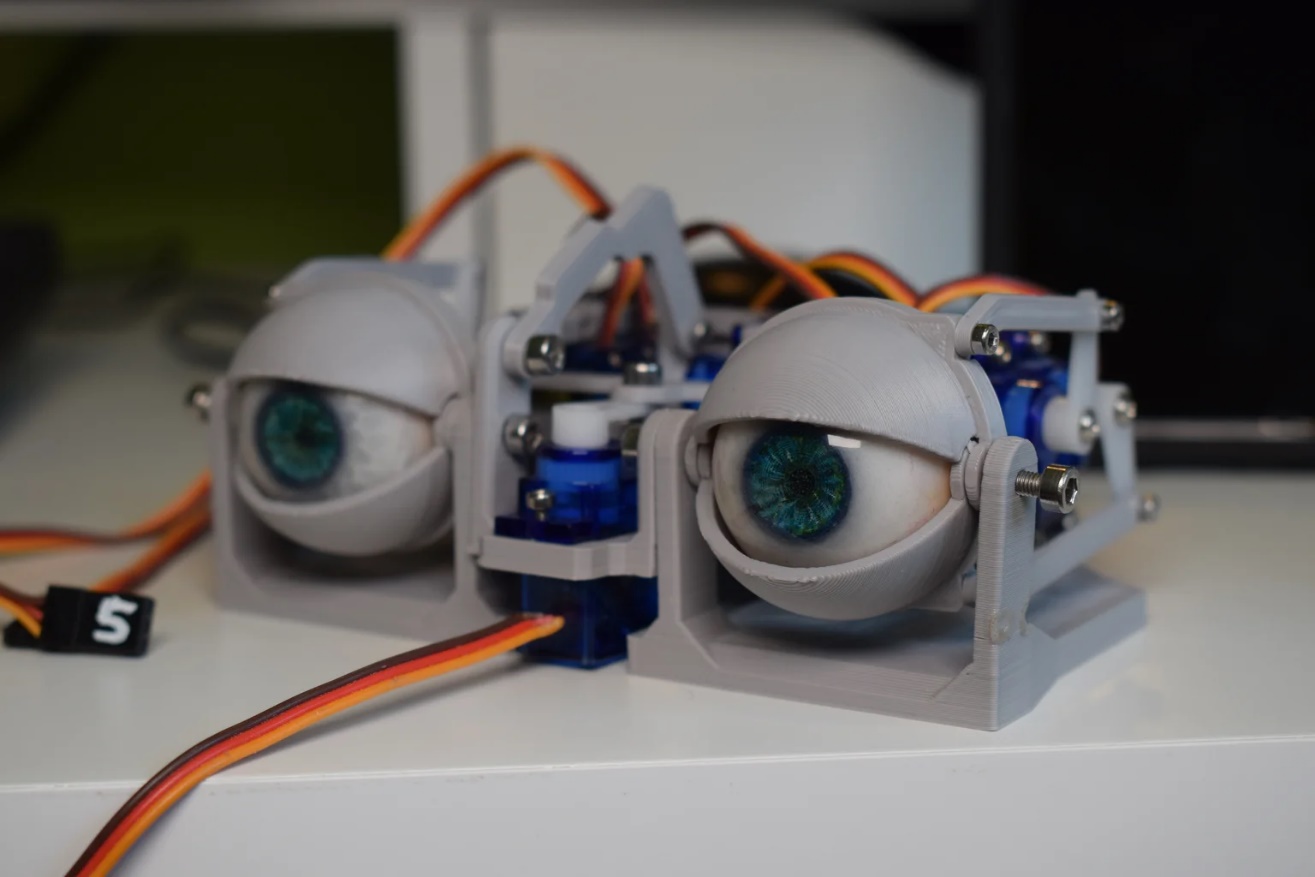
2

IOT Project: Keep An eye on it

Artevelde Hogeschool

IOT



Inhoudstafel

[IOT Project 3](#_Toc168839499)

[Arduino IDE 3](#_Toc168839500)

[Controller 3](#_Toc168839501)

[Hardware 3](#_Toc168839502)

[PowerSupply 4](#_Toc168839503)

[Components 5](#_Toc168839504)

[Code: ESP32\_Controller.ino 5](#_Toc168839505)

