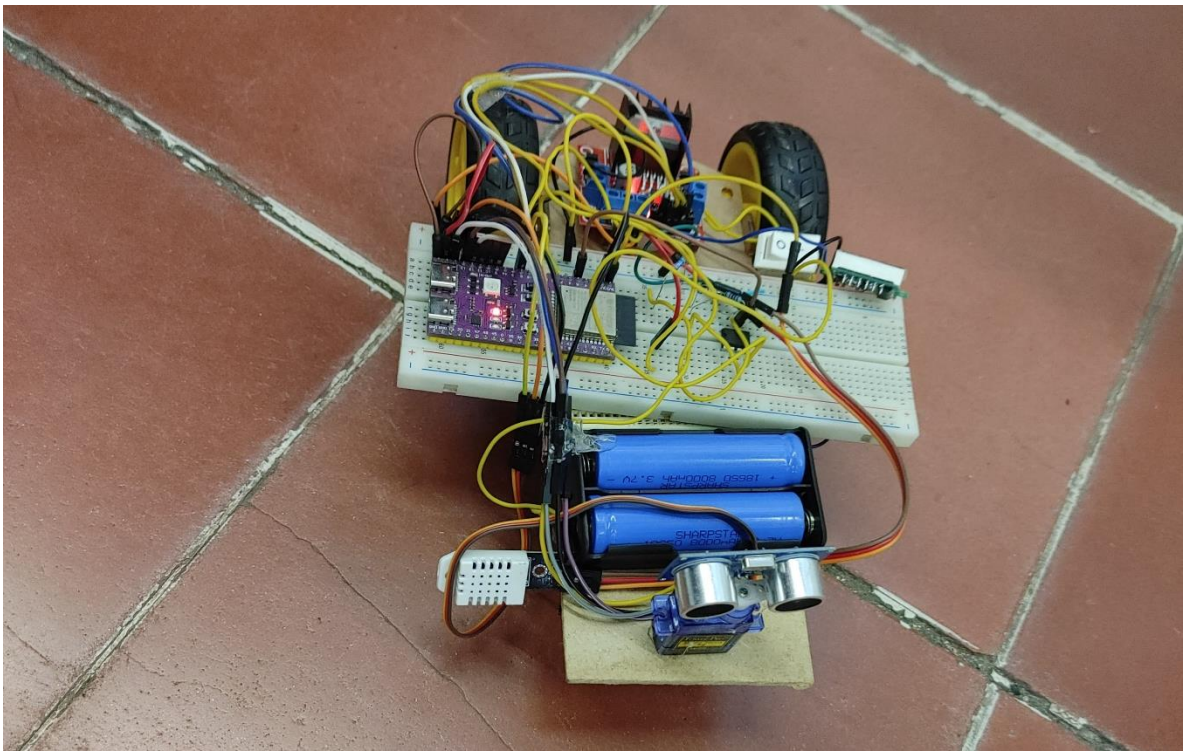

Ilustración 3 Carrito funcionando

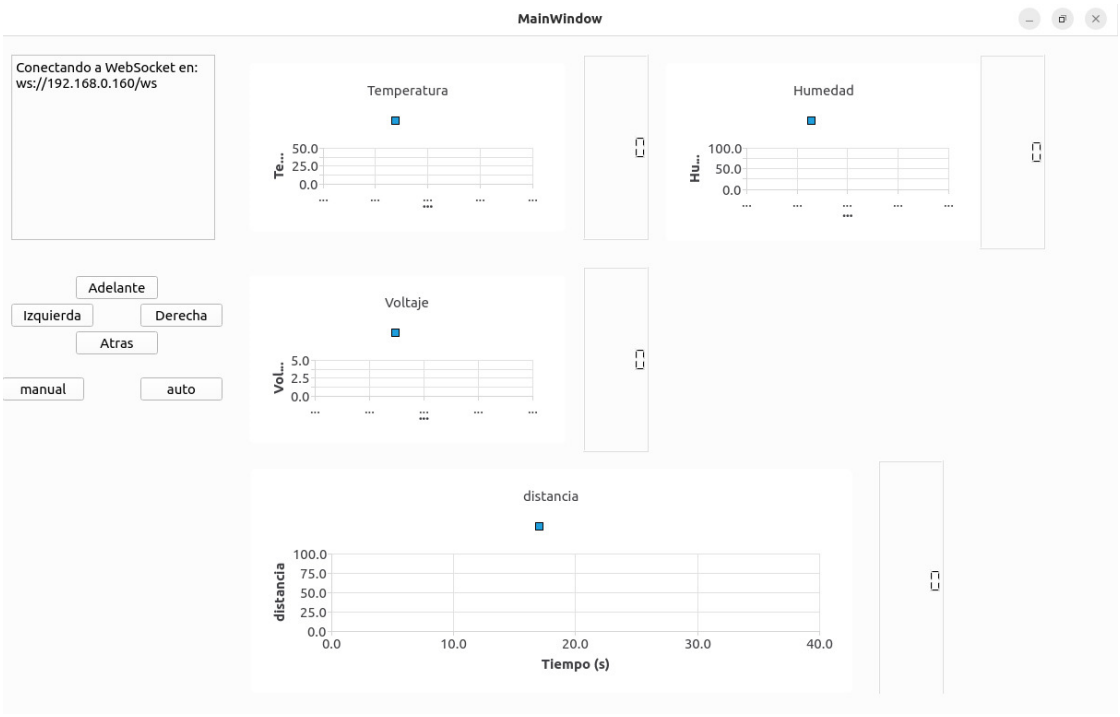

Ilustración 4 Interfaz del proyecto

Ilustración 5 DB-base de datos (Estructura)

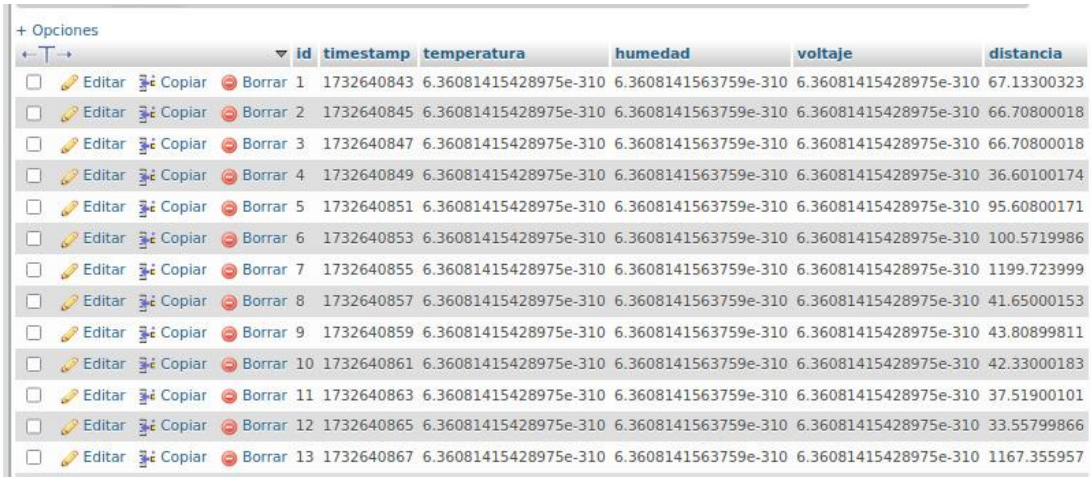

Ilustración 6 Demostracion de llenado DB

Conclusión:

En este proyecto, integramos tecnologías de software y hardware para desarrollar un sistema autónomo de navegación basado en un carrito controlado por Qt y Arduino. A lo largo del proceso, abordamos múltiples aspectos técnicos y metodológicos, destacando los siguientes:

1. **Diseño del sistema**:
    - Utilizamos un servomotor, un sensor de distancia, y un controlador de motores para permitir que el carrito evalúe su entorno y tome decisiones de movimiento basadas en distancias medidas en tres posiciones: izquierda, centro y derecha.
2. **Programación y control**:
    - El código en **Arduino** se diseñó para controlar el hardware, como el servomotor y los motores, ejecutando comandos recibidos desde la interfaz Qt.
    - En **Qt**, implementamos una máquina de estados para manejar la lógica de navegación, gestionando comandos hacia el carrito y procesando datos recibidos del sensor de distancia.
3. **Integración hardware-software**:
    - Mediante comunicación WebSocket, logramos que el sistema Qt interactúe en tiempo real con el microcontrolador, garantizando sincronización entre el análisis de datos y las decisiones de movimiento del carrito.
4. **Metodología aplicada**:
    - Se definieron etapas claras: elección de materiales, desarrollo de código, pruebas unitarias, y ajustes iterativos para optimizar el comportamiento del carrito en distintos escenarios.

Este proyecto demuestra cómo combinar programación en plataformas robustas como Qt con la flexibilidad del ecosistema Arduino permite crear soluciones eficientes para sistemas autónomos. Además, la estructura basada en una máquina de estados facilita la gestión lógica del sistema, mientras que la comunicación en tiempo real asegura una operación fluida y confiable.

Aunque este proyecto no funciona del todo como se esperaba, se aprenden bastantes cosas como las mencionadas anteriormente, destacando en mi caso la integración de hardware-software, la cual es indispensable destacar en mi desarrollo y crecimiento de este proyecto, que aunque me toco en bina, se que hubiera podido hacerlo con un poco de ayuda.