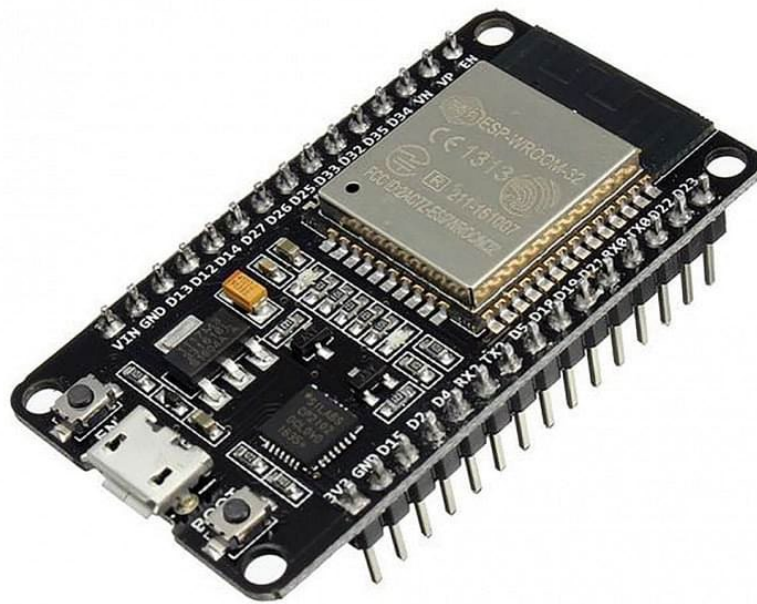


17. APRIL 2024



ESP32 PROJEKT

CNT WERKSTÄTTE

BATAEV SULIMAN, ELIAS DANZINGER
2DHIT

Inhalt

Einleitung	2
Aufgabe.....	2
Material	2
Libraries	2
DHT22	3
Aufbau DHT22	3
Funktionsweise DHT22.....	3
Servo SG90	6
Aufbau Servo SG90	6
Funktionsweise Servo SG90.....	6
Webserver	9
Aufbau Webserver	9
Funktionsweise Webserver	9
Telegram	12
Aufbau Telegram Interface	12
Funktionsweise Telegram Interface	12
ESP-Now	15
Aufbau ESP-Now.....	15
Funktionsweise ESP-Now	16
Befehle	20
Fazit	20

Einleitung

Aufgabe

Entwicklung eines Systems mit drei ESP32-Mikrocontrollern, das ESP-Now, einen Webserver, WhatsApp Interface integriert hat und durch eine Fernbedienung gesteuert werden kann, sowie Servos zur Tor-Simulation und DHT22-Temperatur- und Luftfeuchtigkeitssensoren verwendet.

Jedoch mussten wir das Projekt etwas abändern, da wir nicht wussten, ob die Fernbedienung funktioniert und somit die Steuerung mit Fernbedienung über Antenne gestrichen haben.

Ebenfalls wurde das WhatsApp Interface gestrichen da dieses nicht funktioniert hat und immer wieder Probleme gemacht hat.

https://github.com/ObamaSuli/ESP32_TOR_Projekt.git <- Der Link zum vollständigen Code

Material

ESP32 (3x), SG90 Micro Servomotor 9G (3x), DHT22 AM2302 Temperatursensor (3x), Breadboard (3x), Widerstand 4,7k Ω (3x), Jumperkabel

Libraries

Hier ist ein Bild von all den Libraries, die man für die Reproduktion dieses Projekts benötigt. Alle Libraries sind im Library Manager der Arduino IDE aufzufinden bis auf diese drei. Diese .zip Dateien muss man herunterladen und dann unter Sketch > Include Library > Add.ZIP Library hinzufügen.

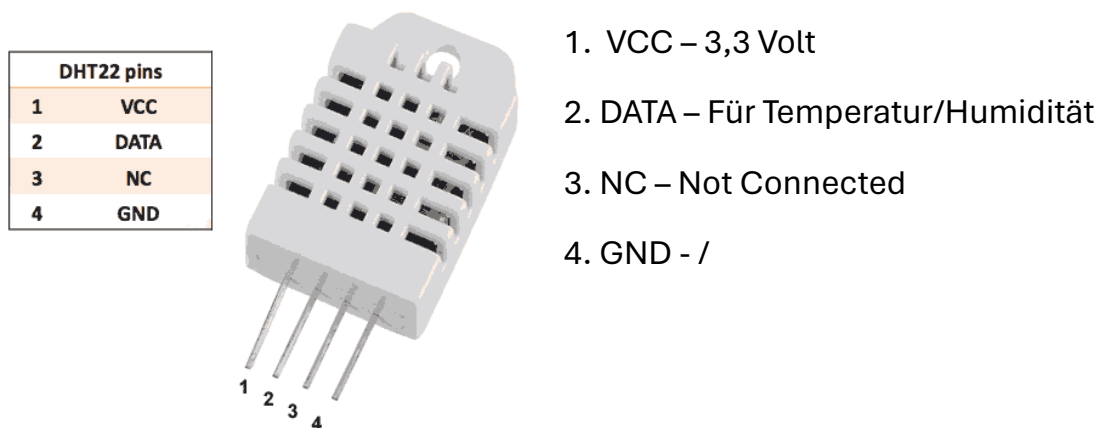
- https://github.com/adafruit/Adafruit_Sensor
- <https://github.com/adafruit/DHT-sensor-library/archive/master.zip>
- <https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot/archive/master.zip>

```
13  #include <ESP32Servo.h>
14  #include <ELECHOUSE_CC1101_SRC_DRV.h>
15  #include <Wire.h>
16  #include <WiFi.h>
17  #include <esp_now.h>
18  #include <AsyncTCP.h>
19  #include <ESPAsyncWebServer.h>
20  #include <Adafruit_Sensor.h>
21  #include <DHT.h>
22  #include <WiFiClientSecure.h>
23  #include <UniversalTelegramBot.h>
24  #include <ArduinoJson.h>
```