[Eye Controller (Receiver) 10](#_Toc168839506)

[Hardware 10](#_Toc168839507)

[Code: ESP32\_receiver.ino 10](#_Toc168839508)

[Raspberry 15](#_Toc168839509)

[RaspAP 15](#_Toc168839510)

[Software 15](#_Toc168839511)

[Add 2nd Wlan 16](#_Toc168839512)

[3D Print 18](#_Toc168839513)

[Controller 18](#_Toc168839514)

[EyeSupport 19](#_Toc168839515)

# IOT Project

## Arduino IDE

Used Board

* ESP-WROOM-32

Download the ESP32 Board

<https://dl.espressif.com/dl/package_esp32_index.json>

Adds esp32 by Espressif Systems

## Cable Data Table

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
| VRX | VRY | | SW | POT OUT | | GND | 3V3 | | GND | 5V | |

Pot:

* 5V
* G32
* GND

Joystick

* SW: G4
* VRY: SN
* VRX: SP
* +5V: 3V3
* GND

Servo Controller:

* VCC: 3V3
* SDA: G21
* SCL: G22
* GNDµ

Servo’s:

* X: 0
* Y: 1
* LT: 2
* LB: 3
* RT: 4
* RB: 5

## Eye Controller

### Hardware

A diagram of a circuit board

Description automatically generated

PowerSupply

Diagram of a power supply system

Description automatically generated

### Components

|  |  |
| --- | --- |
| Li-Ion 18650 3400mAh | A blue battery with black and red wires  Description automatically generated |
| TP4056 | A blue circuit board with a silver metal object  Description automatically generated with medium confidence |
| B10K | A close-up of a small metal object  Description automatically generated |
| Joystick | A black round object with a black handle  Description automatically generated with medium confidence |
| ESP32-Wroom-32 | A close-up of a microchip  Description automatically generated |

### Code: ESP32\_Controller.ino

#include <WiFi.h>

#include <PubSubClient.h>

#include <wifiSecret.h>

// WiFi credentials

const char\* ssid = SECRET\_SSID; // Enter your Wi-Fi name

const char\* password = SECRET\_PASS;  // Enter Wi-Fi password

const char\* hostname = "ESP32-Controller";

// MQTT Broker

const char\* mqtt\_broker = "10.3.141.1";

const char\* topic = "arduino/test";

const int mqtt\_port = 1883;

// Pin definitions

#define POTENTIOMETER\_PIN  32

#define JOYSTICK\_SW\_PIN    22

#define JOYSTICK\_VRX\_PIN   A0

#define JOYSTICK\_VRY\_PIN   A3

#define CLICK\_INTERVAL 400 // Time interval for double click detection in milliseconds

#define DEBOUNCE\_DELAY 50  // Debounce delay in milliseconds

WiFiClient espClient;

PubSubClient client(espClient);

int prevJoystickSwitchState = 1;

int prevJoystickXValue = -1;

int prevJoystickYValue = -1;

float prevPotentiometer = -1.0;

const int numSamples = 10;    // Number of samples to take

const float filterAlpha = 0.2; // Filter alpha for smoothing (0 < alpha < 1)

unsigned long lastDebounceTime = 0;

unsigned long lastClickTime = 0;

bool lastButtonState = HIGH;

bool buttonState = HIGH;

int clickCount = 0;

int currentJoystickState = 0;

void setup\_wifi() {

  delay(10);

  Serial.println();

  Serial.print("Connecting to ");

  Serial.println(ssid);

  // Stel de hostname in

  if (!WiFi.setHostname(hostname)) {

    Serial.println("Setting hostname failed!");

  }

  WiFi.begin(ssid, password);

  unsigned long startAttemptTime = millis();

  // Wait for connection with a timeout

  while (WiFi.status() != WL\_CONNECTED && millis() - startAttemptTime < 10000) {

    delay(500);

    Serial.print(".");

  }

  if (WiFi.status() != WL\_CONNECTED) {

    Serial.println("Failed to connect to WiFi");

    // Optionally, you can restart the ESP32 here to try again

  } else {

    Serial.println("");

    Serial.println("WiFi connected");

    Serial.print("IP address: ");

    Serial.println(WiFi.localIP());

