

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

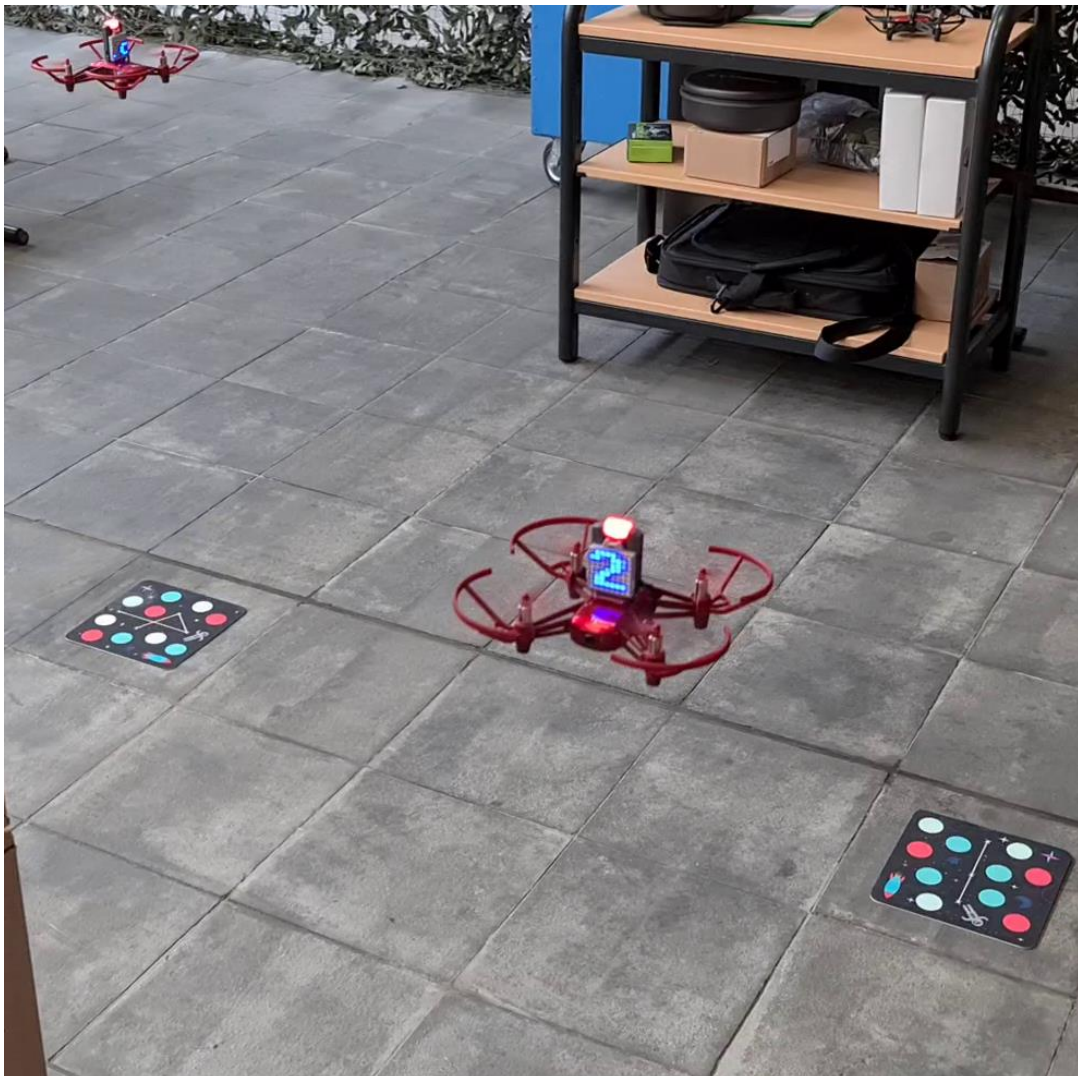
Frühjahrssemester 2022
Verfasser: Karl Scholz

Internet of Things Praktikum

Teil 2

Multikopter als verteilte intelligente Systeme

25.05.2022



Arbeitsanweisung & Praktikumsablauf

WICHTIG

Um Ihnen während des Praktikums möglichst viel fachliches Wissen vermitteln zu können, bitten wir Sie folgende Punkte, wie zum Beispiel die Einrichtung der Softwares, als Vorbereitung vor dem eigentlichen Praktikumstermin durchzuführen.

WICHTIG

Vor dem Praktikumstermin:

Arbeiten Sie im Vorfeld des Praktikums die Abschnitte bis einschließlich Punkt 9 *Überblick main.cpp* zuhause durch. Sie müssen bei den meisten Punkten nur etwas lesen, aktiv werden müssen Sie nur bei folgenden Punkten:

4 Vorbereitung

Hier wird beschrieben, wie Sie die IDE installieren und einrichten.

7 Hardwaredaten laden

Anschließend sollen Sie in Punkt 6 die Hardwaredaten laden und das leere Projekt einmal erfolgreich Kompilieren.

9 Überblick main.cpp

Zuletzt sollen Sie noch das Projekt laden und öffnen, sodass Sie bei ihrem Praktikums-termin sofort loslegen können.

Während des Praktikumstermins

Alle anderen Punkte.

Sollten sich bei der Vorbereitung Fragen ergeben, können Sie diese gerne im Vorfeld adressieren und per Mail an wiwebgf250@bundeswehr.org schicken. 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

| | |
|--|-----------|
| Arbeitsanweisung & Praktikumsablauf | 2 |
| Inhaltsverzeichnis | 3 |
| Abkürzungsverzeichnis | 5 |
| 1 Grundlagen..... | 6 |
| 1.1 Multikoptersystem..... | 6 |
| 1.1.1 Tello EDU | 6 |
| 1.1.2 ESP32 „Rucksack“ | 7 |
| 1.2 Der Versuchsaufbau | 9 |
| 1.2.1 Mission Pads | 9 |
| 1.2.2 Positionierung der Mission Pads | 11 |
| 2 Szenario und Missionsablauf | 12 |
| 3 Style Guide..... | 13 |
| 4 Vorbereitung | 14 |
| 4.1 IDE | 14 |
| 4.2 Extensions für die IDE | 14 |
| 4.3 MQTT Debugging Programm | 14 |
| 5 IntelliSense..... | 15 |
| 6 Anmerkung Debugging..... | 17 |
| 7 Hardwaredaten laden | 18 |
| 8 Einrichten des Multikopter IoT Projekts | 19 |
| 8.1 Vorbereitung | 19 |
| 8.2 Öffnen des Projekts | 19 |
| 8.3 Ordnerstruktur..... | 20 |
| 9 Überblick main.cpp..... | 21 |
| 9.1 Globale Variablen..... | 21 |
| 9.2 setup()..... | 21 |
| 9.3 loop() | 21 |
| 9.4 Callback()..... | 21 |

| | | |
|-----------|---|-----------|
| 10 | Aufgaben | 23 |
| 10.1 | Vervollständigen der Variablen | 23 |
| 10.2 | Herstellen der WiFi-Verbindung | 24 |
| 10.3 | Schreiben der MQTT Callback Funktion..... | 26 |
| 10.4 | Verbinden mit dem MQTT Broker | 28 |
| 10.5 | Callback Funktion testen..... | 30 |
| 11 | Fliegender Programmabschnitt | 33 |
| 11.1 | Kurzerklärung SDK..... | 33 |
| 11.2 | Zusammenfassung wichtiger SDK-Kommandos | 35 |
| 11.3 | Zustandsautomat..... | 36 |
| | Abbildungsverzeichnis..... | 37 |

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 |
| 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 Grundlagen

1.1 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 1: Tello Talent

1.1.1 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.

