

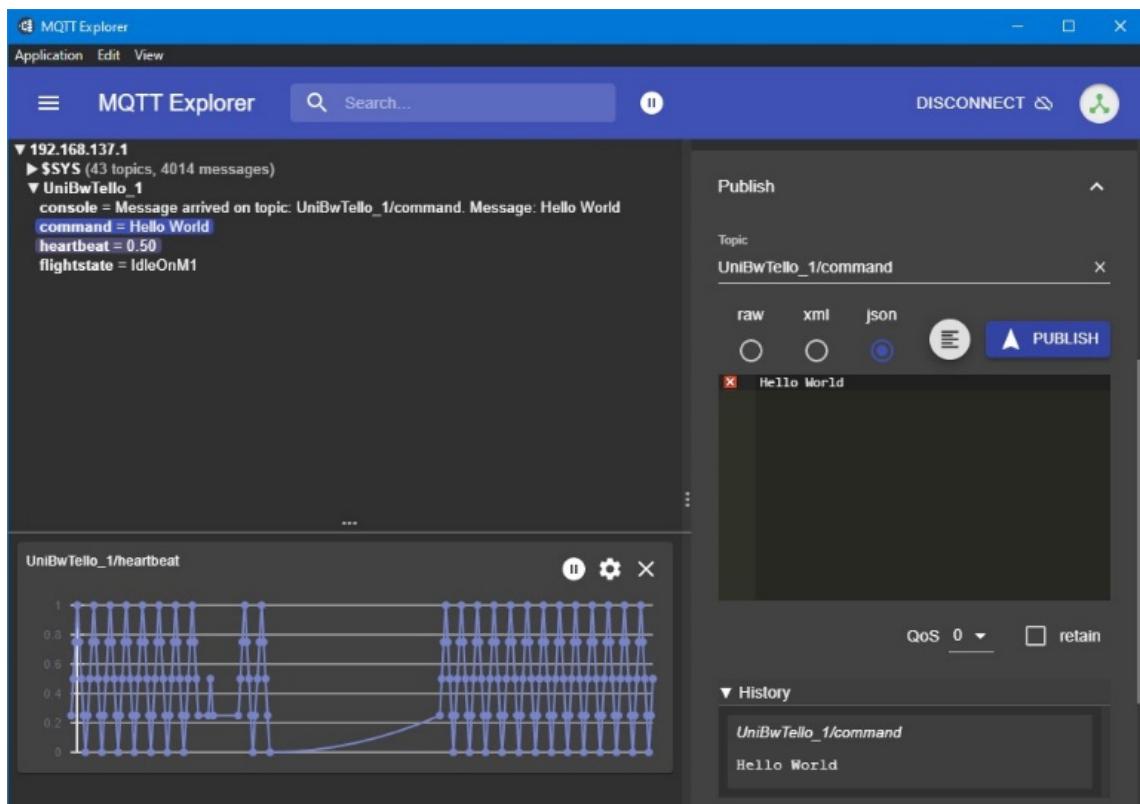
Universität der Bundeswehr München
ETTI 2: Verteilte Intelligente Systeme
Prof. Dr. rer. nat. Antje Neve
in Zusammenarbeit mit
WIWeB GF250

Frühjahrstrimester 2023

Verfasser: Karl Scholz

Internet of Things Praktikum

Teil 3
MQTT Benutzung mit PC und Mikrocontroller
26.05.2023



Voraussetzung und Arbeitsanweisung

Voraussetzung:

Dieses Dokument setzt eine Einrichtung an Ihrem Rechner voraus. Diese haben Sie in Teil 1 des Praktikums durchgeführt. Zudem werden Sie alle inhaltlichen Informationen aus Teil 2 benötigen.

Arbeitsanweisung:

Arbeiten Sie dieses Dokument chronologisch durch. Auch wenn beide Projekte theoretisch unabhängig voneinander sind, ist die Erfahrung, die Sie durch das Programmieren des Python Skriptes erlangen wichtig, damit Sie bei der komplexeren Programmierung des Mikrocontrollers keine Probleme mehr mit den Grundlagen haben.

Inhaltsverzeichnis

Inhalt

Voraussetzung und Arbeitsanweisung.....	2
Inhaltsverzeichnis	3
Abkürzungsverzeichnis	4
1 Pong Back Python Programm schreiben	5
1.1 Bibliothek einbinden	5
1.2 Callback Funktion definieren	5
1.3 Client Objekt einrichten	7
1.4 Programm ausführen und testen	9
1.5 Zusatzaufgabe: Inkrement	11
2 Programmieren des ESP32 Mikrocontrollers.....	13
2.1 Öffnen des Projekts	13
2.2 Vervollständigen der Variablen.....	14
2.3 Herstellen der WiFi-Verbindung.....	15
2.4 Schreiben der MQTT Callback Funktion	17
2.5 Verbinden mit dem MQTT Broker.....	19
2.6 Callback Funktion testen	21
3 Troubleshooting.....	24
3.1 Upload auf den uController schlägt unter Linux fehl.....	24
3.2 Upload auf den uController schlägt unter Windows fehl.....	24
Abbildungsverzeichnis.....	25

Abkürzungsverzeichnis

COM	serielle COMmunication Schnittstelle
ESP32	Espressif 32
IoT	Internet of Things
GPIO	General Purpose Input / Output
GND	Ground / Masse
I ² C	Inter-Integrated Circuit Bus
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit (Gyroskop und Beschleunigungsmesser)
IPv4	Internet Protocol Version 4
LED	Light Emitting Diode
LiDAR	Light Detection And Ranging
MP	Mission Pad
MQTT	Message Queuing Telemetry Transport
PIO	PlatformIO
PIP	Package Installer for Python
PWM	Pulse Width Modulation
QoS	Quality of Service (MQTT)
RGB	Red Green Blue
SDK	Software Development Kit
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VSCODE	Visual Studio Code
WiFi	Wireless Fidelity IEEE-802.11

1 Pong Back Python Programm schreiben

In dieser Aufgabe werden Sie ein Programm in der Programmiersprache *Python* umsetzen, welches Nachrichten, die von Ihnen manuell in dem Subtopic *Ping* veröffentlicht werden auch in der Subtopic *Pong* zu veröffentlichen.