  }

}

void setup() {

  // Initialize Serial Monitor

  Serial.begin(115200);

  // Initialize pins

  pinMode(JOYSTICK\_SW\_PIN, INPUT\_PULLUP);

  pinMode(JOYSTICK\_VRX\_PIN, INPUT);

  pinMode(JOYSTICK\_VRY\_PIN, INPUT);

  // Connect to Wi-Fi

  setup\_wifi();

  // Connect to MQTT broker

  client.setServer(mqtt\_broker, mqtt\_port);

  // Attempt to connect to the MQTT broker

  while (!client.connected()) {

    Serial.print("Connecting to MQTT...");

    if (client.connect("ESP32Client")) {

      Serial.println("ESP32Client connected");

    } else {

      Serial.println("failed with state ");

      Serial.println(client.state());

      delay(2000);

    }

  }

}

void loop() {

  // Check if the Wi-Fi is still connected

  if (WiFi.status() != WL\_CONNECTED) {

    Serial.println("WiFi disconnected. Attempting to reconnect...");

    setup\_wifi();

  }

  if (!client.connected()) {

    // Reconnect to MQTT broker if connection is lost

    while (!client.connected()) {

      Serial.print("Reconnecting to MQTT...");

      if (client.connect("ESP32Client")) {

        Serial.println("connected");

      } else {

        Serial.print("failed with state ");

        Serial.print(client.state());

        delay(2000);

      }

    }

  }

  client.loop();

  // Read the potentiometer value

  int potentiometerValue = analogRead(POTENTIOMETER\_PIN);

  // Read the joystick values

  float joystickXValue = analogRead(JOYSTICK\_VRX\_PIN);

  float joystickYValue = analogRead(JOYSTICK\_VRY\_PIN);

  int reading = digitalRead(JOYSTICK\_SW\_PIN);

  // Debounce logic

  if (reading != lastButtonState) {

    lastDebounceTime = millis();

  }

  if ((millis() - lastDebounceTime) > DEBOUNCE\_DELAY) {

    if (reading != buttonState) {

      buttonState = reading;

      if (buttonState == LOW) { // Button pressed

        if ((millis() - lastClickTime) < CLICK\_INTERVAL) {

          clickCount++;

        } else {

          clickCount = 1;

        }

        lastClickTime = millis();

      }

    }

  }

  lastButtonState = reading;

  // Check for single or double click

  if ((millis() - lastClickTime) > CLICK\_INTERVAL && clickCount > 0) {

    currentJoystickState = (clickCount == 1) ? 1 : 2;

    clickCount = 0;

    // Publish the joystick state

    String joystickSwitchStateStr = "S: " + String(currentJoystickState);

    client.publish(topic, joystickSwitchStateStr.c\_str());

    Serial.println(joystickSwitchStateStr);

  }

  // Reset the joystick state to 0 if the button is not pressed

  if (buttonState == HIGH && currentJoystickState != 0) {

    currentJoystickState = 0;

    String joystickSwitchStateStr = "S: " + String(currentJoystickState);

    client.publish(topic, joystickSwitchStateStr.c\_str());

    Serial.println(joystickSwitchStateStr);

  }

  // Convert and publish joystick X value if changed

  if (joystickXValue != prevJoystickXValue) {

    String joystickXValueStr = "X: " + String(joystickXValue);

    client.publish(topic, joystickXValueStr.c\_str());

    Serial.println(joystickXValueStr);

    prevJoystickXValue = joystickXValue; // Update previous state

  }

  // Convert and publish joystick Y value if changed

  if (joystickYValue != prevJoystickYValue) {

    String joystickYValueStr = "Y: " + String(joystickYValue);

    client.publish(topic, joystickYValueStr.c\_str());

    Serial.println(joystickYValueStr);

    prevJoystickYValue = joystickYValue; // Update previous state

  }

  // Publish potentiometer value if the average has changed significantly

  if (potentiometerValue != prevPotentiometer) {

    String potentiometerValueStr = "P: " + String(potentiometerValue);

    client.publish(topic, potentiometerValueStr.c\_str());

