

Modular Recon Drone

Dokumentation

Matthias Haselmaier, B.Sc. und Andreas M. Brunnet, B.Sc.

Table of Contents

Vorwort	1
Aufbau der Drohne	1
Main Control Unit	1
Motor Controller	4
Kameramodul	5
Bildschirm	6
Stromversorgung	7
Schaltplan und PCB	7
Entwicklung einer Steuerung	8
Referenzimplementierung für Android	8
Ausblick	10
Gehäuse	11
Verpolschutz	11
Raspberry PI zero W	11
Glossar	11
Quellen	12

Vorwort

Aufbau der Drohne

Main Control Unit

Die MCU ist die Hauptkontrolleinheit. Ihre Hauptaufgaben sind:

- * Hosten eines Access Points
- * Hosten eines UDP-Sockets
- * Hosten eines TCP-Servers
- * Hosten des TWI Master

Grundlage der MCU bildet ein *Espressif ESP32* (DevKit v.1, siehe [1](#)).

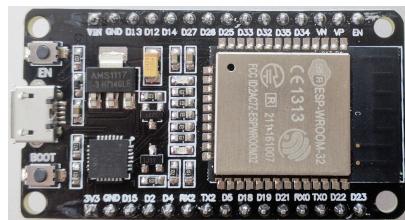


Abbildung 1. *Espressif ESP32 DevKit v1 Development Board*

Im Detail stellt der ESP32 einen Access Point mit WPA2 PSK Sicherung zur Verfügung. Das Passwort AP wird hierbei mittels des Hardware RNG in Form einer achtstelligen positiven Ganzzahl generiert. Das Passwort und die SSID des AP werden auf dem Bildschirm (siehe Abbildung [\[sec:oled_screen\]](#)) dargestellt.

Entwicklung

Der ESP32 ist mit 244 Mhz und zwei CPU Kernen ein relativ potenter 32 Bit Microcontroller. Zudem verfügt der Controller über eine Vielzahl an Schnittstellen. Neben den oben erwähnten Funktionalitäten, verfügt der ESP noch über:

- Bluetooth 4.2 (inkl. BLE)
- 18 ADC Kanäle mit 12 bit Auflösung
- 2 DAC mit 8 bit Auflösung
- 10 Touch Sensoren (Kapazitiv)
- 4 SPI Controller
- 2 TWI Controller
- 2 I²S Controller
- 3 UART Controller
- SD/SDIO/MMC/eMMC Host Controller
- CAN Bus 2.0
- Motor und LED PWM
- Hall Effekt Sensor
- Hardwarebeschleunigte Kryptografie (AES, SHA-2, RSA, ECC, RNG)

Software für den ESP32 kann mit Hilfe der Arduino IDE entwickelt werden. Jedoch ist das SDK des Herstellers, auch als ESP-IDF bezeichnet, vorzuziehen, da sie der Arduino IDE gegenüber wesentlich mächtiger ist.

ESP-IDF

Die Installation der ESP-IDF ist unter den gängigen Betriebssystemen mit Hilfe der Anleitung auf der Herstellerseite (siehe [\[esp_idf_install\]](#)) unkompliziert möglich. Dem SDK liegt zusätzlich noch eine Ansammlung an Beispielprojekten, sortiert nach Modulen (Wifi, Bluetooth, GPIO, etc) bei. Diese sind, wie auch die Komponenten der SDK selbst (Header Dateien) umfangreich und gut dokumentiert. Beim Aufsetzen eines neuen Projektes empfiehlt es sich, eines der Beispielprojekte zu nehmen (im Zweifelsfall das Hello World), da die ESP-IDF bereits eine auf Make basierende Buildchain anbietet.

Diese lässt sich mit folgenden Befehlen bedienen (aus dem Projektverzeichnis heraus):

- make menuconfig → TUI zur Konfiguration des Projektes (sdkconfig.h)
- make all → Bau des Projektes
- make flash → ggf. Bau des Projektes und Flashen des ESP
- make clean → Bereinigung des Build-Verzeichnisses
- make monitor → Starten des pythonbasierten seriellen Monitors

Um die Buildchain des SDK ohne Einschränkungen verwenden zu können, wird eine Python 2 installation benötigt. Diese sollte entsprechend unter Verwendung von *menuconfig* unter *SDK Tool Configuration* → *Python 2 Interpreter* gesetzt werden.