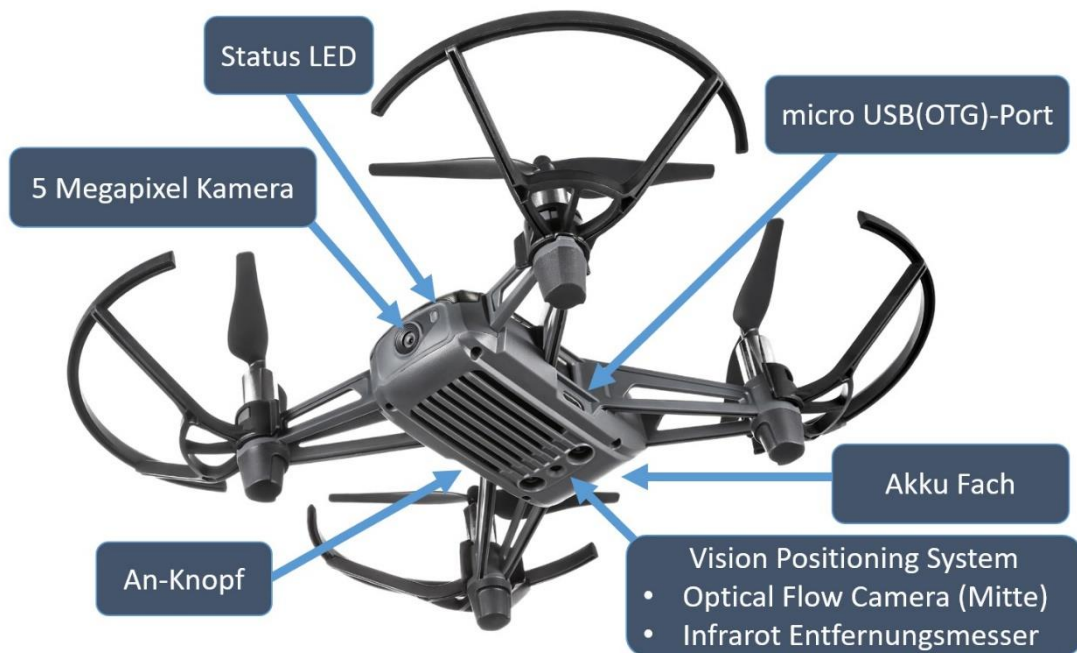


Abbildung 2: Tello EDU - Merkmale

1.1.2 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 3: 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 (I²C-Bus)* angesteuert.



Abbildung 4: LED-Matrix mit darüber sitzendem LiDAR Sensor

1.2 Der Versuchsaufbau

1.2.1 Mission Pads

Die *Tello EDU* arbeitet mit sog. *Mission Pads* (MP), welche sie mit ihrer Optical Flow Kamera erkennen kann. Diese 20 x 20 cm Matten sind mit den Zahlen von 1-8 kodiert.

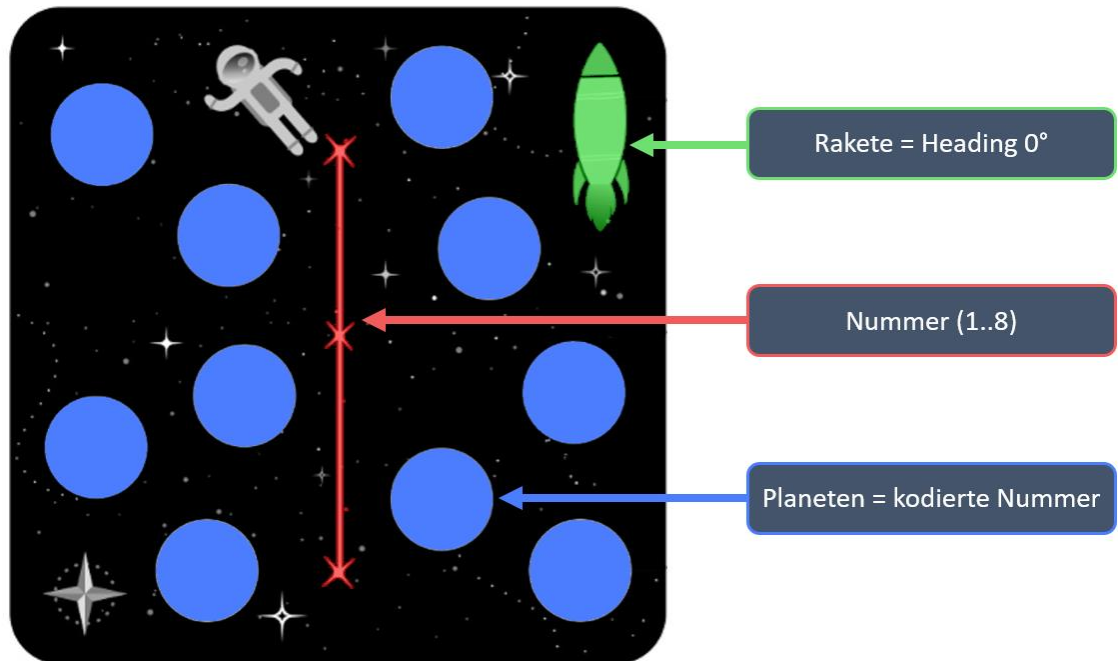


Abbildung 5: Elemente des Mission Pads

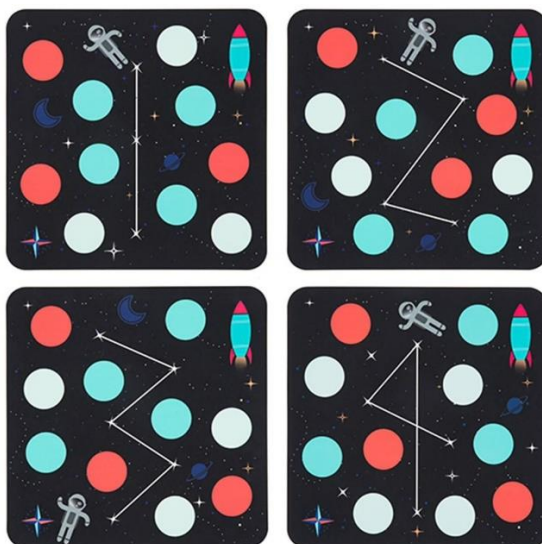


Abbildung 6: kodierte Mission Pads 1-4

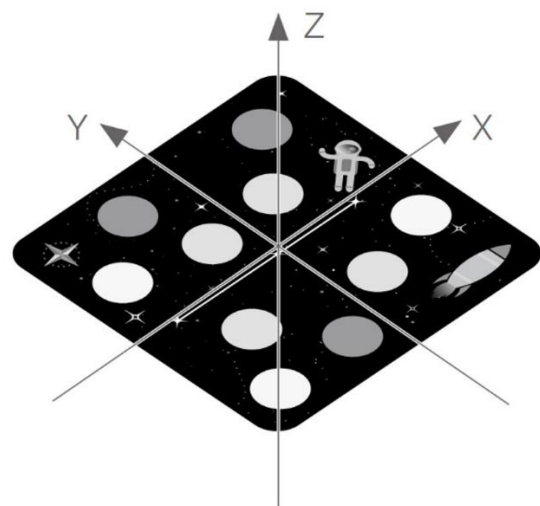


Abbildung 7: Koordinatensystem des Mission Pads

Kommandos für Mission Pads

Für die Mission Pads gibt es zwei wichtige Kommandos:

1. Go Kommando


```
"go <X> <Y> <Z> <Speed> <MP>"
"go 0 0 50 100 m-2" //Beispiel: Auf 50cm Höhe mittig Ausrichten
```

Mit dem *go* Kommando lässt sich die Tello absolut an einer Position über dem Mission Pad positionieren. Ein Anwendungsbeispiel wäre das genaue Ausrichten nach dem Startvorgang mittig über dem *Mission Pad*.