    Serial.println(potentiometerValueStr);

    prevPotentiometer = potentiometerValue; // Update previous average

  }

  // delay(200); // Small delay to prevent bouncing issues

}

float analogMultiSample(int pin) {

  float sum = 0.0;

  for (int i = 0; i < numSamples; i++) {

    sum += analogRead(pin);

  }

  return sum / numSamples;

}

## Eye Receiver

### Hardware

|  |  |
| --- | --- |
| Servo |  |
| 16-channel 12-bits PWM driver I2C robot module |  |
| ESP32-Wroom-32 | A close-up of a microchip  Description automatically generated |

### Code: ESP32\_receiver.ino

#include <Wire.h>

#include <Adafruit\_PWMServoDriver.h>

#include <WiFi.h>

#include <PubSubClient.h>

#include <wifiSecret.h>

// WiFi credentials

const char\* ssid = SECRET\_SSID;

const char\* password = SECRET\_PASS;

const char\* hostname = "ESP32-Receiver";

// MQTT Broker

const char\* mqtt\_broker = "10.3.141.1"; // IP of the Raspberry Pi running Mosquitto

const char\* topic = "arduino/test";

const int mqtt\_port = 1883;

// Adafruit PWM Servo Driver

Adafruit\_PWMServoDriver pwm = Adafruit\_PWMServoDriver();

const int servoEyeX = 0; // Servo connected to channel 0

const int servoEyeY = 1; // Servo connected to channel 1

const int servoLeftUp = 2; // Servo connected to channel 2

const int servoLeftDown = 3; // Servo connected to channel 3

const int servoRightUp = 4; // Servo connected to channel 4

const int servoRightDown = 5; // Servo connected to channel 5

WiFiClient espClient;

PubSubClient client(espClient);

// Servo min and max pulse lengths

const int SERVOMIN\_X = 220; // Minimum pulse length count (out of 4096)

const int SERVOMAX\_X = 440; // Maximum pulse length count (out of 4096)

const int SERVOMIN\_Y = 250; // Minimum pulse length count (out of 4096)

const int SERVOMAX\_Y = 500; // Maximum pulse length count (out of 4096)

unsigned long lastReconnectAttempt = 0;

void setup() {

  Serial.begin(115200);

  // Initialize the PWM driver

  pwm.begin();

  pwm.setPWMFreq(60);  // Analog servos run at ~60 Hz updates

  // Connect to Wi-Fi

  setup\_wifi();

  // Connect to MQTT broker

  client.setServer(mqtt\_broker, mqtt\_port);

  client.setCallback(callback);

  connect\_to\_broker();

}

void setup\_wifi() {

  delay(10);

  Serial.println();

  Serial.print("Connecting to ");

  Serial.println(ssid);

  // Stel de hostname in

  if (!WiFi.setHostname(hostname)) {

    Serial.println("Setting hostname failed!");

  }

  WiFi.begin(ssid, password);

  unsigned long startAttemptTime = millis();

  // Wait for connection with a timeout

  while (WiFi.status() != WL\_CONNECTED && millis() - startAttemptTime < 10000) {

    delay(500);

    Serial.print(".");

  }

  if (WiFi.status() != WL\_CONNECTED) {

    Serial.println("Failed to connect to WiFi");

    // Optionally, you can restart the ESP32 here to try again

  } else {

    Serial.println("");

    Serial.println("WiFi connected");

    Serial.print("IP address: ");

    Serial.println(WiFi.localIP());

  }

}

// Attempt to connect to the MQTT broker

void connect\_to\_broker() {

  while (!client.connected()) {

    Serial.print("Connecting to MQTT...");

    if (client.connect("ESP32Subscriber")) {

      Serial.println("ESP32Subscriber connected");

      client.subscribe(topic);

    } else {

      Serial.print("failed with state ");

      Serial.print(client.state());

      delay(2000);

    }

  }

}

void callback(char\* topic, byte\* payload, unsigned int length) {

  // Serial.print("Message arrived [");

  // Serial.print(topic);

  // Serial.print("] ");

  String message;

  for (int i = 0; i < length; i++) {

    message += (char)payload[i];

  }

  // Serial.println(message);