DHT22

Aufbau DHT22

1. Als erstes muss man diese beiden Libraries herunterladen damit der DHT22 funktioniert.
 - https://github.com/adafruit/Adafruit_Sensor
 - <https://github.com/adafruit/DHT-sensor-library/archive/master.zip>
2. Danach muss man sich entscheiden welchen GPIO-Pin auf dem ESP32 man für den DATA-Pin des DHT22 verwenden wird. In diesem Beispiel verwenden wir den GPIO-Pin 32



Funktionsweise DHT22

1. Diese Zeile initialisiert eine Instanz der DHT-Klasse zur Kommunikation mit einem DHT22-Sensor und gibt den GPIO-Pin sowie den Sensortyp an

```
60  
61 DHT dht(DHTPIN, DHTTYPE);  
62
```

2. Diese beiden Funktionen lesen die Temperatur bzw. die Luftfeuchtigkeit vom DHT22-Sensor und geben sie als Zeichenfolge zurück, wobei "--" zurückgegeben wird, wenn das Lesen fehlschlägt.

```

273 String readDHTTemperature() {
274     // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
275     // Read temperature as Celsius (the default)
276     float t = dht.readTemperature();
277     // Read temperature as Fahrenheit (isFahrenheit = true)
278     //float t = dht.readTemperature(true);
279     // Check if any reads failed and exit early (to try again).
280     if (isnan(t)) {
281         Serial.println("Failed to read from DHT sensor!");
282         return "--";
283     }
284     else {
285         Serial.println(t);
286         return String(t);
287     }

```

```

291 String readDHTHumidity() {
292     // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
293     float h = dht.readHumidity();
294     if (isnan(h)) {
295         Serial.println("Failed to read from DHT sensor!");
296         return "--";
297     }
298     else {
299         Serial.println(h);
300         return String(h);
301     }
302 }

```

3. In der später mit ESP - Now Versendeten Nachricht Outcome werden „temp“, „hum“ und „gateState“ auf aktuelle Werte gesetzt

```

447 void sendAllData(byte dest){
448
449     outcome.temp = readDHTTemperature();
450     outcome.hum = readDHTHumidity();
451     outcome.gateState = gateStatus;
452
453     espNow(dest, 5);
454 }

```

4. Diese Funktion aktualisiert die Daten für drei verschiedene Sensoren. Sie verwendet readDHTHumidity() und readDHTTemperature() für die

Feuchtigkeits- und Temperaturwerte vom DHT22-Sensor und speichert sie entsprechend in den Variablen humidity1, humidity2, humidity3, temperature1, temperature2 und temperature3. Wenn das Lesen fehlschlägt, wird "--" zurückgegeben. Die Teile des Codes, die sich auf den Torstatus beziehen, sollten ignoriert werden, da sie für den Servo relevant sind.

```
475 void refreshAllData(){
476     Serial.println("refreshing");
477     if(1 != THIS){
478         espNow(1, 0);
479         delay(2000);
480         humidity1 = income.hum;
481         temperature1 = income.temp;
482         if(income.gateState){
483             gateStatus1 = "offen";
484         }else{
485             gateStatus1 = "zu";
486         }
487     }else{
488         humidity1 = readDHTHumidity();
489         temperature1 = readDHTTemperature();
490         if(gateStatus){
491             gateStatus1 = "offen";
492         }else{
493             gateStatus1 = "zu";
494         }
495     }
496 }
497 income.temp = "--";
498 income.hum = "--";

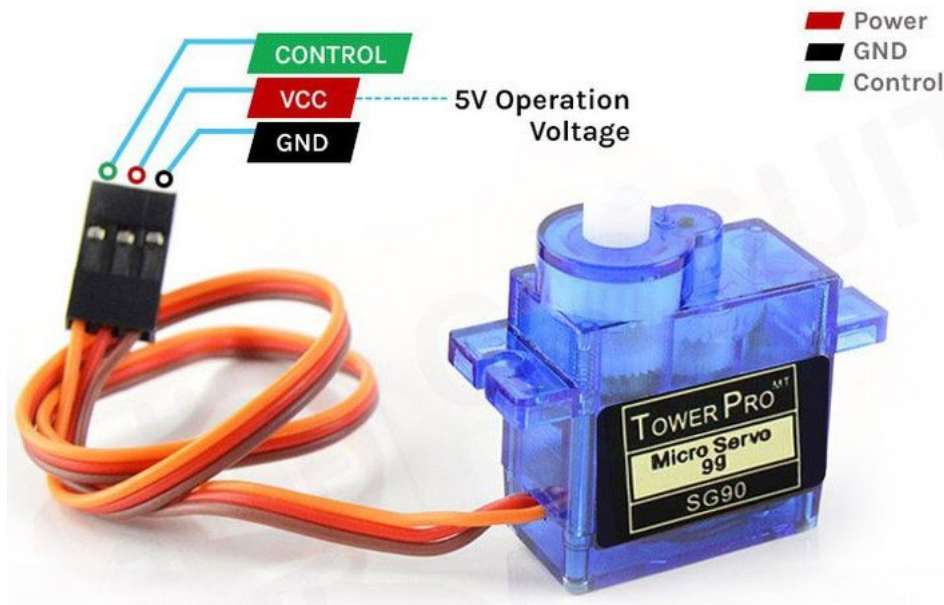
500 if(2 != THIS){
501     espNow(2, 0);
502     delay(2000);
503     humidity2 = income.hum;
504     temperature2 = income.temp;
505     if(income.gateState){
506         gateStatus2 = "offen";
507     }else{
508         gateStatus2 = "zu";
509     }
510 }else{
511     humidity2 = readDHTHumidity();
512     temperature2 = readDHTTemperature();
513     if(gateStatus){
514         gateStatus2 = "offen";
515     }else{
516         gateStatus2 = "zu";
517     }
518 }
519 income.temp = "--";
520 income.hum = "--";

522 if(3 != THIS){
523     espNow(3, 0);
524     delay(2000);
525     humidity3 = income.hum;
526     temperature3 = income.temp;
527     if(income.gateState){
528         gateStatus3 = "offen";
529     }else{
530         gateStatus3 = "zu";
531     }
532 }else{
533     humidity3 = readDHTHumidity();
534     temperature3 = readDHTTemperature();
535     if(gateStatus){
536         gateStatus3 = "offen";
537     }else{
538         gateStatus3 = "zu";
539     }
540 }
541 income.temp = "--";
542 income.hum = "--";
543 Serial.println(generateLongText());
544 }
```