2. Jump Kommando

```
"jump <X> <Y> <Z> <Speed> <Heading> <currentMP> <targetMP>"
"jump 100 0 0 100 0 m1 m2");
//Beispiel M2 liegt ca. einen Meter vor M1
```

Das *jump* Kommando bedeutet, dass die *Tello EDU* erst einen groben Flug zur Position X, Y relativ vom jetzigen Mission Pad, auf der Höhe Z ausführt. Dort angekommen sucht sie das Ziel-Mission-Pad, richtet sich mittig aus und dreht sich, sodass die Frontkamera relativ zum Raketensymbol den Winkel <Heading> einschließt.

|  <MP> | Bedeutung |
|--|--|
| m1 | Mission Pad 1 |
| m2 | Mission Pad 2 |
| ... | ... |
| m8 | Mission Pad 8 |
| m-1 | Zufälliges Mission Pad (sinnlose Funktion, da Sichtbereich sehr eingeschränkt) |
| m-2 | Das nächste Mission Pad |

Informationen zu weiteren Befehlen → *Tello_SDK_3.0_User_Guide_en.pdf*

1.2.2 Positionierung der Mission Pads

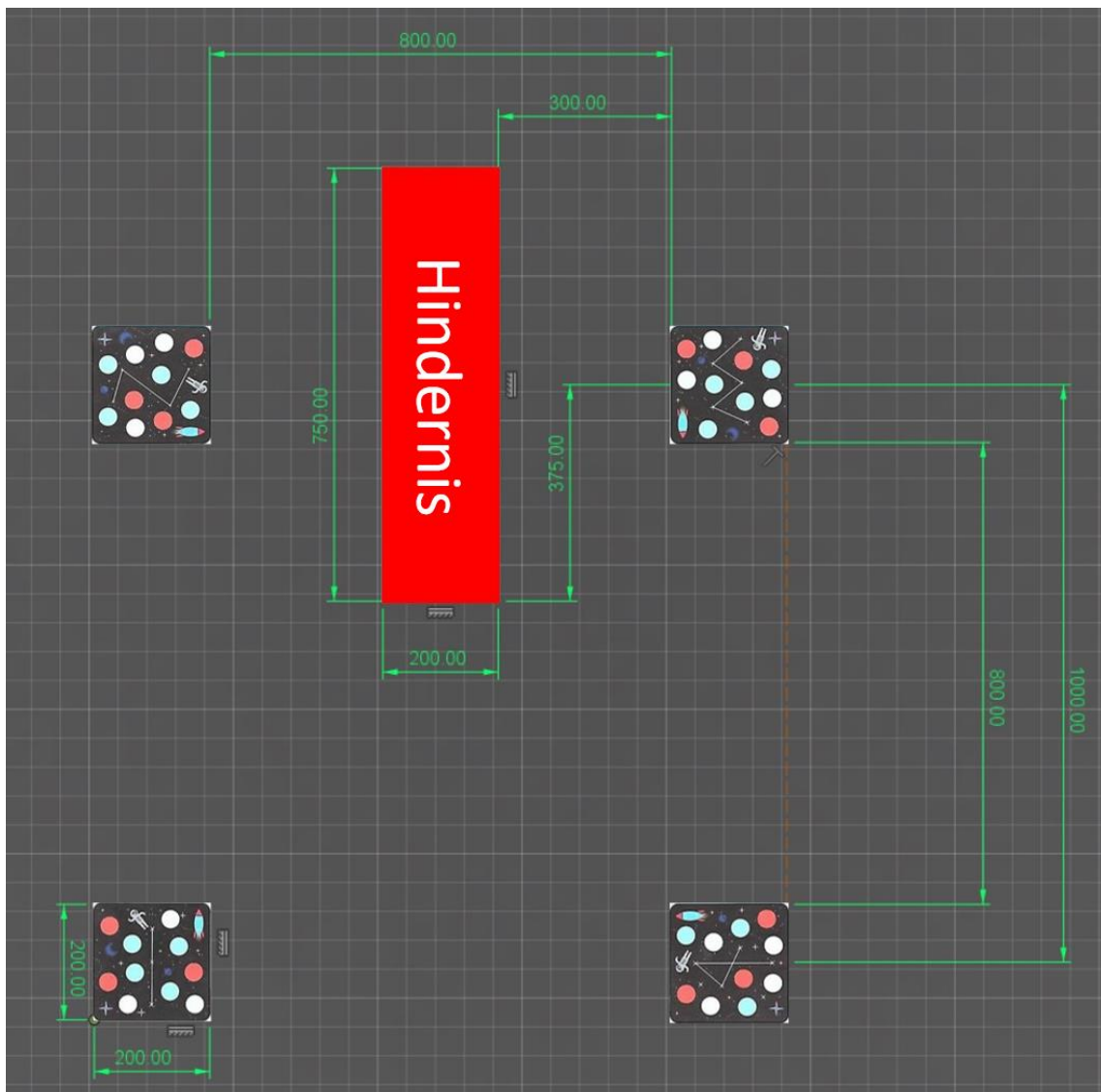


Abbildung 8: bemaßter Versuchsaufbau [mm]

Zu Beginn ihres Praktikums werden Sie aufgefordert die Mission Pads wie in der Abbildung gezeigt auf dem Boden zu platzieren und den Weg zwischen Mission Pad 2 und 3 mit einem Hindernis, welches höher ist als Sie fliegen wollen, zu blockieren. Die Mission Pad Erkennung funktioniert auf 80 cm Höhe am besten.

2 Szenario und Missionsablauf

Sie werden in diesem Internet of Things Praktikum allein oder in einer Gruppe eine Drohne programmieren, die einen Parcours bewältigen soll. Allerdings müssen Sie sich diesen Parcours mit einem Partner bzw. einer Partnergruppe teilen. Um eine Kollision zu vermeiden, müssen ihre Drohnen miteinander über das *Message Queuing Telemetry Transport (MQTT)* Protokoll kommunizieren. Dafür bekommen Sie eine *AircraftID*, zum Beispiel *UniBwTello_3*, sodass sie im Netzwerk eindeutig sind.

Ablauf:

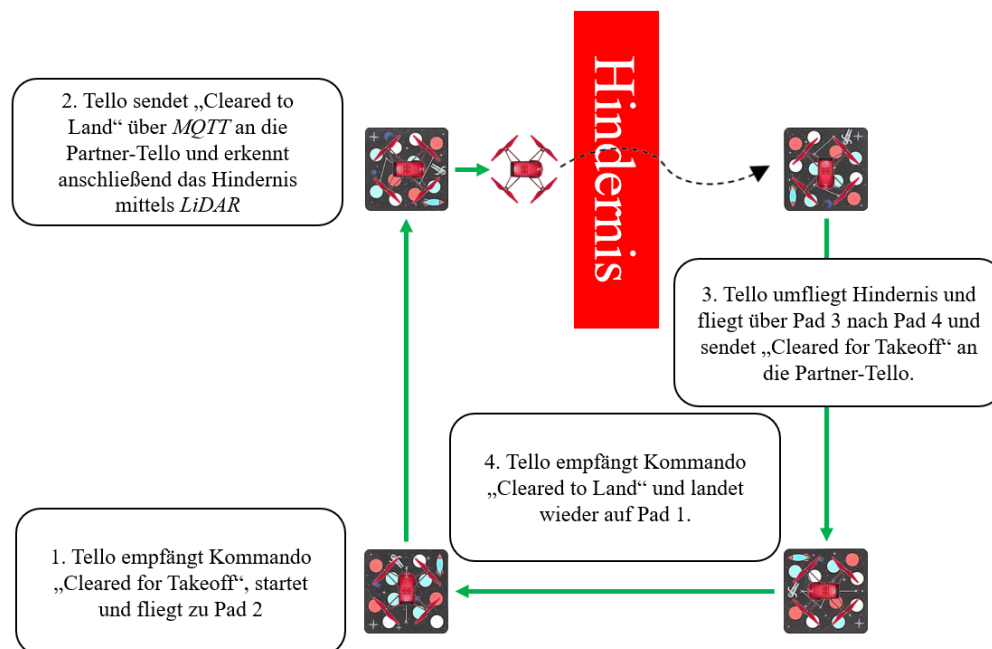


Abbildung 9: Szenario und Missionsablauf

Tello1 wird auf das *MPI* gestellt, bekommt über das *MQTT* Protokoll manuell den Startbefehl und fliegt anschließend um den Kurs. Sie erkennt das Hindernis mit dem *LiDAR* Sensor und umfliegt es automatisch. Währenddessen wird *Tello2* auf das *MPI* gestellt und angeschaltet. Kommt die *Tello1* zum *MP4*, wird Sie darüber schwebend stehen bleiben und mittels *MQTT* der *Tello2* den Startbefehl geben. *Tello2* startet dann und gibt *Tello1*, sobald Sie weg ist, die Landeerlaubnis. *Tello1* landet und die beiden Rollen sind getauscht. Theoretisch sollte der Ablauf dann unbegrenzt wiederholbar sein und die beiden *Tellos* den Parcours abfliegen, das Hindernis umfliegen und miteinander kommunizieren.