  // Horizontale oogbeweging

  int ColIndexX = message.indexOf("X:");

  if (ColIndexX != -1) {

    String ValueMessageX = message.substring(ColIndexX + 2); // Skip "X:"

    int angleX = ValueMessageX.toInt(); // Scale factor

    int pulseX = map(angleX, 0, 4095, SERVOMIN\_X, SERVOMAX\_X);

    pwm.setPWM(servoEyeX, 0, pulseX);

    Serial.print("Servo angleX set to: ");

    Serial.println(angleX);

  }

  // Verticale oogbeweging

  int ColIndexY = message.indexOf("Y:");

  if (ColIndexY != -1) {

    String ValueMessageY = message.substring(ColIndexY + 2); // Skip "Y:"

    int angleY = ValueMessageY.toInt(); // Scale factor

    int pulseY = map(angleY, 0, 4095, SERVOMIN\_Y, SERVOMAX\_Y);

    pwm.setPWM(servoEyeY, 0, pulseY);

    Serial.print("Servo angleY set to: ");

    Serial.println(angleY);

  }

  // // Oogleden

  // int ColIndexP = message.indexOf("Potentiometer Value:");

  // if (ColIndexP != -1) {

  //   String ValueMessageP = message.substring(ColIndexP + 19); // Skip "Potentiometer Value:"

  //   int angleP = ValueMessageP.toInt();

  //   if (angleP >= 0 && angleP <= 180) {

  //     int pulseP = map(angleP, 0, 180, SERVOMIN\_X, SERVOMAX\_X);

  //     pwm.setPWM(servoLeftUp, 0, pulseP);

  //     pwm.setPWM(servoLeftDown, 0, 180-pulseP);

  //     pwm.setPWM(servoRightUp, 0, pulseP);

  //     pwm.setPWM(servoRightDown, 0, 180-pulseP);

  //     Serial.print("Servo angleP set to: ");

  //     Serial.println(angleP);

  //   } else {

  //     Serial.println("Invalid angle for Potentiometer Value");

  //   }

  // }

}

void loop() {

  // Check if the Wi-Fi is still connected

  if (WiFi.status() != WL\_CONNECTED) {

    Serial.println("WiFi disconnected. Attempting to reconnect...");

    setup\_wifi();

  }

  // Check if the MQTT client is still connected

  if (!client.connected()) {

    unsigned long now = millis();

    if (now - lastReconnectAttempt > 5000) {

      lastReconnectAttempt = now;

      // Attempt to reconnect

      connect\_to\_broker();

    }

  } else {

    // Client connected, reset last reconnect attempt time

    lastReconnectAttempt = 0;

  }

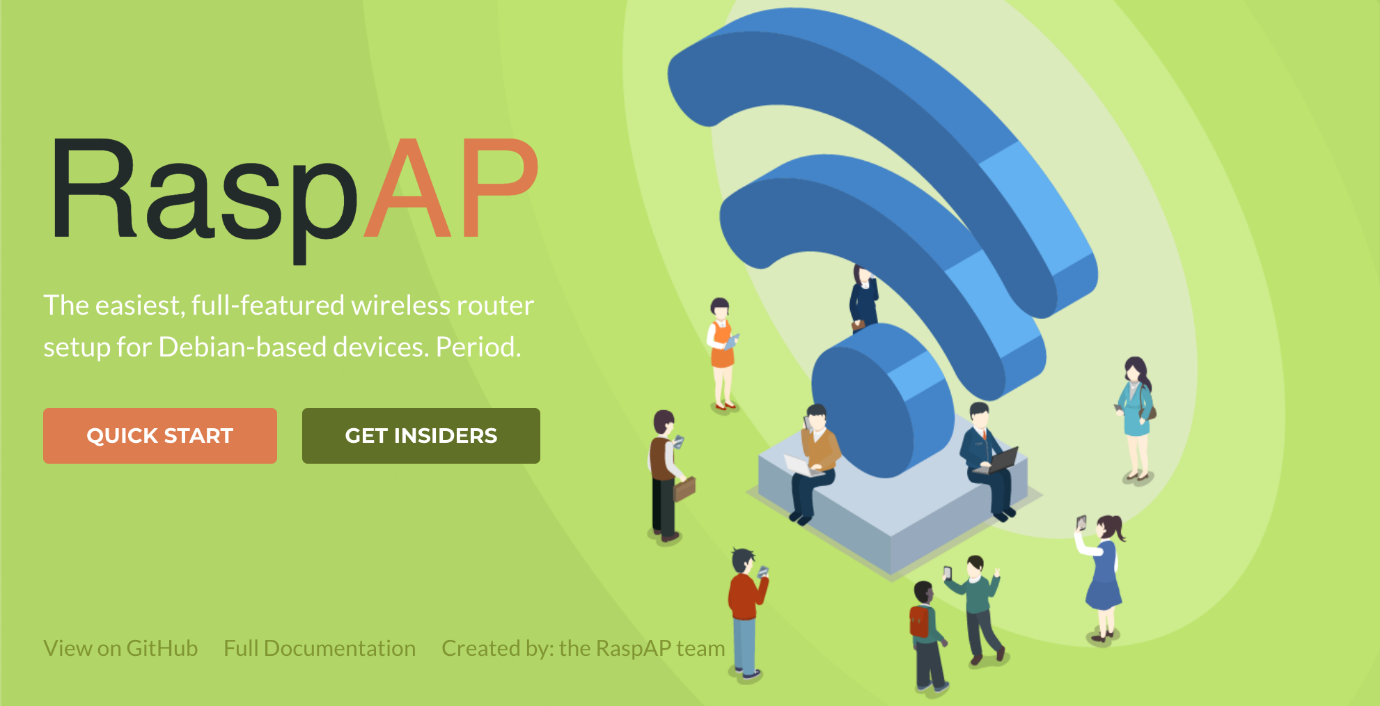
  client.loop();