Servo SG90

Aufbau Servo SG90

1. Als erstes muss man die ServoESP32 Library herunterladen (wähle Version 1.0.3 – die neuere Version haben derzeit Probleme)
2. Danach muss man sich entscheiden welchen GPIO-Pin auf dem ESP32 man für den PWM-Pin des Servo SG90 verwenden wird. In diesem Beispiel verwenden wir den GPIO-Pin 13. Es wird jedoch nicht empfohlen, die GPIOs 9, 10 und 11 zu verwenden, da sie mit dem integrierten SPI-Flash verbunden sind und nicht für andere Zwecke empfohlen werden.
3. Auf der Abbildung kann man sehen, dass der Servo mit einem Volt Wert von 5 betrieben wird, aber dieser kann auch mit 3,3 Volt betrieben werden.



Funktionsweise Servo SG90

1. Diese Ziele definiert die Konstante TOR_OUT mit dem Wert 13. Diese Konstante wird verwendet, um den Output-Pin anzugeben, an dem das Tor

angeschlossen ist. Dieser Pin wird zur Steuerung des Tors verwendet (Servo).

```
47  
48 #define TOR_OUT 13  
49
```

2. Diese Abschnitte des Codes definieren Funktionen zur Steuerung des Tors:

- Die Funktion `openGate()` öffnet das Tor, indem sie den Servo-Motor auf eine Position von 90 Grad dreht und dann eine kurze Verzögerung von 500 Millisekunden einhält. Anschließend wird der Zustand des Tors auf "geöffnet" gesetzt.
- Die Funktion `closeGate()` schließt das Tor, indem sie den Servo-Motor auf eine Position von 0 Grad dreht und dann eine kurze Verzögerung von 500 Millisekunden einhält. Danach wird der Zustand des Tors auf "geschlossen" gesetzt.
- Die Funktion `gate()` überprüft den aktuellen Zustand des Tors. Wenn das Tor geöffnet ist (`gateStatus` ist `true`), wird die Funktion `closeGate()` aufgerufen, um das Tor zu schließen. Andernfalls wird die Funktion `openGate()` aufgerufen, um das Tor zu öffnen.

```
335 // Ändert den Zustand des Tores  
336 void openGate() {  
337     servo1.write(90);           // Oeffnet das TOR  
338     delay(500);  
339     gateStatus = true;         // Speichert den Zustand des Tores  
340 }  
341 void closeGate() {  
342     servo1.write(0);           // Schließt das TOR  
343     delay(500);  
344     gateStatus = false;        // Speichert den Zustand des Tores  
345 }  
346 void gate(){  
347     if(gateStatus){  
348         closeGate();  
349     }else{  
350         openGate();  
351     }  
352 }
```


