Logbook

07/10/2024

I had 2 ideas the reverse car parking system and a security system:

Security system:

- Detect movement
- Switch on during nighttime or on command
- Sound loud noise and send notification to user
- Use some kind of sensor to detect night time or set up a specific time it turns on
- Motion and temperature sensor to detect movement
- Turn off on command or daytime

I decided to go with reverse car parking system

14/10/2024

Aims and objectives

Objective: To design and implement a reverse car parking sensor that integrates real-time distance measurement, live video feed accessible through a mobile app, and an audible buzzer for proximity alerts.

Success evaluation

Success will be measured by the system's ability to detect obstacles accurately, provide a clear video feed, and respond in real-time with audible feedback during parking.

Planning

Research hardware components like ultrasonic sensors, camera module, microcontroller (ESP32), and a mobile app framework.

Understanding communication protocols (Wi-Fi).

Analysis

Opted for a Wi-Fi connection instead of Bluetooth because Wi-Fi provided the required bandwidth for video streaming.

Observations

An alternative solution is to use Bluetooth Low Energy (BLE) for distance data while maintaining Wi-Fi for video. Decided Wi-Fi for simplicity and integration.

Reflection

This device can significantly improve driver confidence and safety while reverse parking.

21/10/2024

Aims and objectives

Objective: Successfully upload and execute an example code on an ESP32.

The primary issue I identified was during the upload process. With Keith guidance, holding the reset button while connecting the cable and initiating the upload process resolved the problem.

Questions

The unusual behaviour, I think, comes from the ESP32 crashing and going into an infinite loop while the code is uploading, thus necessitating manual intervention.

Hypothesis

Might be an hardware defect

Observations

An alternative Solutions is to replace the ESP32 with a new one, assuming the issue might be a hardware defect.

Discussion

Thanks to Keith suggestion of holding down the reset button during code upload. This advice was beneficial in resolving the issue.

28/10/2024

Aims and objectives

Successfully calibrate the ultrasonic sensor and implement code to calculate distances.

Evaluation of success

Achieve accurate and reliable distance data

Planning

Learn the pin layout and functionality of the ultrasonic sensor.

Set up the hardware on the breadboard.

Write and upload the code to calculate and display distance readings.

Test and calibrate the sensor for accuracy.

Encountered issues

Initial confusion due to difficulty in identifying the VCC, Trig, Echo, and GND pins, leading to improper connections.

The lack of visual indicators made it challenging to confirm if the sensor was powered and operational.

Hypothesis

Once the sensor is correctly connected and coded, it will provide reliable distance measurements by calculating the time taken for the ultrasonic pulse to return.

Misplacement or incorrect pin assignments will lead to inaccurate or no readings.

Hardware setup

Connected the ultrasonic sensor to the breadboard and linked it to the ESP32 and assigned VCC and GND to power rails and Trig and Echo to pins.

Code implementation

Uploaded code to send a pulse from the Trig pin and measure the time taken on the Echo pin.

Formula applied: Distance (cm) = (Duration / 2) * Speed of Sound.

Testing

Placed objects and compared sensor readings to actual distances.

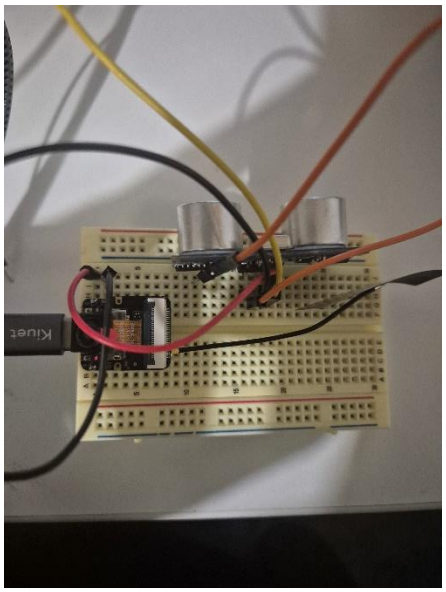
Results

Outcome

The sensor was successfully calibrated, and a functional distance calculation program was implemented. Initial challenges were resolved with the help of Keith guidance and experimentation in the hackspace.

Keith’s helpful suggestions in resolving pin assignment issues and understanding the sensor's functionality and guidance from hackspace peers helped me understand the hardware setup.

```
// Define ultrasonic sensor pins
#define TRIG_PIN D1
#define ECHO_PIN D0

void setup() {
  Serial.begin(115200);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  Serial.println("Ultrasonic sensor test initialised.");
}

void loop() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  int distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(500);
}
```



04/11/2024

Aims and objectives

Establish internet connectivity for the ESP32.

Set up a web server to display real-time distance measurements from the ultrasonic sensor.

Test integration of ultrasonic sensor data and the web server interface.

Evaluation of success

The internet connection and server setup were established, and the ultrasonic sensor's data was successfully integrated and displayed.

Planning

Configure Wi-Fi connectivity for the ESP32.

Write and upload code for setting up an HTTP server.

Incorporate the ultrasonic sensor code into the server logic to transmit real-time distance readings.

Design

I encountered issues combining the ultrasonic sensor code with the web server logic.

I find my solution in the assignment 1 brief.

Internet setup

Configured ESP32 with Wi-Fi SSID and password using the WiFi.h library.

Wrote code to verify successful connection and output the assigned IP address.