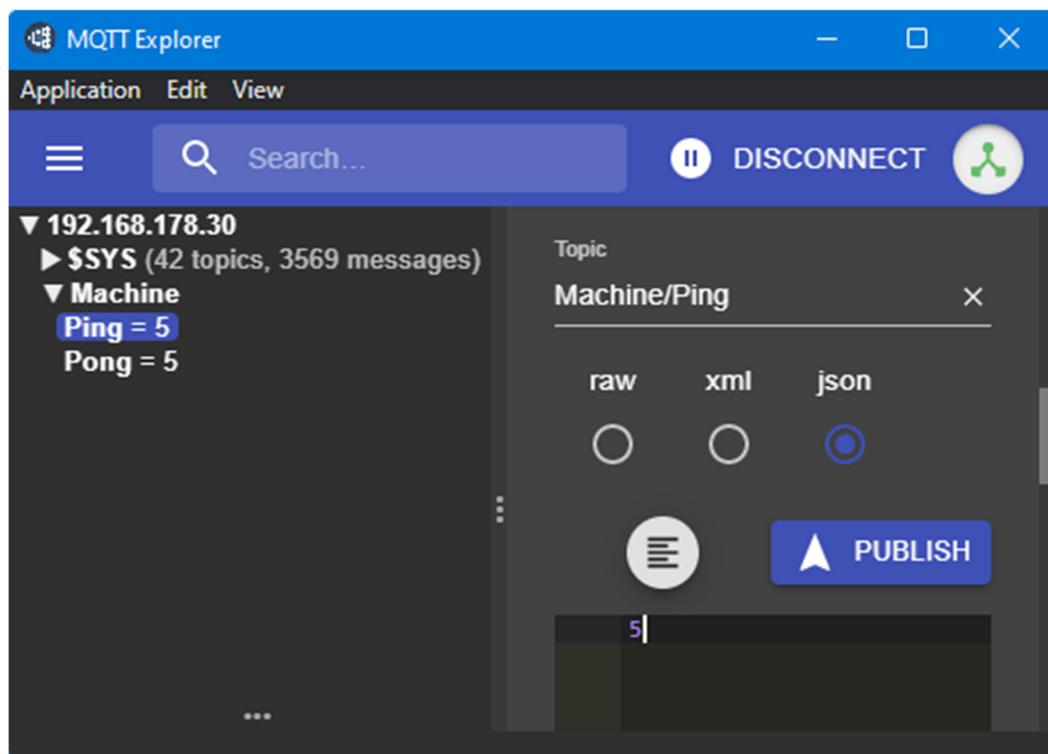


Abbildung 1: PongBack-Maschine

Erstellen Sie dazu eine leere Datei und speichern sie als *.py*-Datei ab, zum Beispiel:

PongBack.py

1.1 Bibliothek einbinden

Binden Sie die Bibliothek ein, es empfiehlt sich immer einen Alias zu nutzen, hier *mqtt*.

```
import paho.mqtt.client as mqtt
```

1.2 Callback Funktion definieren

Um Funktionen im Code zu benutzen, müssen sie bekannt und damit vor der Benutzung im Code definiert sein.

Definieren Sie deshalb eine Funktion

```
def Callback():
```

mit den Parametern

```
client, userdat, message
```

Diese wird später von der Bibliothek immer dann aufgerufen, wenn eine neue Nachricht auf ein von Ihnen abonniertes Topic ankommt.

Im Körper der Funktion müssen wir als erstes die Nachricht in Topic und Payload aufteilen und dekodieren.

Erstellen Sie eine Variable für den Inhalt der angekommenen Nachricht

```
strMessage =
```

und weisen Sie ihr den Wert von *message.payload* mit *utf-8* dekodiert und als String konvertiert zu.

```
str(message.payload.decode("utf-8"))
```

Erstellen Sie eine weitere Variable für das Topic der Nachricht

```
strTopic =
```

und weisen Sie ihr direkt den Wert von *message.topic* zu. Hier ist keine Typkonvertierung oder Dekodierung notwendig.

Im folgenden Abschnitt soll, sofern das Topic wirklich das *Ping*-Subtopic ist, dieselbe Nachricht im *Pong*-Subtopic veröffentlicht werden.

Prüfen sie dementsprechend ob

```
strTopic == "Machine/Ping"
```

wahr ist.

Falls ja, veröffentlichen Sie eine Nachricht mit der Funktion *publish()* des Objektes *client*, was wir noch erstellen werden.

```
client.publish()
```

Als Parameter sollen Sie zuerst das Topic

```
"Machine/Pong"
```

und danach den als String ausgelesenen Inhalt der empfangenen Nachricht übergeben.

Lassen Sie sich zu Debugging Zwecken mit der Funktion *print()* ausgeben, was gerade passiert.

Zur Erinnerung: *print()* benutzen Sie so:

```
var = 3
print("1", str(2), var)
```

Output:

1 2 3

1.3 Client Objekt einrichten

Erstellen Sie eine neue Variable außerhalb der Callback Funktion

```
client =
```

und initialisieren diese mit einem neuen Objekt der Klasse

```
mqtt.Client()
```

Diese benötigt von Ihnen eine *Client-ID*, d.h. einen Namen, mit dem Sie sich beim Broker später identifizieren kann. Dieser ist frei wählbar, zum Beispiel:

```
"PongBack-Machine"
```

Führen Sie anschließend die Funktion

```
connect()
```

des Objekts aus und übergeben Sie ihr als erstes einen String mit der IPv4-Adresse Ihres Brokers ohne Angabe des Ports und als zweiten Parameter, als Integer, den Port. Dies sieht zum Beispiel so aus:

```
"127.0.0.1", 1883
```

Damit das Subtopic *Ping* im *MQTT-Explorer* erscheint, soll jetzt eine erste Nachricht unter diesem Topic veröffentlicht werden. Benutzen Sie dazu die gleiche Funktion, die Sie auch in der Callback Funktion *Callback()* am Ende benutzt haben. Stellen Sie aber sicher, dass Sie die Nachricht unter dem Topic

```
"Machine/Ping"
```

veröffentlichen. Den Inhalt können Sie beliebig wählen.

