

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

Frühjahrstrimester 2025

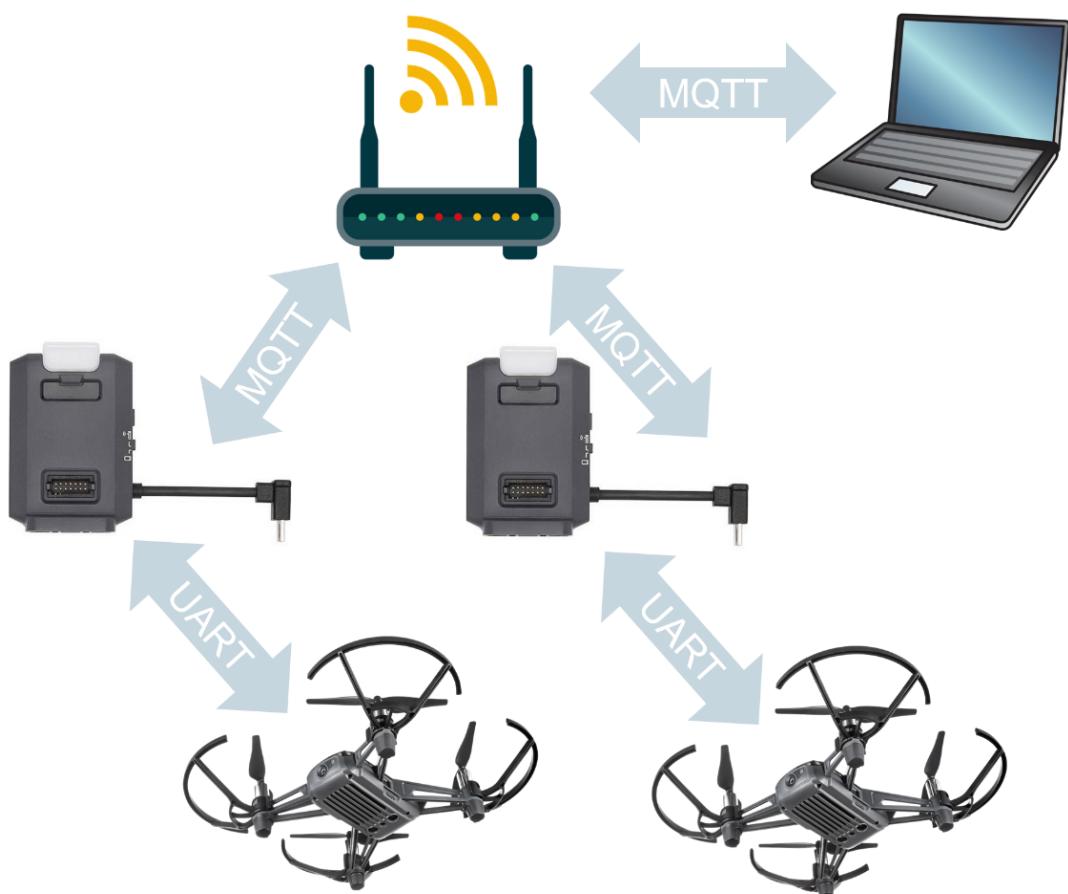
Verfasser: Karl Scholz

# Internet of Things Praktikum

## Teil 2

Informationen über das benutzte Multikoptersystem

Lesen bis 06.06.2025 08:00



## Arbeitsanweisung

Lesen Sie sich dieses Dokument bis zum 06.06.2025 um 08:00 durch. Sie müssen in diesem Dokument nicht aktiv werden, es empfiehlt sich allerdings die beschriebenen Informationen, vor allem zum Code, nebenbei an ihrem Rechner nachzuvollziehen.

Sollten sich bei der Vorbereitung Fragen ergeben, können Sie diese gerne im Vorfeld adressieren und sich per Mail melden. Auch zu Beginn des Praktikumstermins wird es die Möglichkeit geben einzelne Fragen zu stellen. Beachten Sie jedoch, dass das Praktikum eine gewissenhafte Vorbereitung voraussetzt.