Danach muss noch der serielle Port, unter dem der ESP im Betriebssystem angebunden wird, konfiguriert werden. Dies geschieht unter *Serial flasher options* → *Default serial port*.

Die ESP-IDF nutzt das Echtzeitbetriebssystem FreeRTOS (siehe [\[freertos\]](#)). Dieses stellt Tasks, Message Queues zur Intertaskkommunikation, Tasksynchronisationsmechanismen (Mutex, etc) zur Verfügung.

Als TCP/IP Stack wird LwIP (siehe [\[lwip\]](#)) genutzt. Es handelt sich um eine leichtgewichtige Implementierung, deren Fokus auf Embedded Systems liegt. Angelehnt ist die API an die der POSIX Sockets.

MCU Software

Die Struktur des Projektes stellt sich wie folgt dar:

- main_control_unit
 - components
 - camera
 - u8g2
 - main

Für das Ansprechen der Komponenten Kamera (siehe Abschnitt [Kameramodul](#)) und Display (siehe Abschnitt [Bildschirm](#)) wurden jeweils entsprechende externe Bibliotheken verwandt (näheres in den entsprechenden Kapiteln). Diese sind, wie andere externe Bibliotheken im Allgemeinen, im Ordner *components* abgelegt. Der eigentliche Code der Drohne befindet sich im Ordner *main*. Der Aufbau der eigentlichen Projektdateien ist in Abbildung 2 definiert.

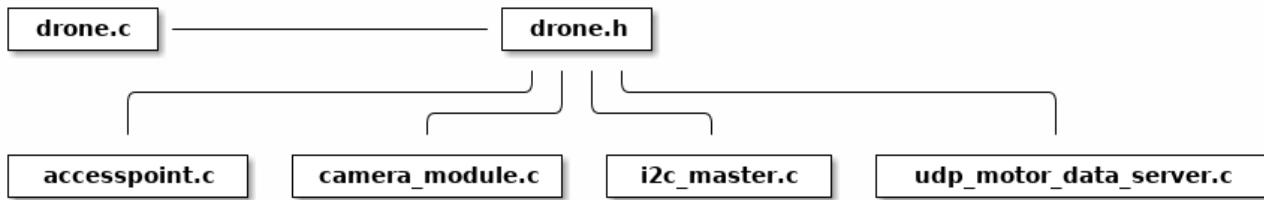


Abbildung 2. MCU Code Struktur

Zunächst werden die Komponenten initialisiert. Nach der Initialisierung werden die einzelnen Module (*accesspoint.c*, etc) gestartet. Hierbei werden entsprechend für die einzelnen Module jeweils Tasks generiert um möglichst parallel zu laufen. ===== Main Neben der Initialisierung der einzelnen Software-Module, ist auch der Systemeventhandler und die Bildschirmaktualisierung untergebracht. Der Eventhandler fängt die einzelnen Zustände des Accesspoints ab (Client verbündet sich, Client trennt sich) und setzt entsprechende Flags in der definierten Systemeventgroup. Mit Hilfe der Eventgroup kann nun entsprechender Text auf dem Display angezeigt werden, abhängig davon, ob ein Nutzer verbunden ist oder nicht.