}

## Raspberry



### RaspAP



### Software

MQTT Broker

### Add 2nd Wlan

Om de ASUS USB-N13 C1 draadloze adapter op een Raspberry Pi te installeren, moet je een aantal stappen volgen. Hieronder staat een gedetailleerde handleiding:

Voorbereiding

Zorg ervoor dat je Raspberry Pi is bijgewerkt met de laatste software.

Open een terminal en voer de volgende commando's uit om je systeem bij te werken:

bash

Code kopiëren

sudo apt update

sudo apt upgrade

Installatie van vereiste pakketten

Installeer de vereiste pakketten voor het bouwen van stuurprogramma's:

bash

Code kopiëren

sudo apt install dkms build-essential libelf-dev linux-headers-$(uname -r)

Download de stuurprogramma's

Ga naar de GitHub-pagina waar de stuurprogramma's voor de ASUS USB-N13 C1 beschikbaar zijn. Je kunt bijvoorbeeld de Realtek 8812au driver gebruiken:

bash

Code kopiëren

git clone https://github.com/aircrack-ng/rtl8812au.git

Build en installeer de stuurprogramma's

Navigeer naar de directory van de gedownloade stuurprogramma's en bouw deze:

bash

Code kopiëren

cd rtl8812au

sudo make dkms\_install

Controleren van de installatie

Na het installeren van de stuurprogramma's, moet je de USB-N13 C1 adapter aansluiten op je Raspberry Pi. Controleer of de adapter correct is geïnstalleerd door het volgende commando uit te voeren:

bash

Code kopiëren

lsusb

Je zou iets moeten zien zoals "ASUS USB-N13 802.11n Network Adapter".

Netwerkconfiguratie

Gebruik raspi-config om de Wi-Fi in te stellen:

bash

Code kopiëren

sudo raspi-config

Ga naar "Network Options" en vervolgens "Wi-Fi" om je Wi-Fi-netwerk te configureren.

Herstart je Raspberry Pi

Na het instellen van de Wi-Fi moet je de Raspberry Pi opnieuw opstarten:

bash

Code kopiëren

sudo reboot

Na het herstarten zou je Raspberry Pi verbinding moeten kunnen maken met Wi-Fi via de ASUS USB-N13 C1 adapter. Als je problemen tegenkomt, kun je de logbestanden controleren voor meer details:

bash

Code kopiëren

dmesg | grep 8812au

Dit zou je een gedetailleerde gids moeten geven om de ASUS USB-N13 C1 draadloze adapter op je Raspberry Pi te installeren.

## 3D Print

Hardware Ender 3V3

### Controller

Base

A grey rectangular object with holes

Description automatically generated

Top

A grey rectangular object with holes

Description automatically generated

### EyeSupport