# Inhaltsverzeichnis

## Inhalt

<b>Arbeitsanweisung.....</b>	<b>2</b>
<b>Inhaltsverzeichnis .....</b>	<b>3</b>
<b>Abkürzungsverzeichnis .....</b>	<b>4</b>
<b>1      Theorie MQTT (Wiederholung aus der Vorlesung) .....</b>	<b>5</b>
<b>2      Multikoptersystem.....</b>	<b>6</b>
2.1    Tello EDU .....	6
2.2    ESP32 „Rucksack“ .....	7
<b>3      Style Guide .....</b>	<b>9</b>
<b>4      IntelliSense .....</b>	<b>10</b>
<b>5      Anmerkung Debugging.....</b>	<b>12</b>
<b>6      Ordnerstruktur des Starterprojekts.....</b>	<b>13</b>
<b>7      Überblick main.cpp .....</b>	<b>14</b>
7.1    Globale Variablen.....	14
7.2    setup() .....	14
7.3    loop() .....	14
7.4    Callback() .....	14
<b>Abbildungsverzeichnis.....</b>	<b>16</b>

## Abkürzungsverzeichnis

COM	serielle COMmunication Schnittstelle
ESP32	Espressif 32
IoT	Internet of Things
GPIO	General Purpose Input / Output
GND	Ground / Masse
I <sup>2</sup> 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
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 Theorie MQTT (Wiederholung aus der Vorlesung)

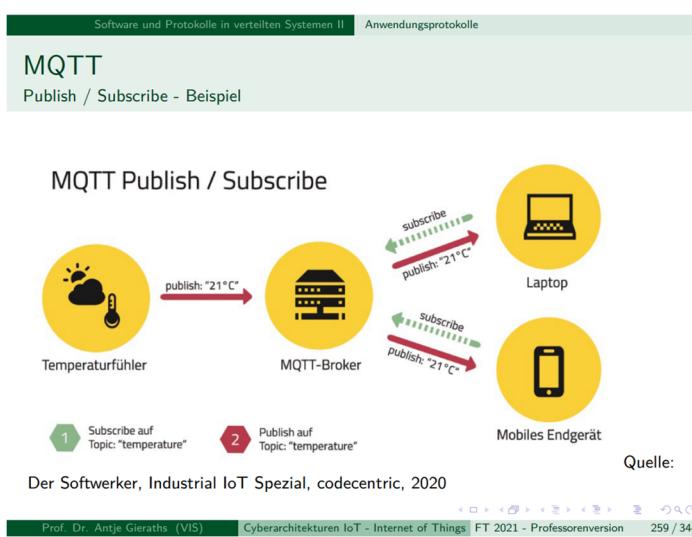


Abbildung 1: Funktionsweise MQTT, Vorlesung Neve S.259.

Das Internet Protokoll MQTT basiert auf dem Publisher / Subscriber Prinzip. Das heißt, dass Sender und Empfänger nichts übereinander wissen.

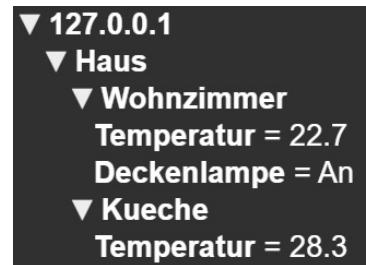


Abbildung 2: Nachrichten und Topics

Eine Nachricht besteht nur aus ihrem Inhalt und einem Topic, um den Inhalt zu kategorisieren.

Hier gibt es eine Nachricht mit dem Wert „22.7“ des Topics *Temperatur*, sowie eine andere Nachricht mit dem Wert „An“ des Topics *Deckenlampe*. Beide sind allerdings Subtopics der Topic *Wohnzimmer*, welche wiederum selbst ein Subtopic von *Haus* ist.

Subtopics haben einen gleichen Anfang bis einschließlich eines Schrägstrichs. Im Code sehen Topics bzw. Subtopics so aus:

```
strLivingroomTempTopic = "Haus/Wohnzimmer/Temperatur"
```

Sie dienen nicht nur der Übersichtlichkeit, sondern auch der Organisation:

```
strAllLivingroomTopics = "Haus/Wohnzimmer/#"
```

Mit einem „#“ kann man hier zum Beispiel alle IoT Geräte im Wohnzimmer auf einmal abonnieren. Zudem dient ein „+“ als Platzhalter in der Mitte:

```
strAllIndoorTempTopics = "Haus/+/*"
```

Hiermit lassen sich alle Temperatursensoren im Haus abfragen.