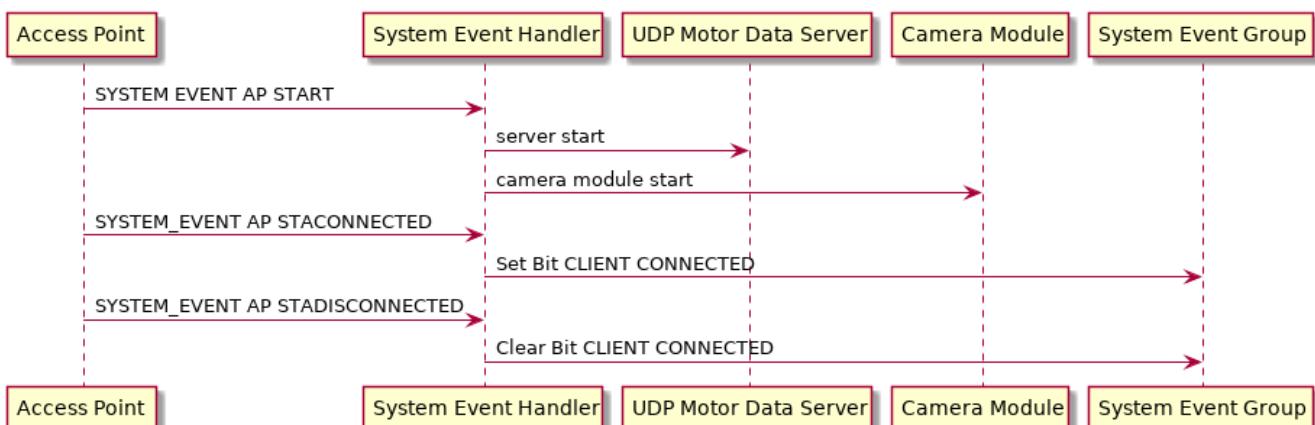


Abbildung 3. Ablauf, Start des AP, Client verbündet, Client trennt sich

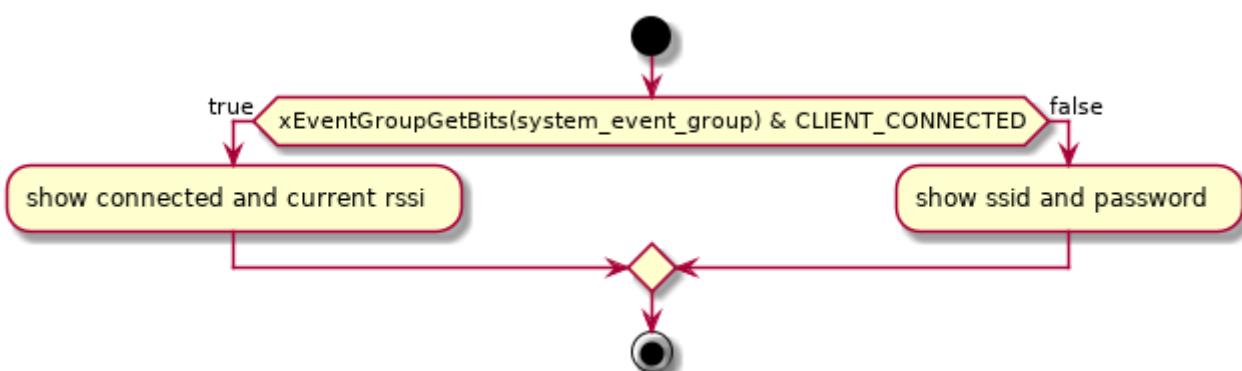


Abbildung 4. Display Textupdate

Access Point

Das AP Modul übernimmt neben der Initialisierung des Wifi auch die Initialisierung des TCP/IP Stacks. Während des Initialisierungsvorgangs wird mit Hilfe des RNG, wie bereits erwähnt, ein acht stelliges numerisches Passwort generiert. Die Konfiguration des Accesspoints lautet wie folgt:

Table 1. AP Konfiguration

SSID	Recon Drone
Channel	0
Authmode	WPA_WPA2_PSK
Hidden	false
Max Connections	1
Beacon Interval	100

Motor Control

Camera

Motor Controller

Um die beiden Motoren anzusteuern, wurde ein ATTiny25 Mikrocontroller pro Motor verwendet. Über das TWI wird den beiden Mikrocontrollern die gewünschte Richtung und Geschwindigkeit von der MCU mitgeteilt. Sie werden über die Adressen 0x01 und 0x02 angesprochen. Auf Abbildung 3 sind die Pins für die Takt- (SCL) und Datenleitung (SDA) des TWIs gezeigt. Da die ATTinys nicht über eine Hardwareimplementierung des TWIs verfügen, musste sie in Software realisiert werden. Hierzu wurde eine bereits existierende Implementation von [\[twi_implementation\]](#) verwendet, welche die USI Schnittstelle der Mikrocontroller passend für das TWI konfiguriert (siehe [\[attiny254585\]](#)).