Nach dem Veröffentlichen sollen Sie das Topic abonnieren („subscriben“). Führen Sie dazu die Funktion

```
subscribe()
```

des *client*-Objektes aus und übergeben Sie als Parameter einen String mit dem *Ping*-Subtopic.

Als letztes müssen Sie Ihrem *client*-Objekt noch Ihre Callback-Funktion bekannt machen.

Setzen Sie dazu die Variable

```
on_message =
```

Ihres *client*-Objektes auf den Namen Ihrer Callback Funktion ohne „()“.

Jetzt ist das Objekt eingerichtet und die Callback Funktion fertig implementiert. Geben Sie sich mit einem Aufruf der *print()* Funktion aus, dass das Programm jetzt läuft.

Danach rufen Sie die Funktion

```
loop_forever()
```

ihres *client*-Objektes auf.

Die PongBack-Maschine ist fertig implementiert und kann ausgeführt werden.

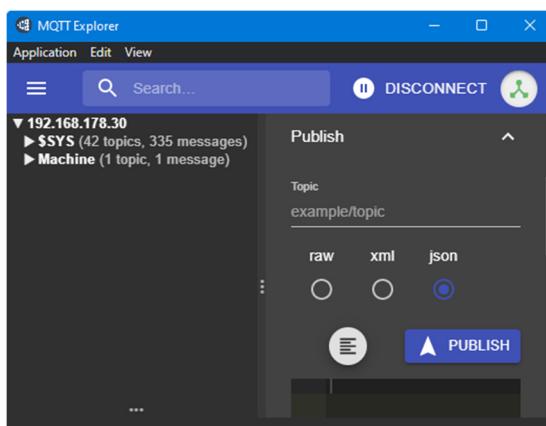
1.4 Programm ausführen und testen

Öffnen Sie zuerst das Programm *MQTT-Explorer* und verbinden sich, wie in *Teil 1 „Benutzung des MQTT-Explorers“* beschrieben, mit ihrem Broker, da sie sonst nicht mitbekommen, wenn Nachrichten veröffentlicht werden.

Führen Sie ihr Programm wie in *Teil 1 „Visual Studio Code mit Extensions für Python Entwicklung installieren“* beschrieben, aus.

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.
Lernen Sie das neue plattformübergreifende PowerShell kennen – https://aka.ms/pscore6
PS E:\Drive\Code\Python> python -u "e:\Drive\Code\Python\20220325_MQTT_Pongback\PongBack.py"
PongBack-Machine running
```

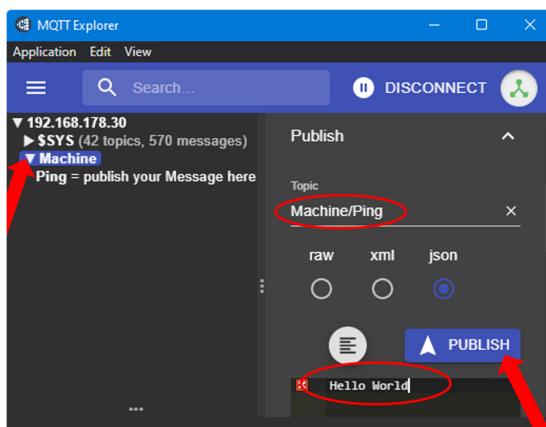
Abbildung 2: PongBack Programm läuft



Sie sollten nun ein neues Terminal bekommen, in dem Sie den Inhalt ihrer Ausgabe, die Sie vor *client.loop_forever()* geschrieben haben, sehen können.

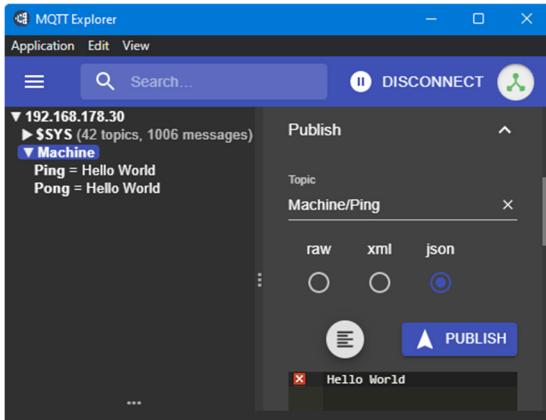
Zusätzlich sollten Sie nun im *MQTT-Explorer* ein neues Topic *Machine* sehen.

Abbildung 3: MQTT-Explorer nach Programmstart



Durch Klicken auf das Topic *Machine* klappen Sie die Subtopics auf. Hier sollte schon eine Nachricht auf das Subtopic *Ping* veröffentlicht worden sein. Veröffentlichen Sie eine beliebige Nachricht in dem Subtopic *Ping*.

Abbildung 4: Subtopics von Machine



Nach dem Veröffentlichen sollte ihre Nachricht erst bei dem Subtopic *Ping* und kurz darauf auch bei dem Subtopic *Pong* auftauchen. Zusätzlich sollten Sie in dem Terminal, in dem ihr Python Skript läuft ihre Ausgabe aus der Callback Funktion sehen.

Abbildung 5: Nachricht auch von Python veröffentlicht

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Lernen Sie das neue plattformübergreifende PowerShell kennen – https://aka.ms/pscore6

PS E:\Drive\Code\Python> python -u "e:\Drive\Code\Python\20220325_MQTT_Pongback\PongBack_BareMinimum.py"
PongBack-Machine running
Received messages: "Hello World" from Topic" PingPong/Ping "
Published messages: "Hello World" to Topic" PingPong/Pong "

```

Abbildung 6: Ausgabe aus ihrer Callback Funktion im Python Terminal

Beenden Sie das Programm, in dem Sie zur Laufzeit Strg+C in die Konsole eingeben.