Die Zeit spielt auch eine wichtige Rolle. Nachrichten werden vom Broker immer nur an die Clients verteilt, die zum Zeitpunkt der Veröffentlichung dieses Topic abonniert haben. Sollte ein Client erst nach dem Veröffentlichen einer Nachricht das Topic einer Nachricht abonnieren, so wird er nicht vom Broker über diese Nachricht informiert.

## 2 Multikoptersystem

In diesem Praktikum wird das System *Tello Talent* von *DJI* und *Robomaster* verwendet. Dabei handelt es sich um eine *Tello EDU*, die einen „Rucksack“ trägt, in dem ein *Espressif 32 (ESP32)* Mikrocontroller der *NINA W10* Reihe von *u-blox* verbaut ist, der der *Tello EDU* über *Universal Asynchronous Receiver Transmitter (UART)* Befehle erteilt.



Abbildung 3: Tello Talent

### 2.1 Tello EDU

Die *Tello EDU* verfügt bereits in der Basisversion neben einer *IMU* und einem Barometer über einen Optical Flow Sensor, sowie einen Infrarot Entfernungsmesser, die nach unten gerichtet sind. Mit diesen Sensoren ist es der Drohne möglich die Fluglage zu bestimmen und bis zu einer bestimmten Höhe auch absolut die Position zu halten. Gerade der Optical Flow macht Befehle wie

```
"forward 50"
```

möglich, was „fliege 50 cm vorwärts“ bedeutet. In der folgenden Abbildung werden die wesentlichsten Merkmale der *Tello EDU* dargestellt.

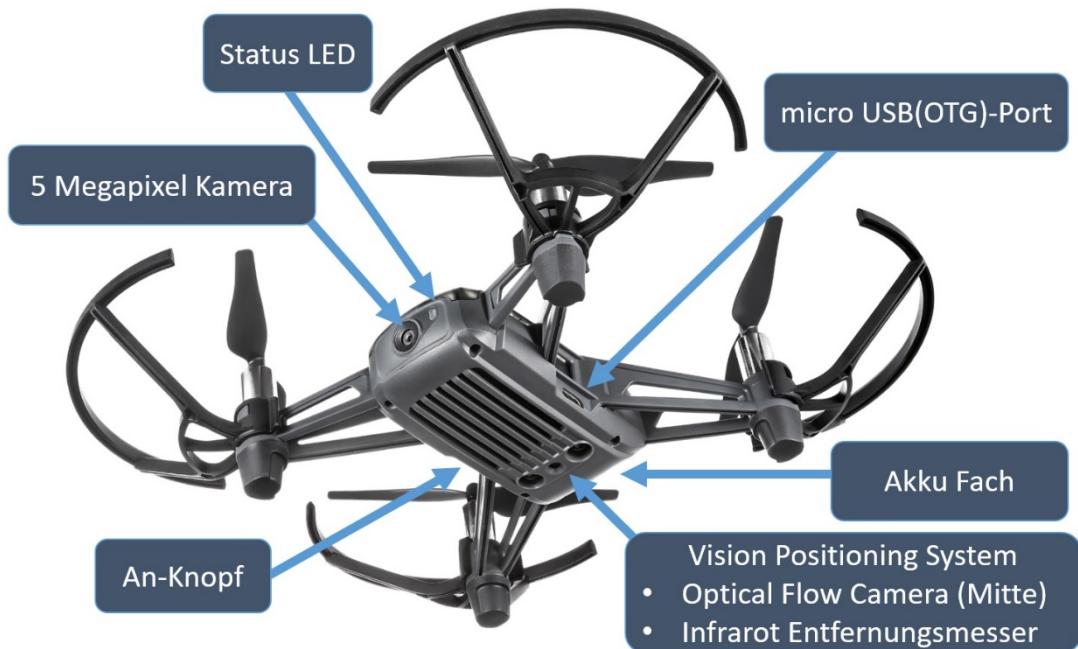


Abbildung 4: Tello EDU - Merkmale

## 2.2 ESP32 „Rucksack“

Der *ESP32 „Rucksack“* trägt den offiziellen Namen *DJI RoboMaster Tello Talent Expansion Kit* und erweitert die *Tello EDU* zur *Tello Talent*. Bis auf die rote Farbe besteht kein Unterschied zwischen der *Tello EDU mit Expansion Kit* und der *Tello Talent*.