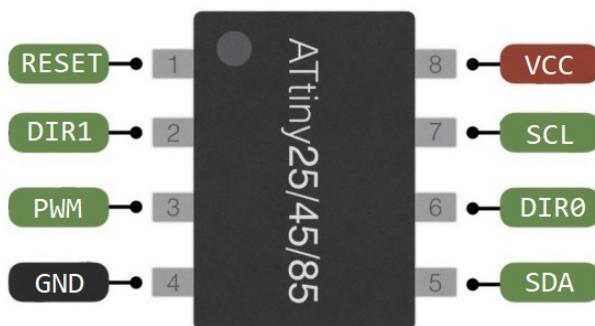


Abbildung 5. ATTiny25 Pinout

Die ATTinys erwarten ein Byte, in dem die Richtung und die Geschwindigkeit kodiert sind. Das höchstwertige Bit kodiert die Richtung. Wird eine 0 empfangen, wird der Pin *DIR0* auf *HIGH* gesetzt und der Pin *DIR1* auf *LOW* gesetzt. Wird eine 1 empfangen, wird entsprechend Pin *DIR0* auf *LOW* und Pin *DIR1* auf *HIGH* gesetzt. Diese beiden Pins sind nie zeitgleich *HIGH*. Durch die unteren sieben Bit des empfangenen Bytes wird die Geschwindigkeit in Prozent angegeben. Intern betreiben die Mikrocontroller hiermit eine Pulsweltenmodulation, welche 127 Schritte unterstützt. Dieses Signal wird über den Pin *PWM* ausgegeben. Zwischen den ATTinys und den Motoren wurde eine H-Brücke eingebaut, um die Motoren mit 11.1V betreiben zu können. Wie die beiden

Mikrocontroller über die H-Brücke mit den beiden Motoren verbunden ist, ist auf Abbildung 4 gezeigt.