Schauen Sie sich kurz bei ihren Kommilitonen um. Sind Sie überdurchschnittlich schnell an diesem Punkt angekommen?

- Ja, dann bearbeiten Sie noch die Zusatzaufgabe 1.5, damit Sie später nicht warten müssen. Hier lernen Sie noch weitere nette Zusatzfunktionen, die aber nicht 100% von Nöten sind.
- Nein, dann machen Sie bitte direkt mit *Aufgabe 2 Programmieren des ESP32 Mikrocontrollers* weiter. Wenn Sie möchten, können Sie diese Aufgabe natürlich gerne zuhause bearbeiten.

1.5 Zusatzaufgabe: Inkrement

Bis jetzt wurden nur Nachrichten verschickt und nicht mit Variablen interagiert. Erstellen Sie zuerst ganz oben eine neue Variable und initialisieren sie mit dem Wert 0.

```
iZahl = 0
```

Um aus der Callback-Funktion darauf zugreifen zu können müssen wir sie im Körper der Funktion noch einmal mit dem Schlüsselwort *global* davor definieren.

```
global iZahl
```

Jetzt können Sie die Variable auch in der Callback-Funktion wie gewohnt benutzen.

Schreiben Sie ihr Programm so um, sodass die Variable *iZahl*, mit der Zahl, die Sie unter *Ping* veröffentlichen erhöht wird.

Tipp: Sie können sich auch rechts am Rand über den „Trend“-Knopf den Verlauf der Nachrichten eines Topics graphisch anzeigen lassen. Somit ist *Ping* quasi $f'(x)$ von *Pong*.

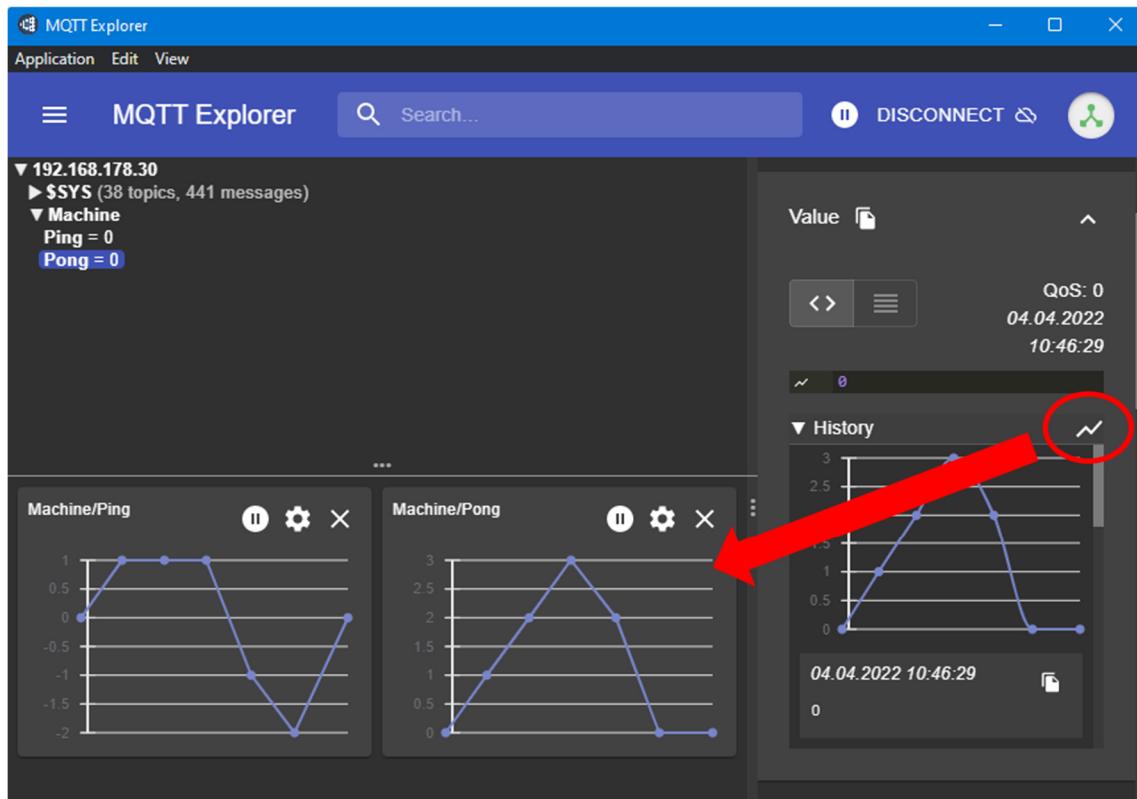


Abbildung 7: Zusatzaufgabe Inkrement: graphisch Darstellung

```
TERMINAL ...
```

PS D:\P_GDrive\Code> python -u "d:\P_GDrive\Code\Python\20220325_MQTT_Pongback\PongBack_IncrementZusatzaufgabe.py"

PongBack-Increment-Machine running

Received messages: " 0 " from Topic" Machine/Ping "

Published messages: " 0 " to Topic" Machine/Pong "

Received messages: " 1 " from Topic" Machine/Ping "

Published messages: " 1 " to Topic" Machine/Pong "

Received messages: " 1 " from Topic" Machine/Ping "

Published messages: " 2 " to Topic" Machine/Pong "

Received messages: " 1 " from Topic" Machine/Ping "

Published messages: " 3 " to Topic" Machine/Pong "

Received messages: " -1 " from Topic" Machine/Ping "

Published messages: " 2 " to Topic" Machine/Pong "

Received messages: " -2 " from Topic" Machine/Ping "

Published messages: " 0 " to Topic" Machine/Pong "

Received messages: " 0 " from Topic" Machine/Ping "

Published messages: " 0 " to Topic" Machine/Pong "

[REDACTED]

Abbildung 8: Zusatzaufgabe Inkrement: Ausgabe im Terminal

2 Programmieren des ESP32 Mikrocontrollers

2.1 Öffnen des Projekts

Öffnen Sie VSCode und gehen Sie auf die Startseite von *PIO* (Haus ganz links unten) und wählen den Punkt *Open Project*. Navigieren Sie **IN** das Starter Projekt hinein auf Ebene der *platformio.ini* – Datei. Bestätigen Sie nun das Öffnen.