Server setup

Initialised a basic HTTP server

Displayed a simple webpage with text output for distance sensor data.

Sensor integration

Integrated ultrasonic sensor code to send real-time distance readings to the web server.

Tested the sensor's performance by placing objects at various distances.

Outcome

The ESP32 successfully connected to Wi-Fi and hosted a functional web server.

Real-time distance measurements from the ultrasonic sensor were displayed on the web interface.

The ESP32 cam functionality was delayed due to unresolved hardware or software issues.

Observations

An alternative solution was to use an app for transmit data from the ESP32 to the user.

My solution: Continue with the ESP32 for both server and sensor functions, because I thought it was more user-friendly

```
void setup() {
  Serial.begin(115200);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Connect to WiFi
  Serial.print("Connecting to WiFi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nConnected to WiFi!");
  Serial.print("Server IP address: ");
  Serial.println(WiFi.localIP());

  // Start the camera
  startCamera();

  // Start the server
  server.begin();
  Serial.println("Server started.");
}

void loop() {
  WiFiClient client = server.available();
  if (!client) return;

  Serial.println("New client connected");
```

```
  String request = client.readStringUntil('\r');
  client.flush();

  if (request.indexOf("/distance") != -1) {
    int distance = getDistance()
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/plain");
    client.println();
    client.print(distance);
  } else if (request.indexOf("/stream") != -1) {
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: multipart/x-mixed-replace; boundary=frame");
    client.println();

    while (client.connected()) {
      camera_fb_t* fb = esp_camera_fb_get();
      if (!fb) {
        Serial.println("Camera capture failed");
        break;
      }

      client.println("--frame");
      client.println("Content-Type: image/jpeg");
      client.println("Content-Length: " + String(fb->len));
      client.println();
      client.write(fb->buf, fb->len);
      client.println();
      esp_camera_fb_return(fb);

      delay(50);
    }
  } else {
```

```
  } else {
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println();
    client.println("<!DOCTYPE HTML>");
    client.println("<html>");
    client.println("<h1>Car Parking Sensor Web Server</h1>");
    client.println("<p>Distance: <span id=\"distance\">Loading...</span> cm</p>");
    client.println("<p><a href=\"/stream\">Click here to view the live video feed</a></p>");
    client.println("<script>");
    client.println("setInterval(() => {");
    client.println("  fetch('/distance').then(response => response.text()).then(data => {");
    client.println("    document.getElementById('distance').innerText = data;");
    client.println("  });");
    client.println("}, 1000);");
    client.println("</script>");
    client.println("</html>");
  }

  Serial.println("Request handled.");
```

11/11/2024

Aims and objectives

Identify and correctly initialise the camera pins for the ESP32.

Successfully implement camera functionality into the project code.

Evaluation of Success:

Objective achieved: After troubleshooting with guidance from Keith and leveraging online resources, the camera was successfully initialized and integrated into the code. Stability testing showed the camera functioned as intended.

Planning

Understand the camera module's pinout and its connection requirements.

Configure the camera's settings within the ESP32.

Design

I encountered issues identifying the correct camera pins and mapping them to the ESP32 board.

Consulted with Keith to verify the pin mappings and ensure connections were correctly established as well as using official documentation and community forums to cross-reference camera settings and initialisation processes.

Pin mapping

Used datasheets and example projects to identify and connect the camera's pins to the ESP32.

Code testing

Wrote a minimal test script to initialise the camera and capture an image and debugged errors by adjusting the configuration parameters for the camera.

Outcome

The ESP32 successfully initialised the camera after correcting pin assignments and refining the code.

The camera captured images and streamed them to a web server interface.

```
// Initialise camera
void startCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_VGA;
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 20;
  config.fb_count = 1;

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera initialisation failed", err);
  } else {
    Serial.println("Camera initialised successfully");
  }
}
```

18/11/2024

Didn’t do much just combining all my features into one code and decide not to implement the buzzer options as it would work outside of the car and wouldn’t be too helpful to the driver

25/11/2024

Aims and objectives

Enhance the project’s security by implementing HTTPS for secure communication.

Use OpenSSL to generate a custom certificate and security key.

Configure the web server to run on port 443 for encrypted communication.

Evaluate the implementation and identify further improvements for production readiness.

Evaluation of success

Partial success: Custom certificates were generated and configured, but the server failed to recognise them during runtime.

Planning

Research the process of enabling HTTPS on the web server, including certificate generation and server configuration.

Solutions

Consulted OpenSSL documentation and community forums for troubleshooting guidance.

Questions

Despite correct syntax and paths, the server rejected the certificate and key files

Explanation

This issue was likely caused by either an incompatible certificate format, incorrect permissions, or errors in the server's HTTPS configuration settings.

Hypothesis

For production, using a validated certificate will provide seamless integration and ensure compliance with industry standards.

Certificate generation

Used OpenSSL to generate a self-signed certificate and private key.

Server configuration

Configured the ESP32 web server to use the generated certificate and listen on port 443 for HTTPS communication.

Results

The certificate and key were successfully generated, but the server failed to recognise them during runtime.