3 Style Guide

Beim Programmieren ist es wichtig einheitlich zu bleiben und sich an einen 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 | <code>int iZahl = -5;</code> |
| ui | Unsigned Integer | <code>unsigned int uiPositiveZahl = 5;</code> |
| b | Boolean | <code>bool bBedingung = true;</code> |
| f | Float | <code>float fGeitkommazahl = -17.3;</code> |
| d | Double | <code>double dGenaueKommazahl = 1.602176;</code> |
| c | Character | <code>char cBuchstabe = 'a';</code> |
| y | Byte | <code>byte yAchtBit = 0x01;</code> |
| sz | C-String (String Zero-Terminated) | <code>char *szMQTTBroker = "127.0.0.1";</code> |
| str | String | <code>String strCommand = "Takeoff";</code> |
| t | Typedef (selbstdefinierter Variablentyp, hier auf Basis von enum) | <code>FLIGHTSTATES_T tFlightState = Idle;</code> |

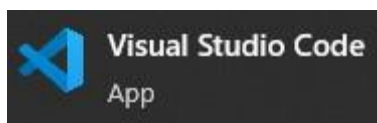
Auch wenn manche Leute wie Linus Torvalds diesen Style in der Linux Kernel Dokumentation als „asinine“ = „idiotisch“ bezeichnen, da der Compiler es ja sowieso wissen würde, raten wir Ihnen dringend sich daran zu halten. Auch wenn es einen minimalen Mehraufwand bedeutet, hilft es aus eigener Erfahrung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

4 Vorbereitung

Alle benötigten Programme sind Freeware. Wenn Sie zum Kauf aufgefordert werden, machen Sie etwas falsch. Benutzen Sie ausschließlich kostenlose Software, Käufe werden Ihnen nicht erstattet.

4.1 IDE

Nutzen Sie einen Rechner mit *Windows*, *MacOS* oder *Linux* und installieren sich das kostenlose *Integrated Development Environment (IDE) Visual Studio Code (VSCode)*.



Download → <https://code.visualstudio.com/>

Abbildung 10: Logo von Visual Studio Code

4.2 Extensions für die IDE

Navigieren Sie (links am Rand) in *VSCode* zum Punkt *Extensions* und suchen Sie nach *PlatformIO (PIO)*, installieren Sie diese und alle weiteren Extensions, die Ihnen während der Installation von *PIO* vorgeschlagen werden. (*C/C++*, *IntelliSense*, etc.)

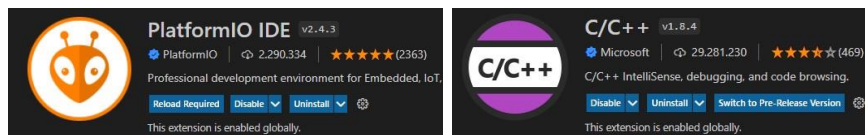
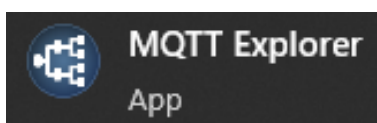


Abbildung 11: Logos der PlatformIO und C/C++ Extensions

4.3 MQTT Debugging Programm

Es gibt Programme, die sich ganz normal als Client beim *MQTT-Broker* anmelden, alles abonnieren und die Nachrichten dann schön graphisch aufbereiten. Dazu bieten Sie dem Benutzer auch noch die Möglichkeit eigene Nachrichten in jedes beliebige Topic zu schicken. Unsere Empfehlung ist der *MQTT Explorer* von *Thomas Nordquist*, der ebenfalls für alle drei Betriebssysteme kostenlos erhältlich ist.



Download → <http://mqtt-explorer.com/>

Abbildung 12: MQTT Explorer Logo

5 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:

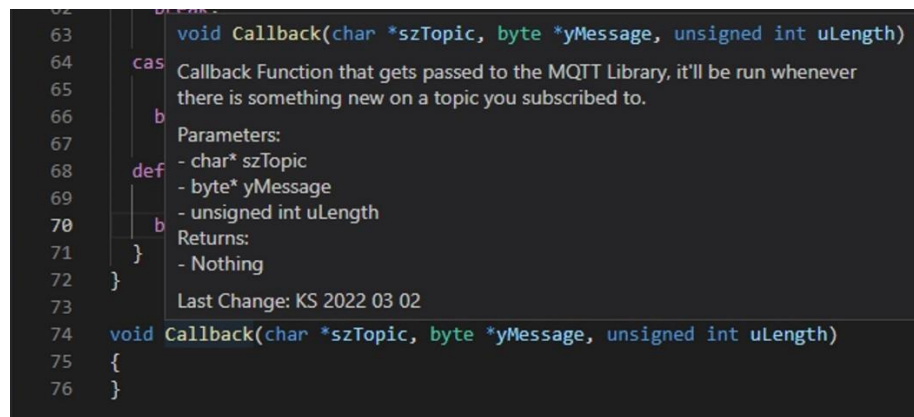


Abbildung 13: IntelliSense: Hover Information

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

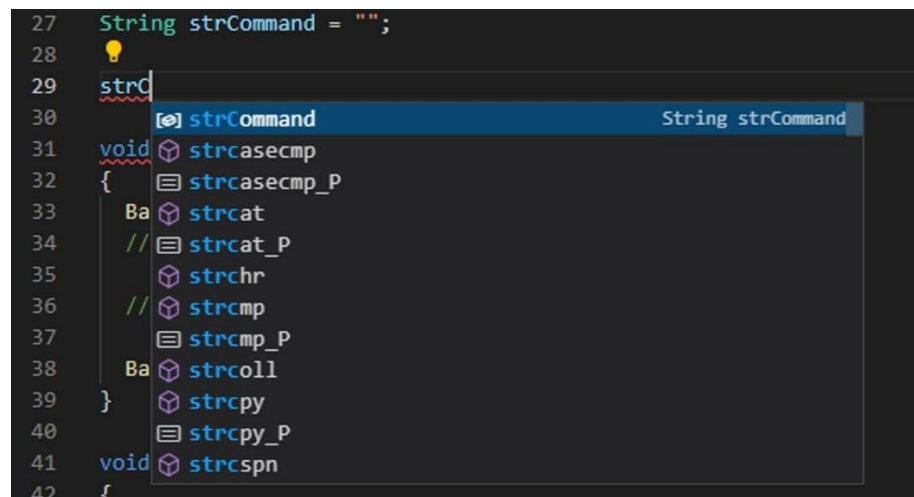


Abbildung 14: IntelliSense Autovervollständigung

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:

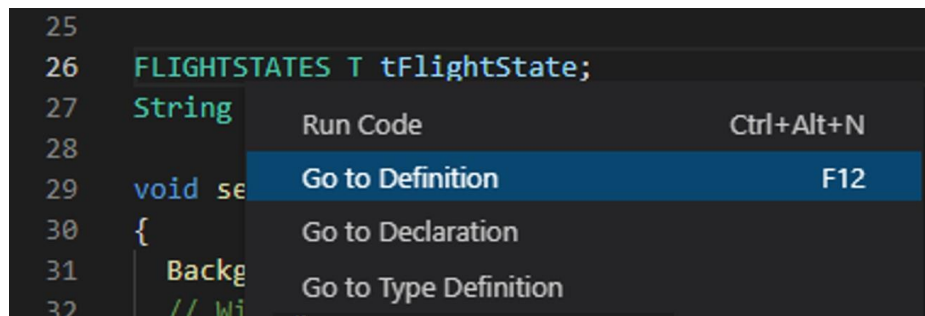


Abbildung 15: IntelliSense Go to Definition

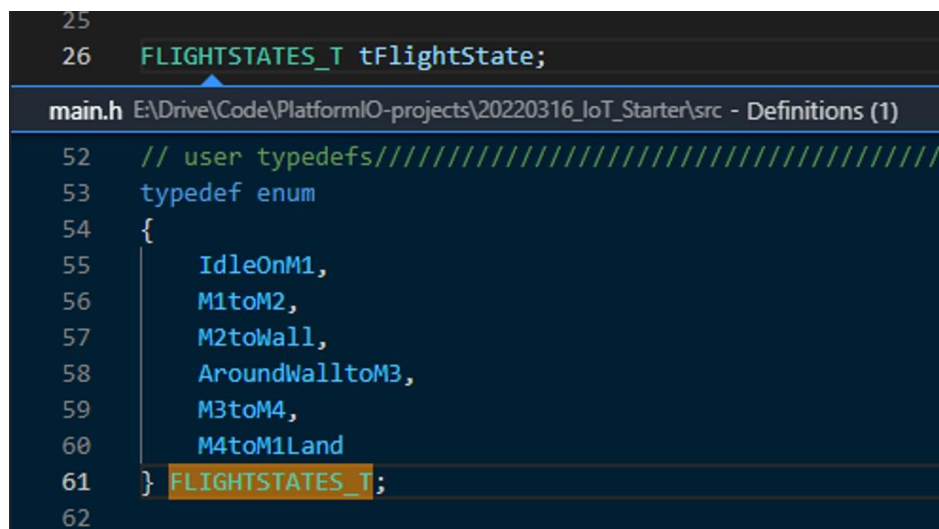


Abbildung 16: IntelliSense Definition Pop-up

6 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.

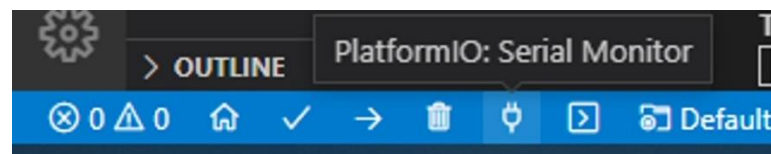


Abbildung 17: Serial Monitor Knopf

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.

7 Hardwaredaten laden

Arbeiten Sie in den folgenden Abschnitten wieder parallel zur Anleitung mit und versuchen Sie die Erklärungen auch im Code zu verstehen.

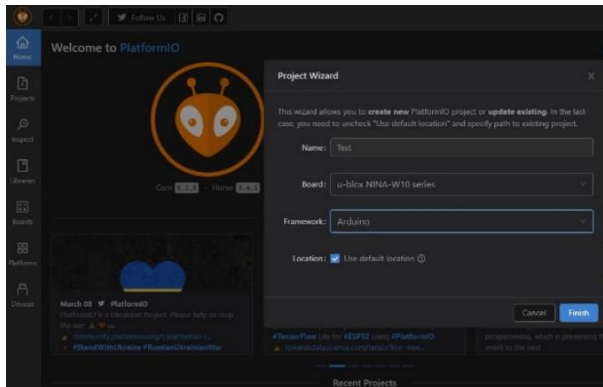


Abbildung 18: PlatformIO Project Wizard

Nach beendeter Installation von *PIO* klicken Sie unten links auf das kleine Haus-Symbol um die Startseite *PI-O:HOME* zu öffnen. Dort erstellen Sie über *New Project* ein neues Projekt mit beliebigem Namen. Wählen Sie als Board „*u-blox NINA-W10 series*“ („*Espressif 32*“ Plattform) und bei Framework „*Arduino*“ aus.

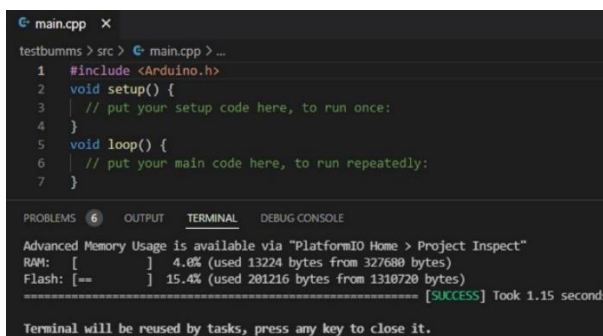


Abbildung 19: erste erfolgreiche Kompilierung

Das Erstellen kann eine Weile dauern, da nun die hardwarespezifischen Daten heruntergeladen werden. Kompilieren Sie anschließend mit dem Haken (*Build*) unten links das gerade erstellte Projekt.

8 Einrichten des Multikopter IoT Projekts

8.1 Vorbereitung

In diesem Praktikum im Rahmen der Vorlesung *Internet of Things* wird der Schwerpunkt auf die Kommunikation bzw. Netzwerkeinbindung gelegt. Zusätzlich werden Sie im Rahmen des Themengebiets *Edge Computing* lokale Logik programmieren. Daher werden alle für Sie im Kontext der Vorlesung nicht relevanten Probleme bereits fertig ausprogrammiert sein. Diese haben Sie in Form einer „Starterprojects“ als ZIP-Datei bekommen, oder von Github heruntergeladen.

<https://github.com/UniBw-ETTI-2-IoT/DroneLabCourse-Starter.git>

Entpacken Sie diese und verschieben Sie den darin befindenden Ordner an einen beliebigen Ort in ihrem Dateisystem. Dieser Ordner ist das Projekt.

8.2 Öffnen des Projekts

Gehen Sie auf die Startseite von *PIO* (Haus ganz links unten) und wählen den Punkt *Open Project*. Navigieren Sie **IN** ihr Projekt hinein auf Ebene des *src* – Ordners. Bestätigen Sie nun das Öffnen.