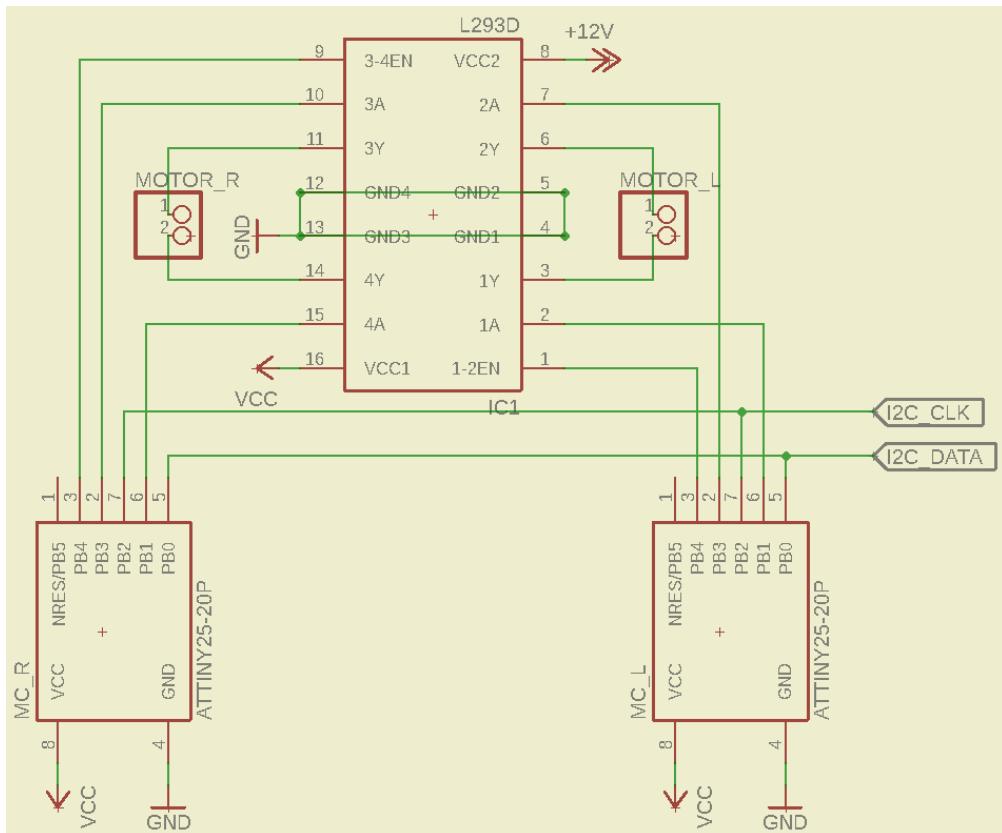


Abbildung 6. Schaltplan der Motorsteuerung

Der Programmcode für beide Mikrocontroller ist identisch. Die Adresse für das TWI wird über das Define-Flag *address* gesetzt werden. Um das Kompilieren des Programms für die beiden Mikrocontroller zu erleichtern, wurde ein Makefile erstellt. Durch den *make* Befehl werden die Programm der beiden Mikrocontroller gebaut. Durch *make flash_L* und *make flash_R* wird entsprechend das Programm für die linke und rechte Motorsteuerung auf die Mikrocontroller geschrieben. Damit das Kompilieren und Programmieren der ATTiny's funktioniert, müssen die AVR-Entwicklungstools *avr-gcc*, *avr-objcopy* und *avrdude* installiert sein. Bevor die Programme auf die Mikrocontroller geschrieben werden können, muss sichergestellt werden, dass der Port des Programmers richtig gesetzt ist. Hierzu kann das *-P*-Flag des *avrdude*-Befehls angepasst werden. Je nachdem, welche Programmer verwendet wird, muss auch das *-c*-Flag angepasst werden. In der aktuellen Version wird davon ausgegangen, dass mit einem AVRISP programmiert wird (siehe [arduino_programmer]).

Kameramodul

Das Kameramodul basiert auf dem OV7725 VGA Sensor von Omnipixel.

Table 2. Anschluss der Kamera am ESP32

Kamera Pin	ESP32 Pin		Kamera Pin	ESP32 Pin
SIOC	GPIO23		SIOD	GPIO25
XCLK	GPIO27		VSYNC	GPIO22
HREF	GPIO26		PCLK	GPIO21

Kamera Pin	ESP32 Pin		Kamera Pin	ESP32 Pin
D2	GPIO35		D3	GPIO17
D4	GPIO34		D5	GPIO5
D6	GPIO39		D7	GPIO18
D8	GPIO36		D9	GPIO19
RESET	GPIO15		PWDN	(über 10kOhm Widerstand auf GND)
3.3V	3.3V		GND	GND

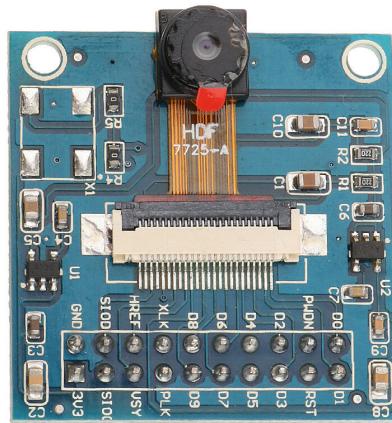


Abbildung 7. Kameramodul OV7725

Bildschirm

Verwendung findet ein monochromes OLED-Display mit einer Auflösung von 128x64 Pixel. Dieses basiert auf dem verbreiteten SSD1306 Controller.