3. Dieser Abschnitt definiert eine Funktion namens `OnDataRecv`, die aufgerufen wird, wenn Daten empfangen werden.
 - `memcpy(&income, incomeData, sizeof(income));` kopiert die empfangenen Daten in die Variable `income`.
 - `Serial.print("Bytes received: ");` gibt die Anzahl der empfangenen Bytes über die serielle Schnittstelle aus.
 - Der Code führt je nach dem empfangenen `action_income` eine entsprechende Aktion aus:
 - Wenn `action_income` 0 ist, werden alle Daten an die Quelle zurückgesendet.
 - Wenn `action_income` 1 ist, wird das Tor geöffnet.
 - Wenn `action_income` 2 ist, wird das Tor geschlossen.
 - Wenn `action_income` 3 ist, wird eine Funktion aufgerufen, um das Tor zu öffnen oder zu schließen, je nach aktuellem Zustand.
 - Wenn `action_income` 5 ist, werden die Daten ignoriert.

```

676 void operateSerial(){
677     /*
678     String operationHereSerial = Serial.readStringUntil(' ');
679     byte boardHereSerial = 0;
680     if(Serial.available() > 0){
681         boardHereSerial = Serial.parseInt();
682     }
683     if(Serial.available() > 0){
684         Serial.read();
685     }
686     */
687     String inputHereSerial = Serial.readStringUntil('\n'); // Eingabe lesen, bis ein Zeilenumbruch (\n) erreicht ist
688
689     interpretText(inputHereSerial);
690
691     income.temp = "--";
692     income.hum = "--";
693     income.gateState = false;
694
695     if(Serial.available() > 0){
696         Serial.read();
697     }

```

4. Diese Funktion `operateSerial()` liest die Eingabe von der seriellen Schnittstelle, interpretiert sie und setzt dann die Werte für Temperatur, Luftfeuchtigkeit und den **Torzustand** auf Standardwerte zurück.

```

388 void OnDataRecv(const uint8_t * mac, const uint8_t *incomeData, int len){
389     memcpy(&income, incomeData, sizeof(income));
390     Serial.print("Bytes received: ");
391     Serial.println(len);
392
393     // Code der je nach income das Tor öffnet / schließt
394     dest_income = income.dest;
395     action_income = income.action;
396     src_income = income.src;
397     if(dest_income == THIS){
398         // wenn dieses Board gemeint war: gehe in send, wodurch gleich die action gehandelt wird (board_income ist sicher THIS)
399         if(action_income == 0){
400             sendAllData(src_income);
401         }else if(action_income == 1){
402             openGate();
403         }else if(action_income == 2){
404             closeGate();
405         }else if(action_income == 3){
406             gate();

```

Webserver

Aufbau Webserver

1. Als erstes muss man die AsyncTCP, ESPAsyncWebserver, ArduinoJson, Wifi, WifiClientSecure Library herunterladen

Funktionsweise Webserver

1. Erstellt einen AsyncWebServer auf dem Port 80

```
64
65 AsyncWebServer server(80);           // Create AsyncWebServer object on port 80
66
```

2. Als nächstes muss man seinen Webserver auch mit HTML/CSS/Javascript schreiben (von dem Code ist in dieser Dokumentation kein Bild da dieses zu groß ist) und kann darin Placeholders erstellen und kann diese durch diese Zeilen ersetzen

```
244 // Replaces placeholder with button section in your web page
245 String processor(const String& var) {
246     //Serial.println(var);
247     if (var == "GATEONE") {
248         return gateStatus1;
249     } else if (var == "TEMPERATUREONE") {
250         return temperature1;
251     } else if (var == "HUMIDITYONE") {
252         return humidity1;
253     } else if (var == "GATETWO") {
254         return gateStatus2;
255     } else if (var == "TEMPERATURETWO") {
256         return temperature2;
257     } else if (var == "HUMIDITYTWO") {
258         return humidity2;
259     } else if (var == "GATETHREE") {
260         return gateStatus3;
261     } else if (var == "TEMPERATURETHREE") {
262         return temperature3;
263     } else if (var == "HUMIDITYTHREE") {
264         return humidity3;
265     }
266     return String();
267 }
```

3. Dieser Code ist ein Beispiel für einen ESP32-Webserver, der verschiedene Funktionen bereitstellt, darunter die Möglichkeit, mit verschiedenen Endpunkten zu interagieren und Parameter zu übergeben.