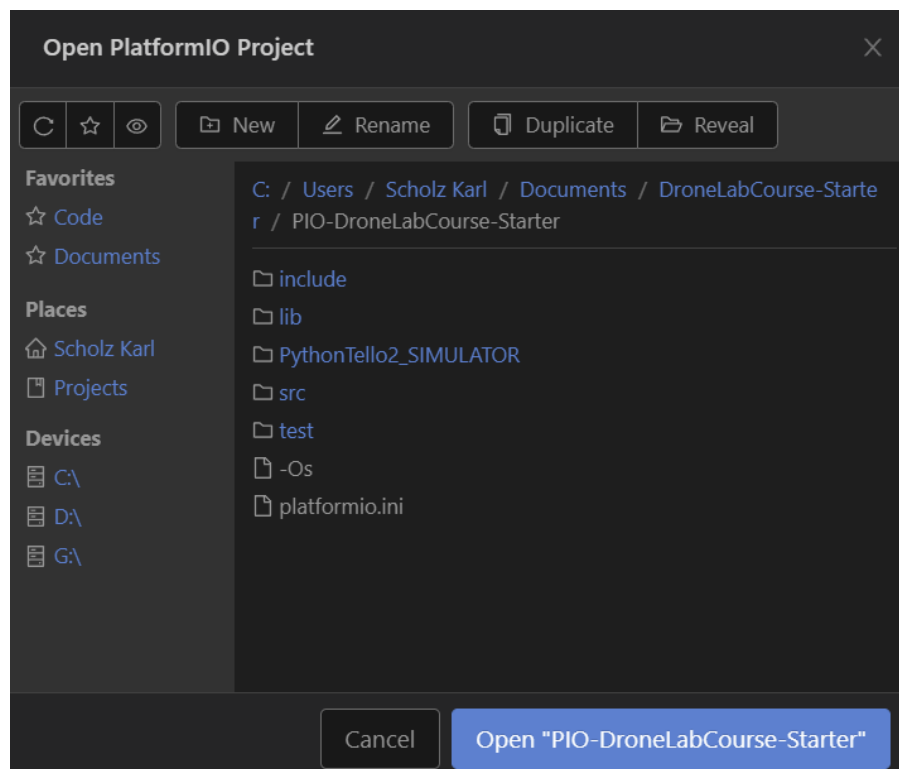


Abbildung 20: PlatformIO: Open Project

8.3 Ordnerstruktur

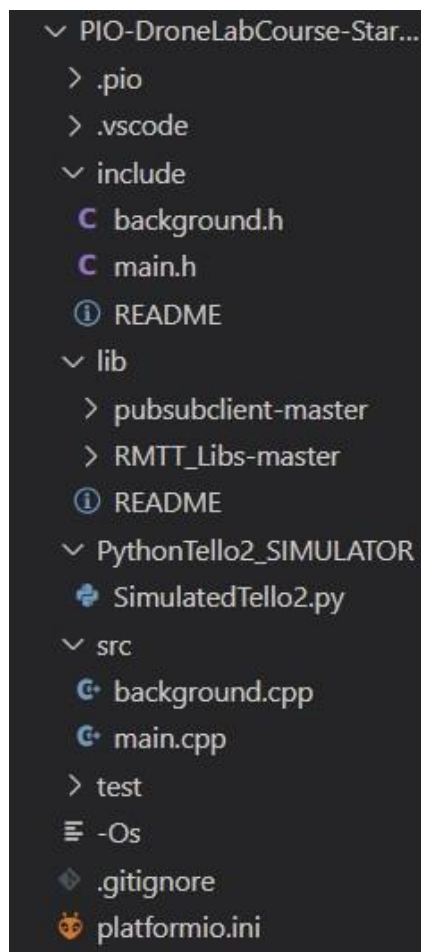


Abbildung 21: Ordnerstruktur des Projekts in PlatformIO

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 einfach über *#include <BibName>* eingebunden werden (vgl. Arduino IDE).

Im *PythonTello2_SIMULATOR* – Ordner befindet sich ein Python-Skript, dass 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.

9 Überblick main.cpp

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

9.1 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.

9.2 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.

9.3 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 Zustandsautomaten 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.

9.4 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-

liothek 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. Die nachfolgenden Aufgaben sind während des Praktikums zu bearbeiten. Überprüfen Sie anhand folgender Liste, ob Sie alle Punkte abgearbeitet haben.

- Sie kennen das Szenario und den Ablauf der Mission ☐
- Sie sind mit den Grundlagen der Tello Talent vertraut ☐
- Sie haben *Visual Studio Code* mit den Extensions installiert ☐
- Sie haben die Hardwaredaten geladen und erfolgreich kompiliert ☐
- Sie haben das Starter-Projekt geöffnet und ohne Veränderung erfolgreich kompiliert ☐
- Sie haben das *MQTT* Debugging Programm installiert ☐
- Sie haben einen Überblick über die Datei *main.cpp* und die Möglichkeiten von *IntelliSense* verstanden ☐

10 Aufgaben

Öffnen Sie 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 schiefgelaufen ist, da sich das Programm im „Starter“-Zustand kompilieren lassen muss.

10.1 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
11  WiFiClient wifi_client;
12  PubSubClient mqtt_client(wifi_client);
13
14  RMTT_Protocol tt_sdk;
15  RMTT_TOF tt_lidar;
16
17  const char *szAircraftID = "";
18  const char *szPartnerAircraftID = "";
19
20  const char *szSSID = "";
21  const char *szPassword = "";
22
23  const char *szMQTTBroker = "";
24  const uint16_t uiMQTTPort = 0;
25
26  FLIGHTSTATES_T tFlightState;
27  String strCommand = "";
28
```

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.

Abbildung 22: `main.cpp` Startzustand der Sektion der globalen Variablen

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.

Nun soll die Zustandsvariable des Automaten initial festgelegt werden,

10.2 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 23: 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 10.1 programmiert haben damit sie im Router Interface eindeutig identifizierbar sind.

Starten Sie den Verbindungsvorgang 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 manchmal kann es mehrere Definitionen einer Funktion geben, sog. überlagerte 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());
```

10.3 Schreiben der MQTT Callback Funktion

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

Abbildung 24: 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:

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

Kontrollstruktur:

In diese, 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 (explizierter Typcast) 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.

10.4 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
63     //BackgroundLoop
64     BackgroundLoop();
65     //State Machine
66     switch (tFlightState)
67     {
68     case IdleOnM1:
69
70         break;
71
72     case M1toM2:
73
74         break;
75
76     case M2toWall:
77
78         break;
79
80     case AroundWalltoM3:
81
82         break;
83
84     case M3toM4:
85
86         break;
87
88     case M4toM1Land:
89
90         break;
91
92     default:
93
94         break;
95     }
96 }
```

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 25: main.cpp Startzustand der Funktion *loop()*

10.5 Callback Funktion testen



Abbildung 26: 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 27: 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.

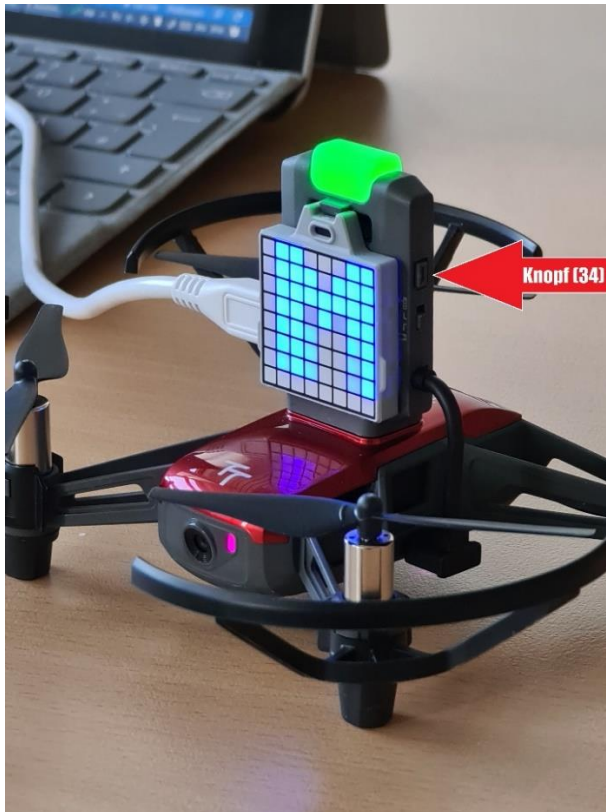


Abbildung 28: Programmstart mit Knopfdruck bestätigt

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.

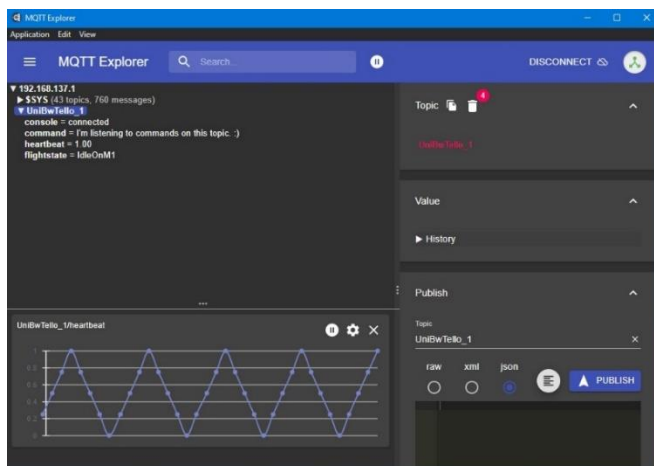


Abbildung 29: MQTT-Explorer nachdem das Programm gestartet wurde

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 aufgehängt hat.

/flightstate zeigt den aktuellen Wert der Zustandsvariable an.

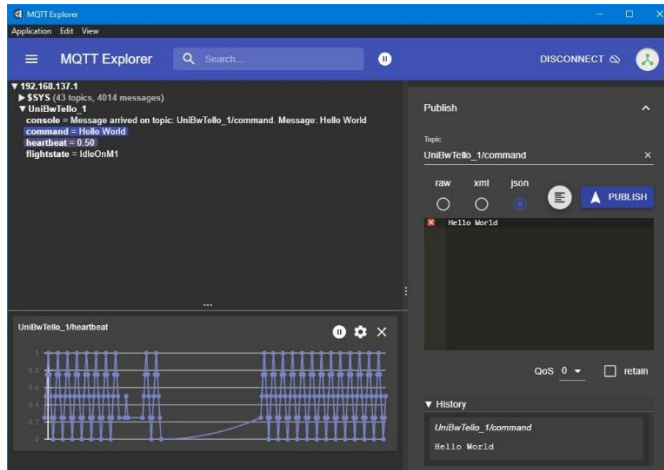


Abbildung 30: manuelle Veröffentlichung von "Hello World" in
/command

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.

11 Fliegender Programmabschnitt

11.1 Kurzerklärung SDK

Zuletzt geht es noch um den Zustandsautomaten und die eigentliche Logik hinter dem Ablauf, wie er in der Einleitung im letzten Unterpunkt skizziert wurde. Folgende Informationen fehlen Ihnen für die Umsetzung noch:

- SDK Kommandos an die Tello EDU schicken

Alle SDK Kommandos, die sie in der Datei *Tello_SDK_3.0_User_Guide_en.pdf* finden, können Sie an die Funktion