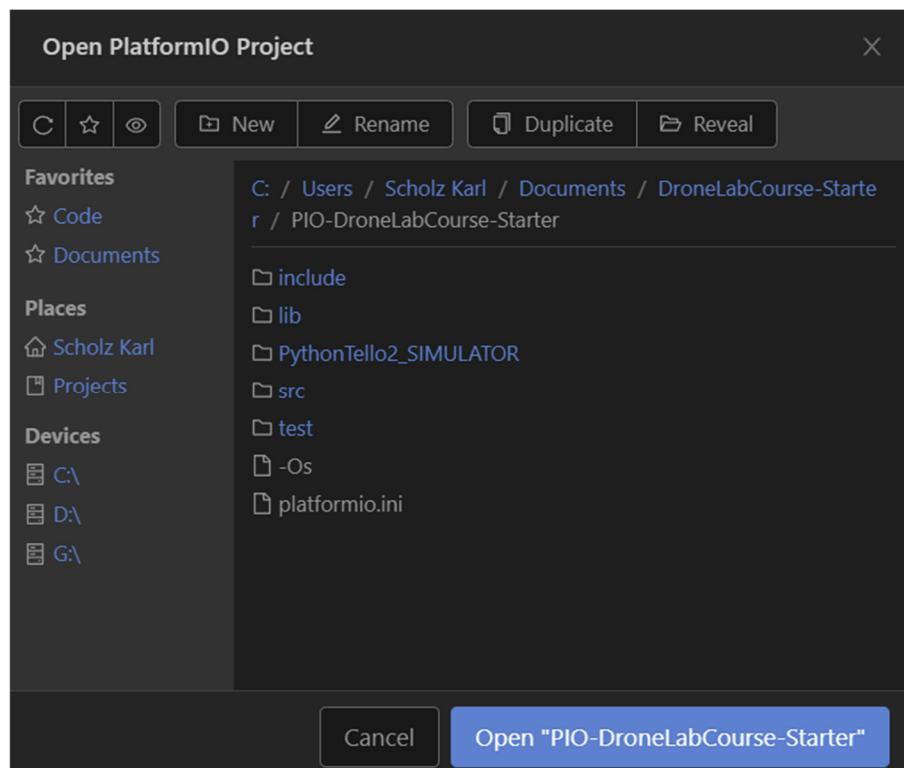


Abbildung 9: PlatformIO: Open Project

Öffnen Sie nun die Datei *main.cpp*. **Sie müssen nur in der Datei main.cpp etwas verändern, in allen anderen Dateien werden Sie nur etwas nachschauen müssen.** Als erstes drücken Sie unten links auf den Haken und Kompilieren Sie das Starter-Projekt. So können Sie kontrollieren, ob bei den vorherigen Einrichtungsschritten etwas schief-gelaufen ist, da sich das Programm im „Starter“-Zustand kompilieren lassen muss.

2.2 Vervollständigen der Variablen

Navigieren Sie in der Datei *main.cpp* zum ersten Abschnitt, in dem die globalen Variablen initialisiert werden. Verändern Sie deren Variablennamen auf keinen Fall, da auch in der *background.cpp* darauf zugegriffen wird.

```

10 WiFiClient wifi_client;
11 PubSubClient mqtt_client(wifi_client);
12
13 RMTT_Protocol tt_sdk;
14 RMTT_TOF tt_lidar;
15
16 const char *szAircraftID = "";
17 const char *szPartnerAircraftID = "";
18
19 const char *szSSID = "";
20 const char *szPassword = "";
21
22 const char *szMQTTBroker = "";
23 const uint16_t uiMQTTPort = 0;
24
25 FLIGHTSTATES_T tFlightState;
26 String strCommand = "";
27
28

```

Abbildung 10: main.cpp Startzustand der Sektion der globalen Variablen

Initialisieren Sie die C-Strings der eigenen, sowie der *AircraftID* ihrer Partnergruppe. Diese müssen sich auch von denen der anderen Gruppen unterscheiden.

Bei *SSID* und *Password* müssen sie ihre *WiFi*-Zugangsdaten eingeben.

Geben Sie die IP-Adresse, über die ihr *MQTT* Broker erreichbar ist im *Internet Protocol Version 4 (IPv4)* Format ohne Port ein, dieser kommt in die separate Variable.

Nun soll die Zustandsvariable des Automaten initial festgelegt werden, schauen sie sich hierzu die Definition des Datentyps an. Dies bestimmt in welchem Zustand ihr Programm beginnen wird. Sie müssen bei Testen einzelner Schritte nicht immer von Anfang an beginnen.

2.3 Herstellen der WiFi-Verbindung

```

28
29 void setup()
30 {
31     BackgroundSetupBEFORE();
32     // WiFi
33
34     // MQTT
35
36     BackgroundSetupAFTER();
37 }
38

```

Nachdem wir jetzt die Variablen alle festgelegt haben können wir uns um das Herstellen der Verbindung kümmern. Navigieren Sie zur Funktion `void setup()`. Schreiben Sie ihren Code unbedingt nur zwischen die beiden Funktionsaufrufe der Backgroundsetups, nicht davor oder danach!

Abbildung 11: main.cpp Startzustand der Funktion `setup()`

Mit der Funktion

`WiFi.setHostname()`

wird der Name ihres Client-Gerätes im WiFi-Netzwerk festgelegt. Mit IntelliSense können Sie nachschauen, was von Ihnen für Parameter erwartet werden. Setzen Sie ihren Hostname auf ihre *AircraftID*, die Sie im Kapitel 2.2 programmiert haben damit sie im Router Interface eindeutig identifizierbar sind.