Justification

Using a self-signed certificate was sufficient for development but lacked compatibility for broader testing and deployment.

```
// Server certificate and private key
const char* server_cert = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDmTCCAoGgAwIBAgIUG83T2ks1nBfZK4TM/ozzxAvUaUAwDQYJKoZIhvcNAQEL
BQAwXDELMAkGA1UEBhMCdWsxEzARBgNVBAgMCmNhbnRlcmJ1cnkxGzAZBgNVBAoM
EnVuaXZlcnNpdHkgb2Yga2VudDEbMBkGA1UEAwwSY2FyIHBhcmtpbmcgc2Vuc29y
MB4XDTI0MTIwOTIzMjkzOVoXDTI1MTIwOTIzMjkzOVowXDELMAkGA1UEBhMCdWsx
EzARBgNVBAgMCmNhbnRlcmJ1cnkxGzAZBgNVBAoMEnVuaXZlcnNpdHkgb2Yga2Vu
dDEbMBkGA1UEAwwSY2FyIHBhcmtpbmcgc2Vuc29yMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAooyviHkk7uHqAqfm83hWC6xDQ9zx85qQLLDwfGr3a2vO
Zemg0yZMHD9D2BPjQG9Cqw4N7AtHj4zZ0yvVuSeNpJEzU3+cmxIKqY2E7hBhaN09
Xe9CeVFRUJvGgzjRZIMmO2BeBDpeidQv/jlaqk/j6nuaOW9w20jCNXpXfY8/K6Wd
OglRMPJvGgsHXUZz7zbBsjFSIWnCku4ltPU7PLJUAlx4zb5dSsbBTThnKZ9ey6/J
B/ChodmLdh+iZLavx0Wn18l5zVPbciLHoB4DQaxhhkOyIj9Ky2Hu3M2dTxSjkfq8
6aOaK9su85CT7Jygt6zKXRxBchp/eLzHry7n3HfzvQIDAQABo1MwUTAdBgNVHQ4E
FgQUpXpqCma3g1B2Eq3O/CRYbMPN2g0wHwYDVR0jBBgwFoAUpXpqCma3g1B2Eq3O
/CRYbMPN2g0wDwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAAG6I
LMrayEXApFu+IZses8s1EiKiBDUxRCpxhSEogiQV647iKgQvuvTczfqgXDUyjFoB
lSTQRU/hXzJLs+fRz6SkY0fpPvr46TdCCsPKk4NZXMSiBb2BheeMCMEh+RmZwB50
izGx2osXDX3DNbvawbxlvKc1uKwU8PZ+BrjjsBLdSGkMdRTEQ0KovDZ0Z0/er30X
N0dW/c8r8rM36ajFKYEs1Rg23jDDs+xrW1CQ0YMAv+88hjn7W34S2cNYRQAMrZ20
jTU9jP25imR0ui+Oov7aDEHP0JBtSJISoLe9vAbTuaqMAB2gkQ/b44P32bqgD15+
tbIJ4PqzIkerSabBSw==
-----END CERTIFICATE-----
)EOF";

const char* server_key = R"EOF(
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQCijK+IeSTu4eoC
p+bzeFYLrEND3PHzmpAssPB8avdra85l6aDTJkwcP0PYE+NAb0KrDg3sC0ePjNnT
K9W5J42kkTNTf5ybEgqpjYTuEGFo3T1d70J5UVFQm8aDONFkgyY7YF4EOl6J1C/+
OVqqT+Pqe5o5b3DbSMI1eld9jz8rpZ06CVEw8m8aCwddRnPvNsGyMVIhacKS7iW0
9Ts8slQCXHjNvl1KxsFNOGcpn17Lr8kH8KGh2Yt2H6Jktq/HRafXyXnNU9tyIseg
HgNBrGGGQ7IiP0rLYe7czZ1PFKOR+rzpo5or2y7zkJPsnKC3rMpdHEFyGn94vMev
Lufcd/O9AgMBAAECggEAIcPilJFBnEOeVgXlog6NpSgX4ulq8wTYzQfGNLoNgiln
OuFAcIwEhFuZ7rEW5CQYb6rgNneTWlzSRJzW7Pqr4BLvW/lmoQmq4FSO19rcOSm3
3SLdppC/OOTa5Bgx2tSpZO0cKnmreKr0ezUTwcwNloAtZZ/iDBL9kJlEwka2wwxz
FhJJGu2kEvjC3gtIUS4XQfrRb2nA5ztL1xY+dl/dl1yl2Tz4ZR5ErVoa24WO5pYp
az7ZRCgG8M0wo7623FFM49psGlJqlBF8IjKpJQzQdWDsBu3ppY3JeJwzBc881a4X
XLUwEwfsQgwIzetQLkP3SmNdJvJccNs/zNs2A0MIcQKBgQDl0FO9zSfidjD4e5zi
3l5e4VaRPE9dE4ylfdCMms/6Xmsw/G3LKKiC0D+YRYIK0HkxcmiGlTnOg0TkhRrn
A0CGXDta19H9mLTb/wV48zmKYfnDaiaqcdMu9Lmm1BrsNN7zvxIBQcOS0ctZfUeo
CG8Vwhx4DM8D4k5VKaVSqSuUCQKBgQC1EkJ6LYjYWw+u57b3vbXvyQCGUD2CNBeD
Apdtsmq5QkP6+UThkDew4Dpi3Tli6N4wu485G9mzxmzuiqtyW1SRamsFwYLwus3x
k9xxYGrbidjB2BgDhv3NINi+aduuo8m64Ny1QjY80e3LC499gt5Cppbm3dCh+UG+
yGN1WtgXFQKBgEuRW1f4E/tlfejXTNlU4Zc1Za/QMxCaEw9/9Ymmhmj/jyPGfZvm
PLL3I1JA2OgRcSiPREus2htMEuVJ13TLHvaPcX0HpqdLBX2pt/TzlaIpYWWejCRT
1WuFUV7INCojoIRp+dOr+lFW5xUAGwXn8A8piw72t0PBsJxUW2Z49jlxAoGAMJw4
UIiM5lh7+16llhjOEROLGoPMH3pQnZFCieaOFDIgAe6NeWi4Krnnssb1MxKFI9wW
dSRu635nnBqKMGcoaumA1ci/mFTuGN6zHZTe2xVEpbWXrMq+tCUPSv8sRNQfwhW/
6sFdwjIxfoMaAH3zoT0L4erGqdU6uzcczFKZFIkCgYADW4U/TXtokjwUGm93fWC3
+IxpZI+BmsAXLkU2ETbfxmtqZ3G4WvBRzRwzvUBzWtp+2ZKHfQkYdY6DorELui4S
3Nte5xx4JEtgjqDjHIb0PdgWUHmRoBLL0V0gm2Q5xCQcswDSDLXKF3NVXYoOcK0d
EoawgjQ6Lh9m8S+4JY8Bhw==
-----END PRIVATE KEY-----
)EOF";

// Global objects
WiFiClientSecure secureClient;
WiFiClient wifiClient;
PubSubClient client(wifiClient);
WiFiServer server(443);
```