Abbildung 5: ESP32 „Rucksack“ bzw. DJI RoboMaster Tello Talent Expansion Kit

Neben dem *ESP32* befindet sich auf dem „Rucksack“ noch eine *Red Green Blue (RGB) Light Emitting Diode (LED)* oben, die immer über *Pulse Width Modulation (PWM)* angeschlossen ist. Des Weiteren gibt es rechts am Rand oben eine Taste, die, wenn sie gedrückt wird, *General Purpose Input/Output (GPIO) Pin 34* auf *Ground (GND)* zieht.

Darunter befindet sich ein Schalter mit einem Router und einem Telefon Symbol. Dieser muss während des Praktikums immer auf Telefon gestellt sein, ansonsten kann es Probleme mit der Wireless Fidelity (*WiFi*) Verbindung geben.

Es sind *GPIO* Pins des *ESP32* herausgeführt (siehe Abb.3, unten mittig), woran standardmäßig ein Modul aufgeclipst wird, das eine 8x8 RB-LED Matrix (nur Rot Blau, Intensität jeweils 8bit) und einen eindimensionalen *Light Detection And Ranging (LiDAR)* Sensor mit einer Reichweite von maximal 2 Metern, sowie einer Auflösung von ca. einem Millimeter besitzt. Beide dieser Geräte werden über einen *Inter-Integrated Circuit Bus (I<sup>2</sup>C-Bus)* angesteuert.



Abbildung 6: LED-Matrix mit darüber sitzendem LiDAR Sensor

### 3 Style Guide

Beim Programmieren ist es wichtig einheitlich zu bleiben und sich an eine sogenannten „Style Guide“ zu halten. Hier wird eine vereinfachte Version von Microsofts *Systems Hungarian* benutzt. Darin setzen sich Variablen immer aus einem Präfix, der Information über den Datentyp der Variable gibt, und einem Namen zusammen. Der Name wird in *CamelCase* geschrieben, das heißt, dass alle Wörter aneinander geschrieben werden und der erste Buchstabe eines Wortes großgeschrieben wird. Im Folgenden Code Ausschnitt wird eine Variable definiert und initialisiert. Diese enthält Information über die Port Nummer des MQTT-Brokers. Sie ist vom Typ *16 bit unsigned Integer*.

```
const uint16_t uiMQTTPort = 1883;
```

In der Tabelle ist abgebildet, welche Bedeutung die verwendeten Präfixe haben.

Präfix	Datentyp	Beispiel
i	Integer	int iZahl = -5;
ui	Unsigned Integer	unsigned int uiPositiveZahl= 5;
b	Boolean	bool bBedingung = true;
f	Float	float fGleitkommazahl = -17.3;
d	Double	double dGenaueKommazahl = 1.602176;
c	Character	char cBuchstabe = 'a';
y	Byte	byte yAchtBit = 0x01;
sz	C-String (String Zero-Terminated)	char *szMQTTBroker = "127.0.0.1";
str	String	String strCommand = "Takeoff";
t	Typedef (selbstdefinierter Variablen-typ, hier auf Basis von enum)	FLIGHTSTATES_T tFlightState = Idle;

Auch wenn manche Leute wie Linus Torvalds diesen Style in der Linux Kernel Dokumentation als „asinine“ = „idiotisch“ bezeichnen, da der Compiler es ja sowieso wissen würde, raten wir Ihnen dringend sich daran zu halten. Auch wenn es einen minimalen Mehraufwand bedeutet, hilft es aus eigener Erfahrung anderen Leuten extrem, sich schnell in fremdem Code zurechtzufinden, vor allem falls Sie einmal Hilfe benötigen. Schließlich haben ja nicht alle Linux erfunden.

## 4 IntelliSense

Die C++ Extension bringt IntelliSense mit. Folgende Funktionen können Sie nutzen.