Starten Sie den Verbindungs vorgang mit

`WiFi.begin()`

Hier müssen sie nur nacheinander den SSID Namen und das Passwort übergeben, die restlichen Parameter können Sie so lassen. Achtung: in C/C++ manchmal kann es mehrere Definitionen einer Funktion geben, sog. überlagerte/überladene Funktionen. Klicken Sie *Go To Declaration* statt *Go To Definition* an. Diese leitet sie zu den Deklarationen der Funktion in der Header Datei:

```
wl_status_t begin(const char* ssid, const char *passphrase = NULL,
int32_t channel = 0, const uint8_t* bssid = NULL, bool connect = true);
```

Hier sehen Sie, dass alle Parameter, bis auf den Netzwerknamen, mit einem Standardwert initialisiert werden, sofern Sie nichts anderes übergeben.

Die Funktion

`WiFi.status()`

gibt Ihnen einen Wert zurück, mit dem Sie feststellen können, ob Sie verbunden oder nicht verbunden sind. Klicken Sie sich mit IntelliSense durch und finden Sie heraus welchen Wert Sie gerne erreichen möchten. (Tipp: Schauen Sie sich den Datentyp des Rückgabewerts an!) Da *Wifi.begin()* im Hintergrund läuft, aber die Herstellung der Verbindung eine bestimmte Zeit dauert, müssen Sie verhindern, dass das Programm weiterläuft, bis Sie verbunden sind. Verwenden Sie eine geeignete Kontrollstruktur ihrer Wahl, die solange wartet bis Sie den gewünschten Rückgabewert erhalten.

Sie sind nun erfolgreich mit ihrem *WiFi*-Netzwerk verbunden, zur Kontrolle können Sie sich mit

WiFi.localIP()

ihre IP-Adresse, die Ihnen der Router zugewiesen hat, ausgeben lassen.

Falls sie es nicht sowieso schon getan haben, lassen Sie sich über *ConsolePrint()* an mehreren Stellen ausgeben, was momentan passiert, damit Sie es nachverfolgen können, zum Beispiel so:

```
ConsolePrint("\nTrying to connect to " + String(szSSID) + "\n");
while (<ihre Bedingung für noch nicht verbunden>){
    delay(500);
    ConsolePrint(".");
}
ConsolePrint("\nWiFi connected successfully to " + String(szSSID) +
"\nIP adress: ");
ConsolePrintln(WiFi.localIP());
```

2.4 Schreiben der MQTT Callback Funktion

```
78 void Callback(char *szTopic, byte *yMessage, unsigned int uiLength)
79 {
80
81 }
```

Abbildung 12: main.cpp Startzustand der Funktion Callback()

Bevor wir uns beim *MQTT Broker* anmelden müssen wir noch die *Callback()* Funktion schreiben. Diese wird jedes Mal ausgeführt, wenn der *Broker* uns benachrichtigt, dass es eine neue Nachricht auf ein von uns abonniertes *Topic* gibt.

Neue lokale Variable:

Da das Topic und die Nachricht nicht *by Value*, sondern *by Reference*, also als Pointer übergeben werden, kann sich der Wert von *Topic* ändern, sobald wir wieder irgendetwas mit *MQTT* benutzen. Erstellen Sie darum als erstes eine lokale Variable vom Datentyp *String* und initialisieren Sie diese direkt mit dem Wert der Variable *szTopic*. Im Arduino Framework gibt es eine Funktion *String(<Variable zum Umwandeln>)*, in die Sie als Parameter fast alle Datentypen übergeben können und dann als Rückgabewert einen *String* erhalten.

Neue lokale Variable:

Deklarieren Sie außerdem noch eine Variable für die Nachricht. Ebenfalls vom Typ *String*.

Kontrollstruktur:

In diese soeben deklarierte, noch leere, Variable soll jetzt *char* für *char* explizit typengewandelt das *byte-Array*, welches über den Pointer auf das erste Element gegeben ist, hineingeschrieben werden. Schreiben Sie dafür als erstes eine geeignete Kontrollstruktur, die eine Variable von einschließlich null erhöht. Die Anzahl der Zeichen wird in der Variable *uiLength* übergeben.

Inhalt der Kontrollstruktur:

Hängen Sie bei jeder Iteration den zum Datentyp *char* umgewandelten (expliziter Typecast) Inhalt des *byte-Arrays* an der Stelle der Laufvariable an den noch leeren Nachrichten-String an.

Ausgabe:

Lassen Sie sich über *ConsolePrint()* eine schön formatierte Nachricht ausgeben, dass eine Nachricht über dieses bestimmte Topic angekommen ist.

Weitere Kontrollstruktur:

Wenn eine Nachricht über das Subtopic */command* ankommt, soll diese in die globale Variable *strCommand* geschrieben werden. Das Topic soll dementsprechend mit

`(String(szAircraftID) + "/command")`

gleich sein. Strings können Sie hier mit `==` vergleichen. Nur dann wird die Nachricht in die globale, schon definierte, Variable *strCommand* geschrieben.

2.5 Verbinden mit dem MQTT Broker

```
void setup()
{
    BackgroundSetupBEFORE();
    // WiFi
    ConsolePrint("\nTrying to connect to " + String(szSSID) + "\n");
    <WiFi Hostnamen setzen>;
    <WiFi Verbindung starten>;
    while (<ihre Bedingung für noch nicht verbunden>)
    {
        delay(500);
        ConsolePrint(".");
    }
    ConsolePrint("\nWiFi connected successfully to " + String(szSSID) +
"\nIP adress: ");
    ConsolePrintln(WiFi.localIP());
    // MQTT

    BackgroundSetupAFTER();
}
```

Navigieren Sie wieder zurück zur *void setup()*. Die sollte jetzt ungefähr so aussehen.

Das Objekt *mqtt_client* der Klasse *PubSubClient* wurde in ~ Zeile 12 erstellt.

```
PubSubClient mqtt_client(wifi_client);
```

Dort wurde dem MQTT Objekt auch schon das WiFi Objekt übergeben. Stellen Sie sicher, dass dies der Fall ist.

Übergeben Sie nun die Variablen mit der *IPv4* Adresse des *MQTT Brokers* und dem Port an die Funktion *setServer()* des Objektes (IntelliSense!)

```
mqtt_client.setServer();
```

Danach müssen Sie die vorher eingerichtete Funktion *Callback()* dem MQTT-Objekt bekanntmachen. Nutzen Sie dafür die

```
setCallback(Callback)
```

Funktion und übergeben Sie den Namen der Callback Funktion.