- `handleWebServer()`: Diese Funktion wird aufgerufen, wenn der Endpunkt `/doSomething` aufgerufen wird. Sie erwartet zwei Parameter (`param1` und `param2`). Wenn `action` den Wert 5 hat, wird `refreshAllData()` aufgerufen, dann wird der Webserver beendet (`server.end()`), die Verbindung zum WiFi wird neu hergestellt und der Webserver wird erneut gestartet. Andernfalls wird die Funktion `espNow()` aufgerufen.
- `connectToWifi()`: Diese Funktion versucht, eine Verbindung zum WiFi-Netzwerk herzustellen und den Webserver zu starten. Zuerst wird eine Verbindung zum WiFi hergestellt. Dann wird eine Schleife ausgeführt, um auf eine erfolgreiche Verbindung zu warten oder bis zu einer festgelegten Zeit (hier 3000 Millisekunden) zu warten. Wenn die Verbindung erfolgreich ist, wird der ESP32-Webserver gestartet und Routen für verschiedene Endpunkte festgelegt, einschließlich einer Root-Seite (`/`), einer Funktion zum Aktualisieren eines GPIO-Pins (`/update`), und einer Funktion zum Abfragen des Zustands des GPIO-Pins (`/state`). Es gibt auch auskommentierten Code für Endpunkte zum Lesen der Temperatur und Luftfeuchtigkeit von Sensoren.

```
567 // WEB - Server
568 void handleWebServer(AsyncWebServerRequest *request){
569     int board = request->arg("param1").toInt(); // Erster Parameterwert aus der URL erfassen
570     int action = request->arg("param2").toInt(); // Zweiter Parameterwert aus der URL erfassen
571     if(action == 5){
572         refreshAllData();
573
574         server.end();
575         //WiFi.end();
576
577         connectToWifi();
578
579         /*
580         server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
581             // Die HTML-Seite mit aktualisierten Werten zurücksenden
582             request->send(200, "text/html", replacePlaceholders(index_html));
583         });
584         */
585     }else{
586         espNow(board, action);
587     }
588 }
```

```

618 void connectToWifi(){
619
620     // Connect to Wi-Fi
621     WiFi.begin(ssid, password);
622     startTime = millis(); // Startzeit erfassen
623     client.setCACert(TELEGRAM_CERTIFICATE_ROOT);
624     while (WiFi.status() != WL_CONNECTED && millis() - startTime < 3000) {
625         delay(1000);
626         Serial.println("Connecting to WiFi..");
627     }
628     if(WiFi.status() != WL_CONNECTED){
629         Serial.println("Could not connect to Wifi");
630     }
631     else{
632         // Print ESP32 Local IP Address
633         Serial.println(WiFi.localIP());
634
635         // Route for root / web page
636         server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
637             request->send_P(200, "text/html", index_html, processor);
638         });
639
640         server.on("/doSomething", HTTP_GET, [](AsyncWebServerRequest *request){ handleWebServer(request); });
641         // Send a GET request to <ESP_IP>/update?state=<inputMessage>
642         server.on("/update", HTTP_GET, [](AsyncWebServerRequest* request) {
643             String inputMessage;
644             String inputParam;
645             // GET input1 value on <ESP_IP>/update?state=<inputMessage>
646             if (request->hasParam("state")) {
647                 inputMessage = request->getParam("state")->value();
648                 digitalWrite(13, inputMessage.toInt());
649             } else {
650                 inputMessage = "No message sent";
651             }
652             Serial.println(inputMessage);
653             request->send(200, "text/plain", "OK");
654         });
655
656         // Send a GET request to <ESP_IP>/state
657         server.on("/state", HTTP_GET, [](AsyncWebServerRequest* request) {
658             request->send(200, "text/plain", String(digitalRead(13)));
659         });
660
661         // Send a GET request to <ESP_IP>/state
662         server.on("/state", HTTP_GET, [](AsyncWebServerRequest* request) {
663             request->send(200, "text/plain", String(digitalRead(13)));
664         });
665
666         /*
667         server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
668             request->send_P(200, "text/plain", readDHTTemperature().c_str());
669         });
670         server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
671             request->send_P(200, "text/plain", readDHTHumidity().c_str());
672         });
673         */
674
675         // Start server
676         server.begin();
677     }

```

Telegram

Aufbau Telegram Interface

1. Als erstes muss man die UniversalTelegramBot Library herunterladen. (Es wird auch die Wifi Bibliothek verwendet – siehe Webserver)
2. Danach Telegram herunterladen und den Botfather verwenden um 3 Bots für die 3 ESP32 zu erstellen – Username und Name der Bots sind irrelevant aber es wichtig sich die Tokens der Bots zu speichern da diese für die Kommunikation gebraucht wird
3. Zusätzlich den IDBot anschreiben und seine ChatID beantragen diese ist ebenfalls für die Kommunikation benötigt

Funktionsweise Telegram Interface

1. Diese Teile des Codes konfigurieren die Verbindung zu einem Telegram-Bot für die Kommunikation mit einem Chat. Hier ist eine Zusammenfassung:
 - String token[]: Dieses Array enthält die Telegram-Bot-Token für verschiedene Boards. Die Indizes im Array entsprechen den Board-Nummern. token[THIS] wird verwendet, um das Bot-Token für das aktuelle Board zu wählen.
 - String BOTtoken: Diese Zeile setzt das Bot-Token auf das entsprechende Token für das aktuelle Board.
 - String CHAT_ID: Hier wird die Chat-ID festgelegt, mit der der Bot kommunizieren soll. Diese ID identifiziert den Chat oder die Gruppe, in der der Bot Nachrichten empfangen und senden soll.
 - WiFiClientSecure client: Hier wird ein sicheres WiFi-Client-Objekt erstellt, das für die Kommunikation mit dem Telegram-Server verwendet wird.
 - UniversalTelegramBot bot(BOTtoken, client): Hier wird ein UniversalTelegramBot-Objekt erstellt, das das Bot-Token und den Client verwendet, um Nachrichten an den Telegram-Server zu senden und zu empfangen.

- lastUpdateTime und updateInterval: Diese Variablen und Konstanten werden für die Aktualisierung des Bots verwendet, um sicherzustellen, dass der Bot regelmäßig aktualisiert wird. Derzeit ist die Aktualisierungsintervall auf 0 Millisekunden eingestellt, was bedeutet, dass der Bot so oft wie möglich aktualisiert wird.