02/12/2024

Aims and objectives

Explore the implementation of duty cycling on the ESP32 to optimise power consumption by utilising light sleep mode.

Maintain critical functionalities, such as connectivity and ultrasonic sensor operation, while minimising energy use during inactivity.

Evaluate the feasibility and reliability of duty cycling in the context of the project’s overall design.

Evaluation of success

Partial success: The functionality of light sleep mode was tested, but persistent server update issues and instability led to prioritising reliability over power optimisation. The decision was influenced by the fact that the device would ultimately be powered by the car's 12V system, making power savings less critical.

Planning

Research and understand the ESP32's light sleep mode and duty cycling capabilities.

Configure the device to enter light sleep mode after inactivity and wake up periodically to check for nearby objects using the ultrasonic sensor.

Test the implementation to ensure reliable operation, especially the transition between sleep and active states.

I encountered these challenges

Errors such as "No data received on Port COM 7" occurred frequently, causing instability during testing.

I tried to Adjust the code to improve compatibility between the duty cycling implementation and the server operation.

The only solution was to reset the ESP32 and re-uploaded the code multiple times.

I even explored online resources and documentation for insights into ESP32 light sleep and duty cycling behaviour, but couldn’t find any help.

Questions

The web server failed to update properly after implementing duty cycling, despite successful code uploads.

Explanation

The issues likely stemmed from improper timing configurations or conflicts between light sleep mode and the ESP32's web server connectivity or even hardware failure.

Implementation

Configured the ESP32 to enter light sleep mode after 30 seconds of inactivity.

Set the ultrasonic sensor to check for objects within a 100 cm range at regular intervals.

Designed the system to exit light sleep mode and fully activate if an object was detected.

Outcome

Light sleep mode successfully reduced power consumption during periods of inactivity.

However, the web server encountered frequent errors and failed to update reliably.

After troubleshooting and code adjustments, the ESP32 eventually resumed normal operation, but the risk of further instability remained.

Analysis

Discontinued the duty cycling implementation to prioritise reliability over additional power savings.

Opted to power the device directly through the car's 12V system, rendering power optimisation less critical.

An alternative solution

Maybe continue refining the duty cycling implementation but running the risk of frying the esp32

```
void enterLightSleep() {
  Serial.println("Entering light sleep...");
  esp_sleep_enable_timer_wakeup(5000000);
  esp_light_sleep_start();
  Serial.println("Woke up from light sleep!");
}

if (distance < 200) {
    // Object detected within 200 cm, activate full functionality
    Serial.println("Object detected! Activating camera and server...");
    // Start your server or other active functionalities here
    delay(10000);  // Keep the system awake for 10 seconds for demonstration
  } else {
    // No object detected, enter light sleep
    Serial.println("No object detected. Entering light sleep...");
    esp_light_sleep_start();
    Serial.println("Woke up from light sleep");  }}
```

Aims and objectives

Implement MQTT communication protocol to enable seamless data exchange between the ESP32 and a remote server or application.

Respond to project feedback by addressing identified weaknesses and incorporating improvements.

Evaluation of success

Partial success: MQTT communication was established.

Planning

Research the MQTT protocol and choose a suitable library for the ESP32.

Set up a test broker (MQTTX broker) and configure basic publish/subscribe functionality.

Implement a lightweight MQTT client on the ESP32 and test communication with the broker.

Extend functionality to include specific project requirements, such as publishing sensor data and receiving commands.

Questions

Published messages occasionally failed to arrive at the broker or were delayed.

Explanation

Connection issues were likely caused by unstable Wi-Fi or incorrect broker settings.