```
int RMTT_Protocol::sendTelloCtrlMsg(char *cmd_str)
```

des in *main.cpp* erstellten Objekts *tt_sdk* der Klasse *RMTT_Protocol* übergeben. Diese Funktion sendet das Kommando nicht einfach, sondern stellt auch sicher, dass es ausgeführt wird. So lange blockiert die Funktion, das heißt, dass die nächste Zeile ihres Programms erst ausgeführt wird, wenn die Ausführungsbestätigung von der *Tello* zurückgekommen ist. Wenn Sie zum Beispiel das Kommando

```
tt_sdk.sendTelloCtrlMsg("takeoff");
```

schicken und in der nächsten Zeile

```
iZahl = 3;
```

steht, dann wird die Variable *iZahl* erst nachdem die *Tello* ihre Motoren angeschaltet, gestartet und sich auf 50 cm stabilisiert hat auf den Wert 3 gesetzt.

Wenn Sie einen Befehl ausführen wollen, auf den es keinen Rückgabewert gibt, z.B.:

| | | |
|------------|---|-------------|
| rc a b c d | Set the lever force values for the four channels of the remote control. a: roll (-100 to 100) b: pitch (-100 to 100) c: throttle (-100 to 100) d: yaw (-100 to 100) | No response |
|------------|---|-------------|

Abbildung 31: Auszug SDK: "rc" Kommando

Dann nutzen Sie nicht die *sendTelloCtrlMsg()* Funktion, weil die ja auf einen Rückgabewert wartet, sondern

```
tt_sdk.SendCMD("rc 0 20 0 0");
```

Dies würde zum Beispiel „Fliege mit 20% Geschwindigkeit nach vorne“ bedeuten. Dieser Flugbefehl ist als einer der wenigen relativ. Absolute Flugbefehle wie zum Beispiel:

```
tt_sdk.sendTelloCtrlMsg("forward 20");
```

benutzen den nach unten gerichteten *Optical Flow Sensor* um 20 cm vorwärts zu fliegen.

- LiDAR Sensor auslesen

Das Objekt *tt_lidar* der Klasse *RMTT_TOF* wird im *Background* verwaltet und hat eine Funktion, um den Wert des *LiDAR* Abstands in **Millimetern** als *unsigned integer* auszugeben.

```
uint16_t RMTT_TOF::ReadRangeSingleMillimeters(void)
```

- LiDAR Daten auf LED-Matrix anzeigen lassen

In *Background.cpp* ist folgende Funktion definiert:

```
void DisplayLidarOnMatrix(float fRange)
```

Rufen Sie diese auf und übergeben Sie dieser einfach einen Wert vom Datentyp *float* in der Einheit **Meter** und sie wird den Abstand visuell auf der *LED*-Matrix darstellen.

11.2 Zusammenfassung wichtiger SDK-Kommandos

1. Abhebe Kommando

```
"takeoff" //Tello startet und schwebt auf 50 cm Höhe über dem Startpunkt
```

2. Lande Kommando

```
"land" //Tello sinkt und stellt die Motoren ab sobald sie am Boden ist
```

3. Bewege Kommandos

```
"<Direction> <x in cm [20..500]>"  
"forward 100" //Beispiel: Einen Meter nach vorne fliegen  
//<Direction> = up, down, left, right, forward, back
```

4. Dreh Kommando

```
"<Direction> <x degrees>"  
"cw 90" //Beispiel: Rechtsdrehung um 90°  
//<Direction> = cw, ccw
```

5. Steuerknüppel Kommando

```
"rc <roll> <pitch> <height> <yaw>"  
"rc 20 100 0 25 " //Die Tello fliegt eine Kurve bei 100%  
Vorwärtsgeschwindigkeit mit 20% seitlicher Geschwindigkeit und 25% Dreh-  
geschwindigkeit um die Hochachse auf gleichbleibender Höhe.
```

6. Positionier Kommando

```
"go <X> <Y> <Z> <Speed> <MP>"  
"go 0 0 50 100 m-2" //Beispiel: Auf 50cm Höhe mittig Ausrichten
```

7. Missionpadwechsel Kommando

```
"jump <X> <Y> <Z> <Speed> <Heading> <currentMP> <targetMP>"  
"jump 100 0 80 100 0 m1 m2";  
//Beispiel M2 liegt ca. einen Meter vor M1
```

8. Missionpaderkennungs Kommando

```
"<mon / moff>"  
"mon";  
//MP-Erkennung muss einmalig nach dem Abhebe Kommando aktiviert werden
```

9. Missionpadskonfigurations Kommando