2. Dieser Abschnitt des Codes überprüft regelmäßig, ob es neue Nachrichten für den Bot gibt, indem er in bestimmten Zeitintervallen

```

80 String token[] = { " ", "7081776857:AAGZu1P5EB-_zNqeIemqdzPqNRixJOczWMI", "7184436868:AAF1fVyoowA9ns-H9hUX_xU0NsBxQBx0LCI", "6720615447:AAGt-R-hI56hcwnogYpZLOU_bw
81 // Initialize Telegram BOT
82 String BOTtoken =token[THIS]; // your Bot Token (Get from Botfather)
83 // Use @myidbot to find out the chat ID of an individual or a group
84 String CHAT_ID= "7009086441";
85 WiFiClientSecure client;
86 UniversalTelegramBot bot(BOTtoken, client);
87
88 // Timer for bot update check
89 unsigned long lastUpdateTime = 0;
90 const unsigned long updateInterval = 0; // Update interval in milliseconds (5 seconds)

```

3. Diese Funktion „handleNewMessages“ verarbeitet neue Nachrichten, die vom Telegram-Bot empfangen wurden.

- Die Funktion geht durch alle neuen Nachrichten, die vom Bot empfangen wurden, numNewMessages.
- Für jede Nachricht wird die Chat-ID der Nachricht überprüft, um sicherzustellen, dass die Nachricht von einem autorisierten Benutzer kommt -> ChatID die oben schon definiert wurde
- Wenn die Nachricht mit "/start" übereinstimmt, wird eine Willkommensnachricht mit einer Liste der verfügbaren Befehle gesendet.
- Die Funktion ruft dann interpretText(text) auf, um den Text der Nachricht zu interpretieren und entsprechende Aktionen auszuführen.
- Wenn die Nachricht den Befehl "/data" gefolgt von einer Zahl (1, 2 oder 3) enthält, wird eine Nachricht mit den Informationen (Temperatur, Luftfeuchtigkeit, Zustand des Tores) des entsprechenden ESPs zurückgesendet.
- Am Ende wird dem Benutzer eine Bestätigungsnachricht gesendet, dass die Operation ausgeführt wurde -> Temperatur- und Luftfeuchtigkeitsvariablen werden zurückgesetzt.


```

942 void handleNewMessages(int numNewMessages) {
943     for (int i=0; i<numNewMessages; i++) {
944         Serial.print("got telegram messages");
945         String chat_id = String(bot.messages[i].chat_id);
946         if (chat_id != CHAT_ID) {
947             bot.sendMessage(chat_id, "Unauthorized user", "");
948             continue;
949         }
950
951         String text = bot.messages[i].text;
952
953         if (text == "/start") {
954             String welcome = "Welcome.\n";
955             welcome += "Use the following commands:\n\n";
956             welcome += "/data - get all Infos of an ESP\n";
957             welcome += "/open - open gate of an ESP\n";
958             welcome += "/close - close gate of an ESP\n";
959             welcome += "Add a number 1 - 3 after the Text to choose ESP\n";
960             bot.sendMessage(chat_id, welcome, "");
961             continue;
962         }

```

```

964         interpretText(text);
965         if(text == "data 1"){
966             String outputString = "TOR 1";
967             outputString += "\nTemperatur: " + income.temp;
968             outputString += "\nHumidity: " + income.hum;
969             if(income.gateState){
970                 outputString += "\nZustand: offen\n";
971             }else{
972                 outputString += "\nZustand: geschlossen\n";
973             }
974
975             bot.sendMessage(chat_id, outputString, "");
976         }else if(text == "data 2"){
977             String outputString = "TOR 2";
978             outputString += "\nTemperatur: " + income.temp;
979             outputString += "\nHumidity: " + income.hum;
980             if(income.gateState){
981                 outputString += "\nZustand: offen\n";
982             }else{
983                 outputString += "\nZustand: geschlossen\n";
984             }

```

```

986     bot.sendMessage(chat_id, outputString, "");
987 }else if(text == "data 3"){
988     String outputString = "TOR 3";
989     outputString += "\nTemperatur: " + income.temp;
990     outputString += "\nHumidity: " + income.hum;
991     if(income.gateState){
992         outputString += "\nZustand: offen\n";
993     }else{
994         outputString += "\nZustand: geschlossen\n";
995     }
996
997     bot.sendMessage(chat_id, outputString, "");
998 }
999 income.temp = "--";
1000 income.hum = "--";
1001
1002 bot.sendMessage(chat_id, "Operated", "");
1003
1004 }
1005 }

```

ESP-Now

Aufbau ESP-Now

1. Als erstes muss man die esp_now Library herunterladen. (Es wird auch die Wifi Bibliothek verwendet – siehe Webserver)
2. Danach muss man die Mac-Adressen der einzelnen ESP32 herausfinden

```

68     uint8_t ed[] = {0x08, 0x3A, 0xF2, 0xB6, 0x79, 0xD0};
69     uint8_t one[] = {0x40, 0x22, 0xD8, 0x56, 0x82, 0x14};
70     uint8_t two[] = {0xB4, 0x8A, 0x0A, 0x57, 0xBE, 0x18};
71     uint8_t three[] = {0x40, 0x22, 0xD8, 0x52, 0xE0, 0xEC};

```

Funktionsweise ESP-Now

1. Um über ESP – NOW die Kommunikation zwischen mehreren ESP32 Boards zu ermöglichen ist es notwendig, die MAC – Adressen der jeweiligen Boards zu kennen. Hierzu wird folgender Code hochgeladen und durchgeführt:

```
#include "WiFi.h"

void setup(){
  Serial.begin(115200);
  WiFi.mode(WIFI_MODE_STA);
  Serial.println(WiFi.macAddress());
}

void loop(){
}
```

im Serial Plotter eine Baud-rate von 115200 einstellen und anschließend den EN-Knopf am ESP32 drücken.