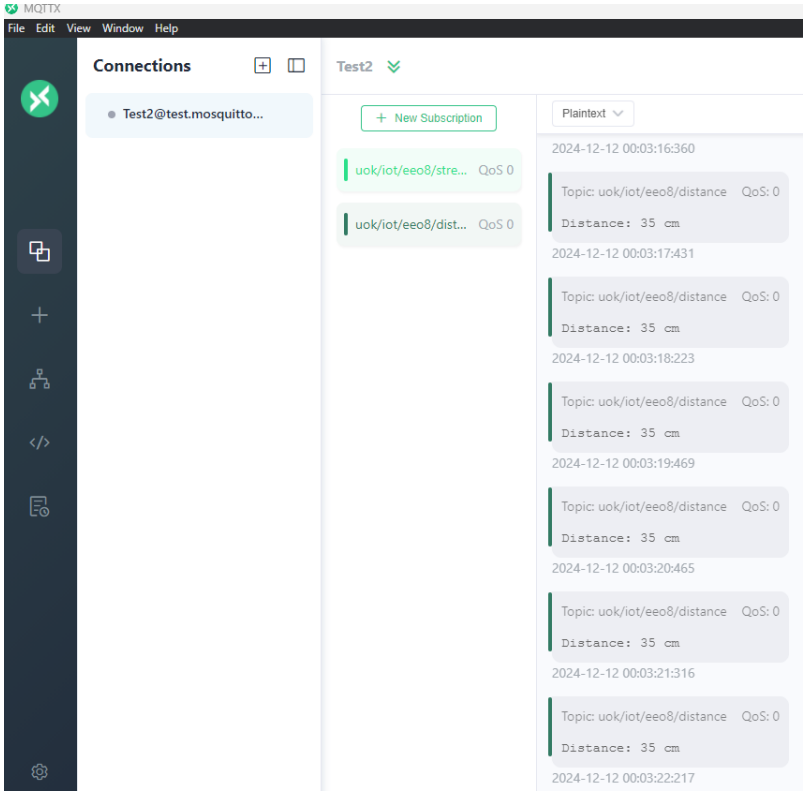
Broker Setup

Configured MQTTX as a local MQTT broker on a laptop.

Outcome

The ESP32 successfully communicated with both local.

Chosen Solution: was to send distance data over the local broker as well as web server. Because the local broker provides connection protocols.



I combined all these features and created my final piece of code for the presentation

```
// Libraries
#include <WiFi.h>
#include <PubSubClient.h>
#include "esp_camera.h"
#include <WiFiClientSecure.h>

// WIFI connection
#define WIFI_SSID "BTB-PJARMX"
#define WIFI_PASSWORD "Pablosmellz64"

// MQTT connection
#define MQTT_BROKER "test.mosquitto.org"
#define MQTT_PORT (1883)
#define MQTT_PUBLIC_TOPIC "uok/iot/eeo8/distance"
```

```
#define MQTT_SUBSCRIBE_TOPIC "uok/iot/eeo8/stream"

// Ultrasonic sensor pins
#define TRIG_PIN D1
#define ECHO_PIN D0

// Camera pins
#define CAMERA_MODEL_XIAO_ESP32S3
#include "camera_pins.h"

// Server certificate and private key
const char* server_cert = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDmTCCAoGgAwIBAgIUG83T2ks1nBfZK4TM/ozzxAvUaUAwDQYJKoZIhvcNAQEL
BQAwXDELMAkGA1UEBhMCdWsxEzARBgNVBAgMCmNhbnRlcmJ1cnkxGzAZBgNVBAoM
EnVuaXZlcnNpdHkgb2Yga2VudDEbMBkGA1UEAwwSY2FyIHBhcmtpbmcgc2Vuc29y
MB4XDTI0MTIwOTIzMjkzOVoXDTI1MTIwOTIzMjkzOVowXDELMAkGA1UEBhMCdWsx
EzARBgNVBAgMCmNhbnRlcmJ1cnkxGzAZBgNVBAoMEnVuaXZlcnNpdHkgb2Yga2Vu
dDEbMBkGA1UEAwwSY2FyIHBhcmtpbmcgc2Vuc29yMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAooyviHkk7uHqAqfm83hWC6xDQ9zx85qQLLDwfGr3a2vO
Zemg0yZMHD9D2BPjQG9Cqw4N7AtHj4zZ0yvVuSeNpJEzU3+cmxIKqY2E7hBhaN09
Xe9CeVFRUJvGgzjRZIMmO2BeBDpeidQv/jlaqk/j6nuaOW9w20jCNXpXfY8/K6Wd
OglRMPJvGgsHXUZz7zbBsjFSIWnCku4ltPU7PLJUAlx4zb5dSsbBTThnKZ9ey6/J
B/ChodmLdh+iZLavx0Wn18l5zVPbciLHoB4DQaxhhkOyIj9Ky2Hu3M2dTxSjkfq8
6aOaK9su85CT7Jygt6zKXRxBchp/eLzHry7n3HfzvQIDAQABo1MwUTAdBgNVHQ4E
FgQUpXpqCma3g1B2Eq3O/CRYbMPN2g0wHwYDVR0jBBgwFoAUpXpqCma3g1B2Eq3O
/CRYbMPN2g0wDwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAAG6I
LMrayEXApFu+IZses8s1EiKiBDUxRCpxhSEogiQV647iKgQvuvTczfqgXDUyjFoB
l5TQRU/hXzJLs+fRz6SkY0fpPvr46TdCCsPKk4NZXMSiBb2BheeMCMEh+RmZwB50
izGx2osXDX3DNbvawbxlvKc1uKwU8PZ+BrjjsBLdSGkMdRTEQ0KovDZ0Z0/er30X
N0dW/c8r8rM36ajFKYEs1Rg23jDDs+xrW1CQ0YMAv+88hjn7W34S2cNYRQAMrZ20
```