Zum Verbinden wird Ihnen die Funktion

```
connect()
```

zur Verfügung gestellt. Als Tipp, falls Sie Probleme haben: Führen Sie die Funktion so lange aus, bis der gewünschte Rückgabewert eintritt. Für die Variable *id* wird nicht die IPv4 Adresse ihres Brokers, sondern die eindeutige Bezeichnung ihres Luftfahrzeuges erwartet.

Schmücken Sie ihren Algorithmus mit print-Aussagen so aus, dass Sie im Betrieb nachvollziehen können, was genau passiert (vgl. WiFi).

Nach erfolgreicher Verbindung mit dem *MQTT* Broker sollen Sie das Topic <ihreAircraftID>/command mit *Quality of Service (QoS)* 1 abonnieren. Wie Sie sich den Topic String zusammenbauen, wissen Sie schon aus der *Callback()* Funktion, das können Sie übernehmen.

Die Methode *subscribe()* erwartet allerdings einen *C-String* (Pointer auf das erste Element eines *char-Arrays*) und kein Objekt der Klasse *String*. Letztere beinhalten allerdings eine Methode Typkonvertierung in einen *C-String*. Die Benutzung funktioniert folgendermaßen:

```
char* szC_String = String("Ich war mal ein String Objekt").c_str();
```

Nachdem sie diese Zeile Code geschrieben haben, gehen sie eine Zeile darüber und veröffentlichen auf diesem Topic eine Nachricht, damit Sie in *MQTT-Debug* Programmen wie dem *MQTT-Explorer* sehen, welches Topic sie abonniert haben. In *MQTT-Explorer* können Sie dann auch darauf doppelklicken und in diesem Subtopic etwas veröffentlichen. Benutzen Sie die Funktion *publish()*, um von ihrem Luftfahrzeug aus etwas zu veröffentlichen. Benutzen Sie zum Veröffentlichen *QoS* 2. (Um die Bibliothek schlank zu halten ist *QoS* 2 leider beim Abonnieren nicht möglich.)

```

59     void loop()
60     {
61         //MQTT Loop
62         //BackgroundLoop
63         BackgroundLoop();
64         //State Machine
65         switch (tFlightState)
66         {
67             case IdleOnM1:
68                 break;
69             case M1toM2:
70                 break;
71             case M2toWall:
72                 break;
73             case AroundWalltoM3:
74                 break;
75             case M3toM4:
76                 break;
77             case M4toM1Land:
78                 break;
79             default:
80                 break;
81         }
82     }
```

Gehen Sie zur Funktion *loop()*. In dieser soll zuerst der Loop des MQTT-Objekts laufen, danach der *BackgroundLoop()* und zuletzt ihr Zustandsautomat.

Schreiben Sie den Aufruf der Funktion *loop()* des Objektes *mqtt_client*.

Abbildung 13: main.cpp Startzustand der Funktion *loop()*

2.6 Callback Funktion testen



Abbildung 14: USB-Kabel an ESP32 anschließen und An-Knopf

Schließen Sie nun den *ESP32* per *USB* an ihren Computer an.

Laden Sie ihr Programm nach erfolgreicher Kompilierung auf den Mikrocontroller (Pfeil unten links)

Legen Sie danach einen Akku ein und drücken den An-Knopf der *Tello* (unter dem *USB*-Anschluss vom *ESP32*)



Abbildung 15: Tello EDU im SDK Modus

Die *Tello* braucht ein paar Sekunden zum Booten, der *ESP32* versucht durchgehend Sie in den *Software Development Kit (SDK)* Modus, in dem sie auf Kommandos hört, zu versetzen.

Ist die *Tello* im *SDK* Modus blinkt sie lila und bestätigt es dem *ESP32*. Dieser leuchtet dauerhaft grün.

Ihr Programm läuft noch nicht ganz. Der Teil Einrichtungsteil `setup()` wurde ausgeführt, aber der `loop()` läuft noch nicht.