Die Mac Adressen könne aufgeschrieben und den jeweiligen Boards zugeordnet werden. Diese müssen später im ESP32 Code gespeichert werden.

```
// MAC - Adressen aller Boards
uint8_t one[] = {0x40, 0x22, 0xD8, 0x56, 0x82, 0x14};
uint8_t two[] = {0xB4, 0x8A, 0x0A, 0x57, 0xBE, 0x18};
uint8_t three[] = {0x40, 0x22, 0xD8, 0x52, 0xE0, 0xEC};
```

2. ESP – NOW erfordert die Libraries: **<esp_now.h>** und **<WiFi.h>**
3. Nun wird eine Struktur für die Nachrichten vorgegeben, von der einmal ein income und einmal ein outcome Variable erstellt wird, um dort eingehende oder zu versendende Daten hineinspeichern zu können. Zusätzlich werden peer info Variablen für die unterschiedlichen ESP32 Boards erstellt.

```
// Struktur von über ESP - NOW versendeten / eingelesenen Nachrichten
typedef struct struct_message {
  byte src;
  byte dest;
  byte action;

  String temp;
  String hum;
  bool gateState;
} struct_message;

struct_message income;
struct_message outcome;

esp_now_peer_info_t peerInfoOne;
esp_now_peer_info_t peerInfoTwo;
esp_now_peer_info_t peerInfoThree;
```

4. Folgende Variablen werden später zusätzlich verwendet:

```

bool dataReceived = false;
bool gateState = false;           // false: zu, true: offen
byte dest_send;                   // Dest bei Nachricht senden
byte action_send;                 // zu versendende Aktion
byte dest_income;                 // Dest einer erhaltenen Nachricht
byte action_income;              // Erhaltene Aktion
byte src_income;                 // Source ESP einer eingegangenen Nachricht
byte count_send = 0;

String temperature1, temperature2, temperature3;
String humidity1, humidity2, humidity3;
String gateState1, gateState2, gateState3;

String success;
int boardHere;

```

5. Folgende Methode wird ausgeführt, wenn Daten versendet wurden. Sie dient zur Erkennung, ob die Daten erfolgreich gesendet werden konnten:

```

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status){
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.print(status == ESP_NOW_SEND_SUCCESS ? " :)" : " :( ");
    if (status == 0) {
        success = " :) ";
    }else {
        success = " :( ";
        Serial.println(success);
        if(count_send < 5){
            // retry
            send(dest_send, action_send);
            count_send++;
        }
    }
}

```

6. Folgende Methode wird ausgeführt, wenn Daten erhalten werden. Hier ist der Code enthalten, der auf die Nachricht reagiert. In unserem Fall wird je nach eingehender action, eine unterschiedliche Aktion durchgeführt.

```

void OnDataRecv(const uint8_t * mac, const uint8_t *incomeData, int len){
    memcpy(&income, incomeData, sizeof(income));
    Serial.print("Bytes received: ");
    Serial.println(len);

    // Code der je nach income das Tor öffnet / schließt
    dataReceived = true;

    dest_income = income.dest;
    action_income = income.action;
    src_income = income.src;
    if(dest_income == THIS){
        // wenn dieses Board gemeint war: gehe in send, wodurch gleich die action gehandelt wird (board_income ist sicher THIS)
        if(action_income == 0){
            sendAllData(src_income);
        }else if(action_income == 1){
            openGate();
        }else if(action_income == 2){
            closeGate();
        }else if(action_income == 3){
            gate();
        }
    }
}

```

```

}else if(action_income == 5){
    if(src_income == 1){
        temperature1 = income.temp;
        humidity1 = income.hum;
        if(income.gateState){
            gateState1 = "offen";
        }else{
            gateState1 = "geschlossen";
        }
    }
    }else if(src_income == 2){
        temperature2 = income.temp;
        humidity2 = income.hum;
        if(income.gateState){
            gateState2 = "offen";
        }else{
            gateState2 = "geschlossen";
        }
    }
    }else if(src_income == 3){
        temperature3 = income.temp;
        humidity3 = income.hum;
        if(income.gateState){
            gateState3 = "offen";
        }
    }
}

```

7. Folgende Methode versendet Daten, nachdem in die Nachricht outcome die mindestens notwendigen Informationen gespeichert wurden

```

// as address 1, 2, 3 to define if it should go to board 1, 2, or 3
void send(byte toBoard, byte action){

    outcome.src = THIS;
    outcome.dest = toBoard;
    outcome.action = action;

    dest_send = toBoard;
    action_send = action;

    esp_err_t result;
    switch(toBoard) {
        case 1:
            result = esp_now_send(one, (uint8_t *) &outcome, sizeof(outcome));
            break;
        case 2:
            result = esp_now_send(two, (uint8_t *) &outcome, sizeof(outcome));
            break;
        case 3:
            result = esp_now_send(three, (uint8_t *) &outcome, sizeof(outcome));
            break;
        default:
            result = ESP_LOG_ERROR;
            break;
    }
    if (result == ESP_OK) {
        Serial.print("Sent with success. Further: ");
    }else if (result == ESP_LOG_ERROR) {
        Serial.println("Wrong board");
    }else {
        Serial.println("Error sending the data" + result);
    }
}

// Send message via ESP-NOW
//esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &address, 1);
}