jTU9jP25imR0ui+Oov7aDEHP0JBtSJISoLe9vAbTuaqMAB2gkQ/b44P32bqgD15+

tbIJ4PqzIkerSabBSw==

-----END CERTIFICATE-----

)EOF";

const char* server_key = R"EOF(

-----BEGIN PRIVATE KEY-----

MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQCijK+IeSTu4eoC

p+bzeFYLrEND3PHzmpAssPB8avdra85l6aDTJkwcP0PYE+NAb0KrDg3sC0ePjNnT

K9W5J42kkTNTf5ybEgqpjYTuEGFo3T1d70J5UVFQm8aDONFkgyY7YF4EOl6J1C/+

OVqqT+Pqe5o5b3DbSMI1eld9jz8rpZ06CVEw8m8aCwddRnPvNsGyMVIhacKS7iW0

9Ts8slQCXHjNvl1KxsFNOGcpn17Lr8kH8KGh2Yt2H6Jktq/HRafXyXnNU9tyIseg

HgNBrGGGQ7IiP0rLYe7czZ1PFKOR+rzpo5or2y7zkJPsnKC3rMpdHEFyGn94vMev

Lufcd/O9AgMBAAECggEAIcPilJFBnEOeVgXlog6NpSgX4ulq8wTYzQfGNLoNgiln

OuFAcIwEhFuZ7rEW5CQYb6rgNneTWlzSRJzW7Pqr4BLvW/lmoQmq4FSO19rcOSm3

3SLdppC/OOTa5Bgx2tSpZO0cKnmreKr0ezUTwcwNloAtZZ/iDBL9kJlEwka2wwxz

FhJJGu2kEvjC3gtIUS4XQfrRb2nA5ztL1xY+dl/dl1yl2Tz4ZR5ErVoa24WO5pYp

az7ZRCgG8M0wo7623FFM49psGlJqlBF8IjKpJQzQdWDsBu3ppY3JeJwzBc881a4X

XLUwEwfsQgwIzetQLkP3SmNdJvJccNs/zNs2A0MIcQKBgQDl0FO9zSfidjD4e5zi

3l5e4VaRPE9dE4ylfdCMms/6Xmsw/G3LKKiC0D+YRYIK0HkxcmiGlTnOg0TkhRrn

A0CGXDta19H9mLTb/wV48zmKYfnDaiaqcdMu9Lmm1BrsNN7zvxIBQcOS0ctZfUeo

CG8Vwhx4DM8D4k5VKaVSqSuUCQKBgQC1EkJ6LYjYWw+u57b3vbXvyQCGUD2CNBeD

Apdtsmq5QkP6+UThkDew4Dpi3Tli6N4wu485G9mzxmzuiqtyW1SRamsFwYLwus3x

k9xxYGrbidjB2BgDhv3NINi+aduuo8m64Ny1QjY80e3LC499gt5Cppbm3dCh+UG+

yGN1WtgXFQKBgEuRW1f4E/tlfejXTNlU4Zc1Za/QMxCaEw9/9Ymmhmj/jyPGfZvm

PLL3I1JA2OgRcSiPREus2htMEuVJ13TLHvaPcX0HpqdLBX2pt/TzlaIpYWWejCRT

1WuFUV7INCojoIRp+dOr+lFW5xUAGwXn8A8piw72t0PBsJxUW2Z49jlxAoGAMJw4

UIiM5lh7+16llhjOEROLGoPMH3pQnZFCieaOFDIgAe6NeWi4KrnnssblMxKFI9wW

dSRu635nnBqKMGcoaumA1ci/mFTuGN6zHZTe2xVEpbWXrMq+tCUPSv8sRNQfwhW/

6sFdwjIxfoMaAH3zoT0L4erGqdU6uzcczFKZFIkCgYAdW4U/TXtokjwUGm93fWC3

+IxpZI+BmsAXLkU2ETbfxmtqZ3G4WvBRzRwzvUBzWtp+2ZKHfQkYdY6DorELui4S

```
3Nte5xx4JEtgjqDjHIb0PdgWUHmRoBLL0V0gm2Q5xCQcswDSDLXKF3NVXYoOcK0d
EoawgjQ6Lh9m8S+4JY8Bhw==
-----END PRIVATE KEY-----
)EOF";

// Initialise WiFi server on port 80
WiFiClientSecure secureClient;
WiFiClient wifiClient;
PubSubClient client(wifiClient);
WiFiServer server(443);

// Calculate the distance with ultrasonic sensor
int getDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  if (duration == 0) {
    Serial.println("No echo received");
  }

  int distance = duration * 0.034 / 2;
  return distance;
}

// Initialise camera
void startCamera() {
  camera_config_t config;
```

```
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;
config.frame_size = FRAMESIZE_VGA;
config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
config.fb_location = CAMERA_FB_IN_PSRAM;
config.jpeg_quality = 20;
config.fb_count = 1;

esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
  Serial.printf("Camera initialisation failed", err);
} else {
  Serial.println("Camera initialised successfully");
```

```
  }
}

void setup() {
  // Initialise Serial communication
  Serial.begin(115200);

  // Initialise ultrasonic sensor pins
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Connect to WiFi
  Serial.print("Connecting to " WIFI_SSID);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  // Initialise MQTT
  client.setServer(MQTT_BROKER, MQTT_PORT);

  while (!client.connected()) {
    if (client.connect(("ESP32-" + String(random(0xffff), HEX)).c_str())) {
      Serial.println("MQTT connected.");
      client.subscribe(MQTT_SUBSCRIBE_TOPIC);
      Serial.print("Subscribed to: ");
```

```
    Serial.println(MQTT_SUBSCRIBE_TOPIC);
   } else {
    Serial.printf("Failed, rc=%d, try again in 5 seconds", client.state());
    delay(5000);
   }
  }

 // Load certificates
  secureClient.setCACert(server_cert);
  secureClient.setPrivateKey(server_key);

  // Start camera
  startCamera();

  // Start server
  server.begin();
  Serial.println("Secure server started on port 443.");
}

void loop() {
  client.loop();

  WiFiClient webClient = server.available();
  if (!webClient) return;

  Serial.println("New user connected");

  String request = webClient.readStringUntil('\r');
  webClient.flush();

  int distance = getDistance();
```

```
  // MQTT distance data
  char msg[50];
  snprintf(msg, sizeof(msg), "Distance: %d cm", distance);
  client.publish(MQTT_PUBLIC_TOPIC, msg);

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Handle requests
  if (request.indexOf("GET / ") >= 0) {
    // main page
    webClient.println("HTTP/1.1 200 OK");
    webClient.println("Content-Type: text/html");
    webClient.println();

    // HTML and CSS styling
    webClient.println("<html lang=\"en\">");
    webClient.println("<head>");
    webClient.println("  <meta charset=\"UTF-8\">");
    webClient.println("  <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">");
    webClient.println("  <title>My Car Parking Sensor</title>");
    webClient.println("  <style>");
    webClient.println("    body { font-family: Arial, sans-serif; text-align: center; padding: 20px; background-color: #e0f7fa; margin: 0; }");
    webClient.println("    h1 { font-size: 35px; margin-bottom: 20px; }");
    webClient.println("    p { font-size: 25px; margin-bottom: 10px; }");
    webClient.println("    button { padding: 10px 20px; background-color: #007bff; color: white; border: none; cursor: pointer; margin: 10px; border-radius: 5px; }");
    webClient.println("    button:hover { background-color: #0056b3; }");
```

```
  webClient.println("    #distanceWarning { color: white; font-size: 2.5em; background-color: red; margin-top: 20px; boarder-radius: 25px;}");
  webClient.println("  </style>");
  // Use distance data
  webClient.println("  <script>");
  webClient.println("    function fetchDistance() {");
  webClient.println("      fetch('/distance').then(response => response.text()).then(data => {");
  webClient.println("        const distance = parseInt(data, 10);");