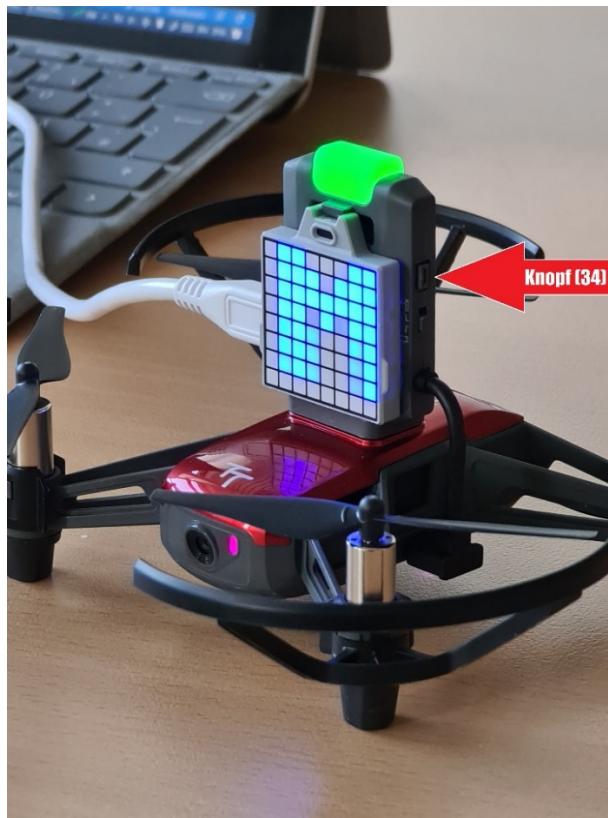


Abbildung 16: Programmstart mit Knopfdruck bestätigt

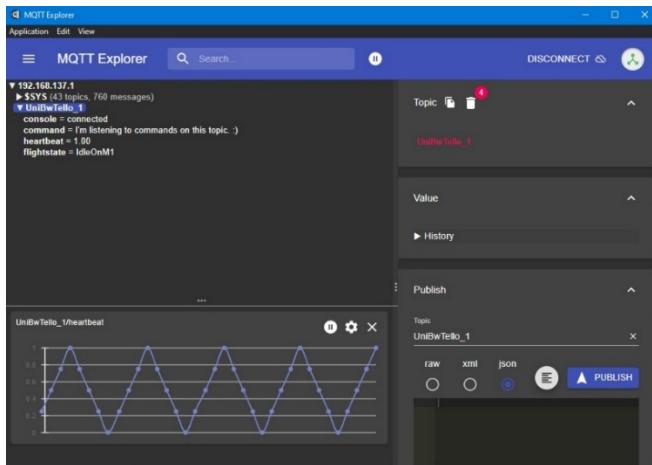


Abbildung 17: MQTT-Explorer nachdem das Programm gestartet wurde

Starten Sie nun das Programm *MQTT-Explorer* und melden sich bei ihrem Broker an.

Da die *Callback()* Funktion nur aufgerufen werden kann, wenn der *MQTT-Loop* läuft, müssen Sie dafür sorgen, dass das Programm weiterläuft.

Drücken Sie dafür den Knopf auf der linken Seite des *ESP32*.

Wenn das Programm läuft, pulsiert die obere *LED* und der *BackgroundLoop()* sorgt dafür, dass der aktuelle Zustand ihres Automaten auf der *LED-Matrix* angezeigt wird.

Im *MQTT-Explorer* sehen Sie jetzt ein großes Topic, die ihrer *AircraftID* entspricht.

Im Subtopic */console* erscheint alles, was Sie über *ConsolePrint()* ausgeben.

/heartbeat ist eine alternierende Ausgabe von Zahlen, um feststellen zu können, ob sich ihr Programm nicht in einer Endlosschleife aufgehängen hat.

/flightstate zeigt den aktuellen Wert der Zustandsvariable an.

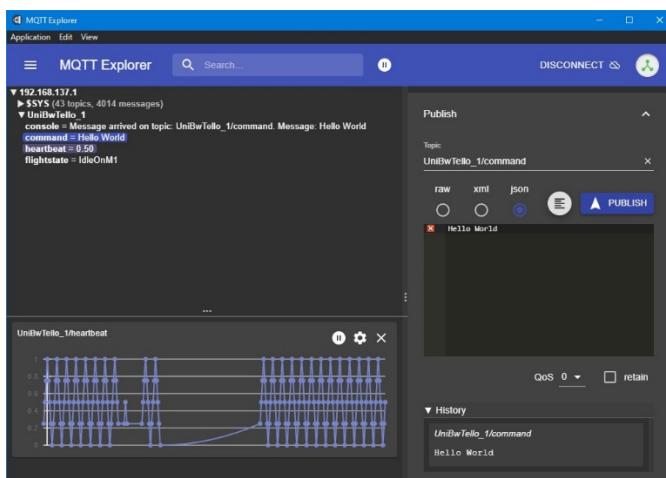


Abbildung 18: manuelle Veröffentlichung von "Hello World" in /command

Im Feld *Publish* können Sie Nachrichten veröffentlichen. Beachten Sie, dass auch eine Eingabe von Enter als \n mitgesendet wird.

Veröffentlichen Sie in das Subtopic */command* eine Nachricht und stellen Sie sicher, dass Sie in */console* ihre Ausgabe aus der *Callback()* Funktion, die Sie vorher geschrieben haben, sehen.

Herzlichen Glückwunsch, jetzt haben Sie alle Voraarbeiten geleistet um mit dem fliegenden Programmteil beginnen. Ab jetzt beginnt der spaßigste Teil!

Öffnen sie nun das vierte PDF

3 Troubleshooting

3.1 Upload auf den uController schlägt unter Linux fehl

Falls sie unter Linux ihr Programm nicht auf den ESP32 spielen können, kann es vorkommen, dass Sie nicht in der Benutzergruppe sind, die auf die tty-Ports zugreifen dürfen.

```
wiweb@ubuntu:~$ sudo adduser BENUTZERNAME dialout
```

3.2 Upload auf den uController schlägt unter Windows fehl

Unter Windows fehlt Ihnen wahrscheinlich der Treiber für die CP210X UART -> USB Schnittstelle. Laden Sie sich den Treiber hier <https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers> herunter. Entpacken Sie die ZIP-Datei. Klicken Sie mit der rechten Maustaste auf die *silabser.inf* – Datei und wählen *installieren*.

Abbildungsverzeichnis

Abbildung 1: PongBack-Maschine	5
Abbildung 2: PongBack Programm läuft.....	9
Abbildung 3: MQTT-Explorer nach Programmstart	9
Abbildung 4: Subtopics von Machine	9
Abbildung 5: Nachricht auch von Python veröffentlicht.....	10
Abbildung 6: Ausgabe aus ihrer Callback Funktion im Python Terminal	10
Abbildung 7: Zusatzaufgabe Inkrement: graphisch Darstellung	11
Abbildung 8: Zusatzaufgabe Inkrement: Ausgabe im Terminal	12
Abbildung 9: PlatformIO: Open Project.....	13
Abbildung 22: main.cpp Startzustand der Sektion der globalen Variablen....	14
Abbildung 11: main.cpp Startzustand der Funktion setup()	15
Abbildung 24: main.cpp Startzustand der Funktion Callback()	17
Abbildung 13: main.cpp Startzustand der Funktion loop()	20
Abbildung 14: USB-Kabel an ESP32 anschließen und An-Knopf.....	21
Abbildung 15: Tello EDU im SDK Modus	21
Abbildung 16: Programmstart mit Knopfdruck bestätigt	22
Abbildung 17: MQTT-Explorer nachdem das Programm gestartet wurde	22
Abbildung 18: manuelle Veröffentlichung von "Hello World" in /command.....	23

Alle nicht aufgeführten Abbildungen dieses Dokumentes sind Screenshots von Windows, Linux, Mac, Visual Studio Code oder MQTT-Explorer und wurden vom Verfasser erstellt.