```

8. Folgende Methode wird vom Restlichen Programm aufgerufen, um Nachrichten zu senden (nicht direkt send). Hierdurch kann kontrolliert werden, ob sicher nicht das eigene Board gemeint ist, und dass mehrere Versuche, eine Nachricht zu versenden durchgeführt werden können, falls die Nachrichten nicht ankommen.

```

void espNow(int toBoard, int action){
  if(toBoard == THIS){
    switch(action){
      case 1:
        openGate();
        break;
      case 2:
        closeGate();
        break;
      case 3:
        gate();
    }
  }else{
    count_send = 0;
    dataReceived = false;
    send(toBoard, action);
    if(action == 0){
      requestStart = millis();
      while(!dataReceived && millis() - requestStart < 2000){
        // awaiting an answer
      }
    }
    delay(500);
  }
}

```

9. Im setup(): folgender Code initialisiert ESP – NOW und die onDataSent Methode

```

// Init ESP-NOW
if (esp_now_init() != ESP_OK) {
  Serial.println("Error initializing ESP-NOW");
  return;
}else{
  Serial.println("Initialized ESP-NOW");
}

esp_now_register_send_cb(onDataSent);

```

10. Folgender Code fügt die Gesprächspartner für ESP – NOW hinzu. Ein ESP32 der hier nicht hinzugefügt wird, kann nicht angesprochen werden. Zuletzt wird noch die onDataRecv Methode initialisiert.

```

// 1
if(THIS != 1){
  Serial.println("Adding peer one..");
  // Register peer
  memcpy(peerInfoOne.peer_addr, one, 6);
  peerInfoOne.channel = 0;
  peerInfoOne.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfoOne) != ESP_OK){
    Serial.println("Failed to add peer one");
    return;
  }
}

// 2
if(THIS != 2){
  Serial.println("Adding peer two..");
  // Register peer
  memcpy(peerInfoTwo.peer_addr, two, 6);
  peerInfoTwo.channel = 0;
  peerInfoTwo.encrypt = false;

  // Add peer

```


Befehle

Folgende Funktionen sind nun implementiert:

- Steuerung eines Servos, der ein Tor simuliert. (Bei jedem ESP32)
- Temperaturmessung über den DHT22. (Bei jedem ESP32 extra)
- Luftfeuchtheitsmessung über den DHT22. (Bei jedem ESP32 extra)
- (Wenn nun von „allen Daten“ gesprochen wird sind folgende drei Dinge gemeint: Temperatur in °C, Luftfeuchtigkeit in % und der Zustand des Tores eines ESP32 (offen / geschlossen). Dies für alle 3 ESP32)
- Kommunikation zwischen den 3 ESP32 ermöglicht über ESP-NOW
- Jeder ESP32 hat einen Webserver. Über diesen können alle Tore gesteuert und alle Daten angezeigt werden.
- Jeder ESP32 hat ein Telegramm – Interface. Hier kann man einige Befehle eingeben:
 - /start : gibt eine Startnachricht aus, die die möglichen Befehle erklärt
 - open 1 / open 2 / open 3: Öffnet das Tor des jeweiligen ESP32
 - close 1 / close 2 / close 3: Schließt das Tor des jeweiligen ESP32
 - Data 1 / Data 2 / Data 3: Gibt die Daten des jeweiligen ESP32 zurück
 - Auch die Kommunikation über Serial ist mit einigen Befehlen möglich:
 - open 1 / open 2 / open 3: Öffnet das Tor des jeweiligen ESP32
 - close 1 / close 2 / close 3: Schließt das Tor des jeweiligen ESP32
 - Data 1 / Data 2 / Data 3: Gibt die Daten des jeweiligen ESP32 zurück
 - wifi : Wenn der ESP32 nicht mit dem WLAN verbunden ist, versucht er sich nun mit dem WLAN zu verbinden
 - // Wenn der ESP32 die setup() Methode durchläuft, oder sich der Servo bewegt, leuchtet die eingebaute LED auf.

Fazit

Allgemein ist uns das Projekt schwergefallen, da wir uns wenig mit der Arduino IDE und dem ESP32 auskannten und sind immer wieder auf Probleme gestoßen. So sei es das eine falsche Library hinzugefügt wurde oder, dass ein Fehler in einer Methode aufgetaucht ist. Im Endeffekt mussten wir die Idee mit der Antenne und Fernbedienung streichen da dieses nach vielem Versuchen nicht funktioniert hat, beziehungsweise die Fernbedienung mit hoher Wahrscheinlichkeit nicht

funktioniert. Das gleiche für das WhatsApp Interface, welches Geld gekostet hätte. Jedoch konnten wir hier es durch ein Telegram Interface ersetzen. Um unser Projekt noch zu verbessern wollten wir auch ein OLED-Display hinzufügen und die DHT22 Werte anzeigen lassen, aber diese Idee musste auch nach langem Versuchen verworfen werden da es einfach nicht funktioniert hat. Wir hatten die Möglichkeit viel zu lernen und zu wachsen, aber die Schlaflosen Nächte die wir wegen Fehlerhaften Code durchleben mussten werden wir nie vergessen.