  webClient.println("        document.getElementById('distanceValue').textContent = distance + ' cm';");
  webClient.println("        const warning = document.getElementById('distanceWarning');");
  webClient.println("        if (distance < 50) {");
  webClient.println("          warning.textContent = 'Warning: Object is getting too close!';");
  webClient.println("        } else {");
  webClient.println("          warning.textContent = '';");
  webClient.println("        }");
  webClient.println("        if (distance < 25) {");
  webClient.println("          window.location.href = '/stream';");
  webClient.println("        }");
  webClient.println("      });");
  webClient.println("    }");
  webClient.println("    setInterval(fetchDistance, 1000);");
  webClient.println("  </script>");
  webClient.println("</head>");
  webClient.println("<body>");
  webClient.println("  <h1>My Car Parking Sensor Web Server</h1>");
  webClient.println("  <p>Distance: <span id=\"distanceValue\">Loading...</span></p>");
  webClient.println("  <p id=\"distanceWarning\"></p>");
  webClient.println("  <button onclick=\"location.href='/distance'\">Get Distance</button>");
  webClient.println("  <button onclick=\"location.href='/stream'\">Start Video Streaming</button>");
  webClient.println("  <button onclick=\"location.href='/graph'\">View data Graph</button>");
```

```
  webClient.println("</body>");
  webClient.println("</html>");

 } else if (request.indexOf("/distance") != -1) {
  webClient.println("HTTP/1.1 200 OK");
  webClient.println("Content-Type: text/plain");
  webClient.println();
  webClient.print(distance);
} else if (request.indexOf("/stream") != -1) {
  webClient.println("HTTP/1.1 200 OK");
  webClient.println("Content-Type: multipart/x-mixed-replace; boundary=frame");
  webClient.println();

  while (webClient.connected()) {
    camera_fb_t* fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      break;
    }

    webClient.println("--frame");
    webClient.println("Content-Type: image/jpeg");
    webClient.println("Content-Length: " + String(fb->len));
    webClient.println();
    webClient.write(fb->buf, fb->len);
    webClient.println();
    esp_camera_fb_return(fb);

    delay(100);
  }
} else if (request.indexOf("/graph") != -1) {
```

```
    webClient.println("HTTP/1.1 200 OK");
    webClient.println("Content-Type: text/html");
    webClient.println();

    // HTML and CSS for graph
    webClient.println("<html lang=\"en\">");
webClient.println("<head>");
webClient.println("  <meta charset=\"UTF-8\">");
webClient.println("  <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">");
webClient.println("  <title>Distance Graph</title>");
webClient.println("  <script src=\"https://cdn.jsdelivr.net/npm/chart.js\"></script>");
webClient.println("  <style>");
webClient.println("    body { font-family: Arial, sans-serif; text-align: center; padding: 10px;
background-color: #f0f0f0; }");
webClient.println("    canvas { max-width: 100%; height: auto; margin: 10px auto; }");
webClient.println("    button { margin: 10px; padding: 10px 20px; background: #007bff; color: #fff;
border: none; cursor: pointer; }");
webClient.println("    button:hover { background: #0056b3; }");
webClient.println("  </style>");
webClient.println("</head>");
webClient.println("<body>");
webClient.println("  <h1>Distance Data Graph</h1>");
webClient.println("  <button onclick=\"location.href='/'\">Go Back</button>");
webClient.println("  <canvas id=\"distanceGraph\"></canvas>");
webClient.println("  <script>");
webClient.println("    const ctx = document.getElementById('distanceGraph').getContext('2d');");
webClient.println("    const graphData = {");
webClient.println("      labels: [],");
webClient.println("      datasets: [{");
webClient.println("        label: 'Distance (cm)',");
webClient.println("        data: [],");
webClient.println("        borderColor: 'blue',");
```

```
webClient.println("      borderWidth: 1,");
webClient.println("      fill: false,");
webClient.println("      tension: 0.3");
webClient.println("    }]");
webClient.println("  };");
webClient.println("  const chart = new Chart(ctx, {");
webClient.println("    type: 'line',");
webClient.println("    data: graphData,");
webClient.println("    options: {");
webClient.println("      responsive: true,");
webClient.println("      scales: {");
webClient.println("        x: {");
webClient.println("          title: { display: true, text: 'Time' },");
webClient.println("          type: 'linear',");
webClient.println("          ticks: { callback: value => new Date(value).toLocaleTimeString() }");
webClient.println("        },");
webClient.println("        y: {");
webClient.println("          title: { display: true, text: 'Distance (cm)' },");
webClient.println("          beginAtZero: true");
webClient.println("        }");
webClient.println("      }");
webClient.println("    }");
webClient.println("  });");
webClient.println("  function updateGraph() {");
webClient.println("    fetch('/distance').then(response => response.text()).then(data => {");
webClient.println("      const distance = parseInt(data, 10);");
webClient.println("      const now = Date.now();");
webClient.println("      if (graphData.labels.length > 20) {");
webClient.println("        graphData.labels.shift();");
webClient.println("        graphData.datasets[0].data.shift();");
webClient.println("      }");
```

```
webClient.println("       graphData.labels.push(now);");
webClient.println("       graphData.datasets[0].data.push(distance);");
webClient.println("       chart.update();");
webClient.println("     });");
webClient.println("    }");
webClient.println("    setInterval(updateGraph, 1000);");
webClient.println("  </script>");
webClient.println("</body>");
webClient.println("</html>");
  }

  Serial.println("Request handled.");
}
```