„Hovern“ Sie mit der Maus über einer dem Namen einer Funktion, um mehr Informationen über diese zu erhalten:

```

62
63     void Callback(char *szTopic, byte *yMessage, unsigned int uLength)
64
65     cas Callback Function that gets passed to the MQTT Library, it'll be run whenever
66     there is something new on a topic you subscribed to.
67     b Parameters:
68     def - char* szTopic
69     - byte* yMessage
70     - unsigned int uLength
71     Returns:
72     - Nothing
73     Last Change: KS 2022 03 02
74     void Callback(char *szTopic, byte *yMessage, unsigned int uLength)
75     {
76     }

```

Abbildung 7: IntelliSense: Hover Information

Die Auto vervollständigung erscheint oft an passender Stelle von allein, sie können es jedoch auch jederzeit über Strg + Leertaste aufrufen:

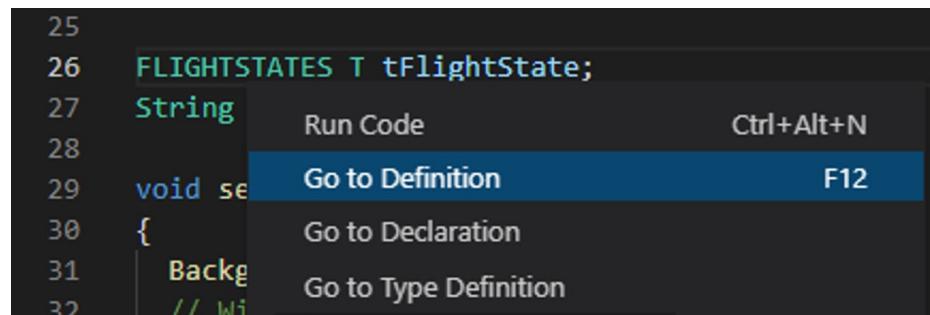
```

27 String strCommand = "";
28
29 strC
30     [?] strCommand
31     void strcasecmp
32     {
33         Ba strcasecmp
34         // strcasecmp_P
35         strchr
36         // strcmp
37         strcmp
38         Ba strcoll
39     }
40         strcpy
41         strcpy_P
42     void strcspn
43

```

Abbildung 8: IntelliSense Auto vervollständigung

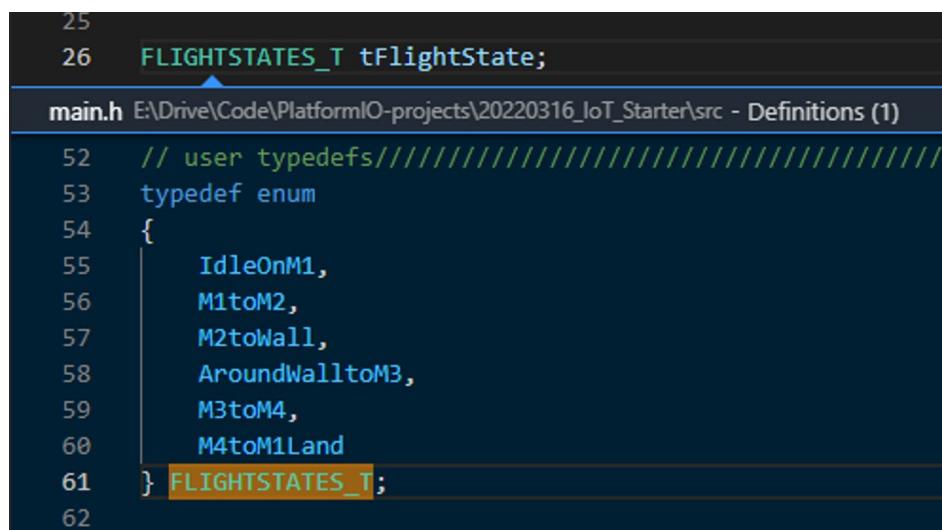
Interessiert Sie etwas von uns Geschriebenes oder aus einer Bibliothek, klicken Sie mit der rechten Maustaste darauf und wählen Sie *Go to Definition* an. Hier am Beispiel der Zustandsvariable:



A screenshot of an IDE showing a context menu for a variable. The menu items are:

- Run Code (Ctrl+Alt+N)
- Go to Definition (F12) - This item is highlighted.
- Go to Declaration
- Go to Type Definition

Abbildung 9: IntelliSense Go to Definition



A screenshot of an IDE showing an IntelliSense definition pop-up for the type `FLIGHTSTATES_T`. The pop-up lists the following enum values:

- IdleOnM1,
- M1toM2,
- M2toWall,
- AroundWalltoM3,
- M3toM4,
- M4toM1Land

Abbildung 10: IntelliSense Definition Pop-up

## 5 Anmerkung Debugging

Grundsätzlich können Sie sich im Arduino Framework Objekte der Klasse String an die Funktion `print()` oder `println()` des Objektes `Serial` übergeben. Diese werden dann über den *Universal Serial Bus (USB)* an die virtuelle *Communication (COM)* Schnittstelle an ihren Computer gesendet. Nachdem hochladen ihres Programms über den Pfeil nach rechts können Sie den *Serial Monitor* über das Steckersymbol unten links öffnen. Hier werden dann ihre Daten angezeigt. In diesem Projekt wurde bereits ein Objekt der Klasse `Serial` erstellt und mit Baud 115200, 8 Datenbits, keine Parität und einem Stop-Bit initialisiert, so ist auch Ihr *Serial Monitor* in der Datei `platformio.ini` konfiguriert, darum müssen Sie sich nicht kümmern.

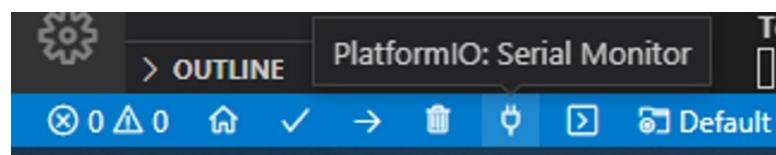


Abbildung 11: Serial Monitor Knopf

Nutzen Sie nicht `Serial.print()`, sondern das Makro `ConsolePrint()` oder `ConsolePrintln()`. Diese sind in `main.h` definiert und erweitern sich so, dass der String, den Sie in die Klammern schreiben sowohl zum seriellen Monitor als auch über *MQTT* in das Subtopic `/console` gesendet wird.

## 6 Ordnerstruktur des Starterprojekts

```

    ▼ PIO-DroneLabCourse-Star...
        > .pio
        > .vscode
    ▼ include
        C background.h
        C main.h
        ⓘ README
    ▼ lib
        > pubsubclient-master
        > RMTT_Libs-master
        ⓘ README
    ▼ PythonTello2_SIMULATOR
        ⚡ SimulatedTello2.py
    ▼ src
        C background.cpp
        C main.cpp
        > test
        ≡ -Os
        ⚡ .gitignore
        ⚡ platformio.ini

```

Abbildung 12: Ordnerstruktur des Projekts in PlatformIO

Nach öffnen des Projekts öffnet sich links der Dateibaum des Projekts.

In der Datei *platformio.ini* sind die Hardware, sowie Anweisungen für den Compiler definiert. Dies ist die Datei, die schon unter dem Punkt *Anmerkung Debugging* angesprochen wurde, da hier auch die *Baudrate* für die serielle Schnittstelle auf 115200bd gesetzt wurde.

Spezifische Bibliotheken, wie Sie hier für MQTT (*pubsubclient-master*) und die Tello Drohne (*RMTT\_Libs-master*) benötigt werden, können in den Ordner *lib* eingefügt werden. Diese können dann im jeweiligen Projekt benutzt werden. Standardbibliotheken wie *Arduino.h* oder *WiFi.h* wurden von *PIO* entsprechend der Hardwaredefinition im Hintergrund bereitgestellt und können über `#include <ibName>` eingebunden werden (vgl. Arduino IDE).

Im *PythonTello2\_SIMULATOR* – Ordner befindet sich ein Python-Skript, das in der Lage ist die zweite fliegende Drohne zu simulieren. So können Sie den kompletten Durchlauf auch nur mit einer Drohne testen, sollten Sie hardwaretechnisch beschränkt sein.

Im *src* Ordner befinden sich die Programmdateien des Projekts. Die dazugehörigen header-Dateien befinden sich im *include* Ordner. Für Sie interessant ist nur die Datei *main.cpp*. **Nur dort müssen Sie während des Praktikumstermins etwas bearbeiten.** Sie können trotzdem in alle anderen Dateien, auch die der Bibliotheken, zum Verständnis hineinschauen, auch wenn das für die Aufgabe nicht von Nöten ist. Bitte verändern Sie dort jedoch nichts.

## 7 Überblick main.cpp

Dies ist nur eine Erklärung des Startzustands, Sie müssen in diesem Abschnitt nichts aktiv machen.

### 7.1 Globale Variablen

Nach dem `#include`-tag der `main.h` Header-Datei finden Sie eine Reihe von Variablen bzw. Objekte. Diese sind schon deklariert, da Sie auch in den `background` Dateien verwendet werden. Hier müssen Sie nur die Initialisierung auf ihre Werte festlegen.