```
"mdirection <0 = untere Kamera / 1 = Frontkamera / 2 = beide>"  
"mdirection 2";  
//Beispiel M2 liegt ca. einen Meter vor M1
```

Weitere Kommandos → *Tello_SDK_3.0_User_Guide_en.pdf*

11.3 Zustandsautomat

Als Unterstützung wird Ihnen noch folgendes Zustandsdiagramm zur Verfügung gestellt:

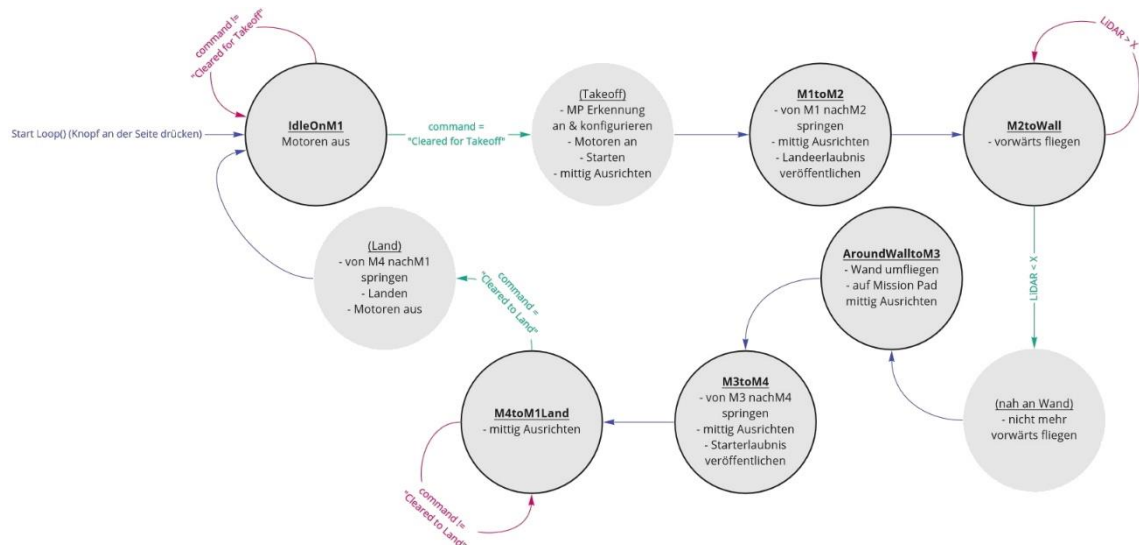


Abbildung 32: Zustandsdiagramm des Automaten

Um Ihnen das Scrollen zu ersparen, finden Sie hier nochmal ein Bild des Versuchsaufbaus.

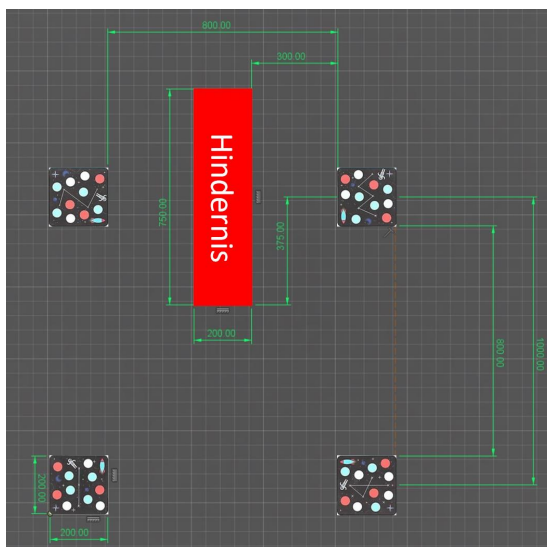


Abbildung 33: bemaßter Versuchsaufbau [mm]

„Tello1 wird auf das MP1 gestellt, bekommt über das *Message Queuing Telemetry Transport (MQTT)* Protokoll manuell den Startbefehl und fliegt anschließend um den Kurs. Sie erkennt das Hindernis mit dem *LiDAR* Sensor und umfliegt es automatisch. Währenddessen wird Tello2 auf das MP1 gestellt und angeschaltet. Kommt die Tello1 zum MP4, wird Sie darüber schwebend stehen bleiben und mittels *MQTT* der Tello2 den Startbefehl geben. Tello2 startet dann und gibt Tello1, sobald Sie weg ist, die Landeerlaubnis. Tello1 landet und die beiden Rollen sind getauscht. Theoretisch sollte der Ablauf dann unbegrenzt wiederholbar sein und die beiden Tellos den Parcours abfliegen, das Hindernis umfliegen und miteinander kommunizieren.“

Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 1: Tello Talent | 6 |
| Abbildung 2: Tello EDU - Merkmale | 7 |
| Abbildung 3: ESP32 „Rucksack“ bzw. DJI RoboMaster Tello Talent Expansion Kit | 7 |
| Abbildung 4: LED-Matrix mit darüber sitzendem LiDAR Sensor | 8 |
| Abbildung 5: Elemente des Mission Pads | 9 |
| Abbildung 6: kodierte Mission Pads 1-4 | 9 |
| Abbildung 7: Koordinatensystem des Mission Pads | 9 |
| Abbildung 8: bemaßter Versuchsaufbau [mm] | 11 |
| Abbildung 9: Szenario und Missionsablauf | 12 |
| Abbildung 10: Logo von Visual Studio Code | 14 |
| Abbildung 11: Logos der PlatformIO und C/C++ Extensions | 14 |
| Abbildung 12: MQTT Explorer Logo | 14 |
| Abbildung 13: IntelliSense: Hover Information | 15 |
| Abbildung 14: IntelliSense Autovervollständigung | 15 |
| Abbildung 15: IntelliSense Go to Definition | 16 |
| Abbildung 16: IntelliSense Definition Pop-up | 16 |
| Abbildung 17: Serial Monitor Knopf | 17 |
| Abbildung 18: PlatformIO Project Wizard | 18 |
| Abbildung 19: erste erfolgreiche Kompilierung | 18 |
| Abbildung 20: PlatformIO: Open Project | 19 |
| Abbildung 21: Ordnerstruktur des Projekts in PlatformIO | 20 |
| Abbildung 22: main.cpp Startzustand der Sektion der globalen Variablen | 23 |
| Abbildung 23: main.cpp Startzustand der Funktion setup() | 24 |
| Abbildung 24: main.cpp Startzustand der Funktion Callback() | 26 |
| Abbildung 25: main.cpp Startzustand der Funktion loop() | 29 |
| Abbildung 26: USB-Kabel an ESP32 anschließen und An-Knopf | 30 |
| Abbildung 27: Tello EDU im SDK Modus | 30 |
| Abbildung 28: Programmstart mit Knopfdruck bestätigt | 31 |
| Abbildung 29: MQTT-Explorer nachdem das Programm gestartet wurde | 31 |
| Abbildung 30: manuelle Veröffentlichung von "Hello World" in /command | 32 |
| Abbildung 31: Auszug SDK: "rc" Kommando | 33 |
| Abbildung 32: Zustandsdiagramm des Automaten | 36 |
| Abbildung 33: bemaßter Versuchsaufbau [mm] | 36 |

Quellen:

Abbildung 1: www.dji.com

Abbildung 2: www.store.dji.com

Abbildung 3: www.conrad.de

Abbildung 4: www.conrad.de

Abbildung 5: www.camforpro.com

Abbildung 6: www.dji.com Tello_SDK_3.0_User_Guide_en.pdf

Abbildung 7: www.dji.com Tello_SDK_3.0_User_Guide_en.pdf

Abbildung 8: www.camforpro.com

Abbildung 9: www.camforpro.com

Abbildung 10: code.visualstudio.com

Abbildung 11: Screenshot: Visual Studio Code Extensions

Abbildung 12: mqtt-explorer.com

Abbildung 13: Screenshot: Visual Studio Code

Abbildung 14: Screenshot: Visual Studio Code

Abbildung 15: Screenshot: Visual Studio Code

Abbildung 16: Screenshot: Visual Studio Code

Abbildung 17: Screenshot: Visual Studio Code / PlatformIO

Abbildung 18: Screenshot: Visual Studio Code / PlatformIO

Abbildung 19: Screenshot: Visual Studio Code / PlatformIO

Abbildung 20: Screenshot: Visual Studio Code / PlatformIO

Abbildung 21: Screenshot: Visual Studio Code / PlatformIO

Abbildung 22: Screenshot: Visual Studio Code / PlatformIO

Abbildung 24: Screenshot: Visual Studio Code / PlatformIO

Abbildung 23: Screenshot: Visual Studio Code / PlatformIO

Abbildung 25: Screenshot: Visual Studio Code / PlatformIO

Abbildung 26: komplett selbsterstellt

Abbildung 27: komplett selbsterstellt

Abbildung 28: komplett selbsterstellt

Abbildung 29: Screenshot: MQTT-Explorer

Abbildung 30: Screenshot: MQTT-Explorer

Abbildung 31: www.dji.com Tello_SDK_3.0_User_Guide_en.pdf

Abbildung 32: komplett selbsterstellt

Abbildung 33: www.camforpro.com

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