My reflection

How I approached this piece of work:

I approached this project with a focus on creating a practical, user-friendly parking assistance system. My goal was to design a system that leverages real-time data and video streaming to enhance situational awareness during parking. I began by selecting components based on functionality, such as the ultrasonic sensor and ESP32 microcontroller, which provided the necessary connectivity and processing power. I prioritised ease of use, simplicity, and reliability in the design, opting for a web-based interface that eliminates the need for additional app installations.

What I found fairly easy was:

The initial setup of the ultrasonic sensor and the integration of the camera module were relatively straightforward. Once I had the necessary resources and guidance, I was able to successfully calibrate the sensor, get real-time distance data, and establish the video feed from the ESP32. The web-based interface was also easy to set up using HTML, CSS, and JavaScript, as these are technologies I'm familiar with, and they provided the flexibility to build a clear and interactive interface.

What I found most difficult was:

The most challenging part of the project was ensuring reliable communication and maintaining system stability, especially when trying to implement MQTT protocol for data transmission and troubleshooting issues with the ESP32's server updates. Another difficulty arose when I tried to optimise power consumption through duty cycling, as I encountered technical issues, which ultimately led me to abandon the feature.

If I were to do the work again, I would do the following differently:

If I were to repeat the work, I would focus more on early-stage testing to identify potential issues with connectivity and power management. I would also spend more time researching and implementing reliable duty cycling and ensuring that all other system components were fully functional.

What I learned is:

I learned how to effectively integrate sensors, microcontrollers, and communication protocols to build an IoT system with real-time data processing and visual feedback. I gained experience in troubleshooting connectivity and server issues with the ESP32 and learned the importance of system stability in practical applications. Additionally, I learned about the challenges of implementing security features like HTTPS and the trade-offs involved when optimising for power consumption and reliability.

I would like specific feedback on:

I'd be interested in suggestions for improving the robustness of the system, especially when it comes to handling server updates and connectivity issues. Like handling HTTPS implementation would also be valuable.

I felt this way about the work:

Overall, I felt a sense of accomplishment upon completing the project, even though some parts were more challenging than anticipated. The process of troubleshooting and problem-solving, while sometimes frustrating, ultimately helped me learn a great deal about IoT systems, connectivity, and security. As well as building a project that I would have needed as I bought a new car and as a new driver I don’t have the most confidence parking without technology assistance.