### 7.2 setup()

Nachdem wir der Einfachheit halber das Arduino Framework benutzen, ist der Einstieg nicht wie gewohnt in `int main(void)` definiert, sondern in `void setup(void)`. Danach wird auch automatisch `void loop(void)` unendlich oft ausgeführt, ohne dass es in unserem Programm aufgerufen wird. In `setup()` werden zwei Funktionen aus der `background.cpp` Datei ausgeführt, dazwischen werden Sie die WiFi- und MQTT-Verbindung initialisieren.

### 7.3 loop()

Hier soll als erstes die `MQTT`-Routine aufgerufen werden, danach wird eine Methode aus der `background.cpp` Datei ausgeführt, welche sich um Dinge wie die obere LED und Sonderfälle wie das Wiederverbinden von verlorener WiFi-Verbindung kümmert. Zudem gibt es eine Kopie des Zustandautomaten der `main.cpp`, in dem über die LED-Matrix zum Beispiel angezeigt wird, in welchem Zustand sich der Automat momentan befindet. Alles Dinge, die über den Rahmen dieses Praktikums hinausgehen, bzw. nicht im Fokus von Internet of Things stehen.

Danach kommt ein leerer Zustandsautomat, in den Sie später Ihre Logik einfügen werden. Die Zustandsvariable `tFlightstate` ist vom `enum` bzw. `integer` basiertem Datentyp `FLIGHTSTATES_T`. Dieser wurde in `main.h` definiert.

### 7.4 Callback()

Diese Methode wird, nachdem Sie es nachher einprogrammiert haben, jedes Mal von der Bibliothek aufgerufen, wenn eine neue MQTT Nachricht an eine der von Ihnen abonnierten Topics gesendet wurde. Der Kopf dieser Methode ist durch die MQTT Bib-

liohek festgelegt. Es wird Ihnen das Topic als *C-String* (nicht als Objekt der Klasse String) und die Nachricht in Form eines Pointers auf das erste Element eines Arrays vom Datentyp *byte* und die Anzahl der Elemente in diesem Array als *unsigned int* übergeben.

Bis hierher sollten Sie das Praktikum im Vorfeld vorbereiten. Überprüfen Sie anhand folgender Liste, ob Sie alle Punkte abgearbeitet haben.

- Sie sind mit den Grundlagen von *MQTT* vertraut
- Sie sind mit den Grundlagen der *Tello Talent* vertraut
- Sie kennen die Trennung der *Tello EDU* und dem *ESP32 „Rucksack“*
- Sie sind mit dem benutzten Style Guide vertraut und gedenken ihn zu benutzen :)
- Sie wissen wie Sie einfache Konsolenausgaben sowohl mit als auch ohne USB-Verbindung tätigen können
- Sie haben einen Überblick über die Datei *main.cpp* und die Möglichkeiten von *IntelliSense* verstanden

## Abbildungsverzeichnis

Abbildung 1: Funktionsweise MQTT, Vorlesung Neve S.259.	5
Abbildung 2: Nachrichten und Topics.....	5
Abbildung 3: Tello Talent.....	6
Abbildung 4: Tello EDU - Merkmale.....	7
Abbildung 5: ESP32 „Rucksack“ bzw. DJI RoboMaster Tello Talent Expansion Kit ....	7
Abbildung 6: LED-Matrix mit darüber sitzendem LiDAR Sensor.....	8
Abbildung 7: IntelliSense: Hover Information .....	10
Abbildung 8: IntelliSense Auto vervollständigung .....	10
Abbildung 9: IntelliSense Go to Definition .....	11
Abbildung 10: IntelliSense Definition Pop-up .....	11
Abbildung 11: Serial Monitor Knopf.....	12
Abbildung 21: Ordnerstruktur des Projekts in PlatformIO.....	13

Alle Abbildungen dieses Dokumentes sind Screenshots von Windows, Linux, Mac, Visual Studio Code oder MQTT-Explorer und wurden vom Verfasser erstellt. Einzige Ausnahme ist Abbildung 1, diese stammt aus dem Vorlesungsskript „Internet of Things“, Seite 259, Frau Prof. Dr. rer. nat. Antje Neve.