Abbildung 8. Oled Diplay 128x64, SSD1306

Das Display wird hierbei über einen TWI-Bus angebunden. Entsprechend Pinout setzt sich wie folgt zusammen:

- GND: Ground, Masse
- VCC: Spannung, 3,3V - 5V

- SDL: TWI, Datenleitung
- SCL: TWI, Takteleitung

Stromversorgung

Im gesamten Projekt werden drei Versorgungsspannungen benötigt: 3.3V für das Kameramodul, 5V für die MCU und die Motorsteuerung und bis zu 12V für die Motoren. Auf Abbildung 7 wird gezeigt, wie die einzelnen Versorgungsspannungen von der Batterie erhalten werden.

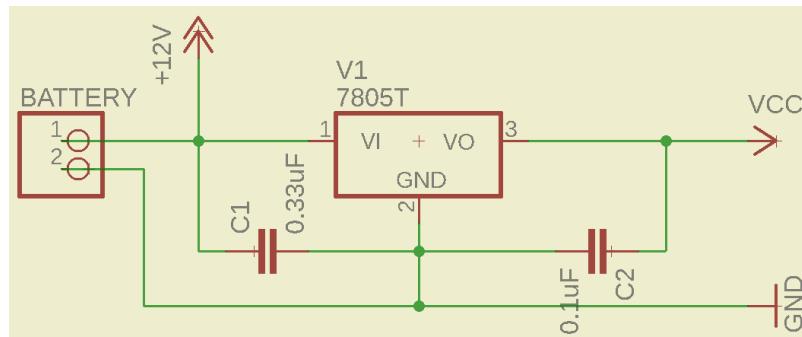


Abbildung 9. Stromversorgung

Die Motoren werden direkt mit der Batteriespannung betrieben. Um die 5V für die MCU und die Motorsteuerung zu erhalten, wurde der Spannungsregler 7805 verwendet. Er benötigt mindest 7V als Eingangsspannung, um 5V als Ausgangsspannung zu erzeugen. Um ein Schwanken der Betriebsspannungen zu vermeiden wurden Kondensatoren vor dem Ein- und Ausgang des Spannungsreglers platziert. Die für das Kameramodul nötigen 3.3V werden von der MCU erzeugt. Bei der Realisierung dieses Projektes wurde sich für einen 3-Zellen LiPo-Akku entschieden, wodurch 11.1V als Batteriespannung anliegen.

Schaltplan und PCB

Zu Beginn des Projekts wurden die einzelnen Komponenten auf einem Breadboard verkabelt und getestet und anschließend auf einer Lochrasterplatine in einem Prototypen verlötet. Dieser Prototyp ist auf Abbildung 8 zu sehen.

[lochrasterplatine] | *lochrasterplatine.png*

Abbildung 10. Lochrasterplatine

Nachdem die Funktionalität des Aufbaus der Hardware verifiziert wurde, wurde ein PCB mit der Eagle Software von Autodesk designed (siehe [\[eagle\]](#)). Die Eagle-Projektdateien sind im Ordner *schematic_and_pcb* zu finden. Anschließend wurde JLCPCB (siehe [\[jlpcb\]](#)) mit der Fertigung des PCBs beauftragt. Das fertig bestückte PCB ist auf Abbildung 9 gezeigt.

[pcb] | *pcb.png*

Abbildung 11. Fertige bestücktes PCB

Entwicklung einer Steuerung

Aufgrund der schlichten Schnittstelle zur Drohne ist es sehr einfach, eine beliebige Steuerung zu implementieren. Um die Drohne anzusteuern, muss sich zuerst mit dem bereitgestellt W-Lan-Netzwerk verbunden werden. Innerhalb dieses Netzwerkes hat die Drohne immer die IP-Adresse 192.168.4.1. Um das Kamerasignal zu erhalten, muss eine TCP-Verbindung zur Drohne über den Port 1234 aufgebaut werden. Die Drohne sendet dann das Bildsignal Byteweise an den Client. Das übersendete Bild ist im Grayscale-Format und besitzt eine Auflösung von 160x120 Pixeln. Um die Drohne zu steuern, müssen ihr zwei Bytes per UDP übertragen werden. Das erste Byte steuert den linken Motor und das zweite Byte steuert den rechten Motor (vgl. Abbildung [\[fig:upd-packet\]](#)).

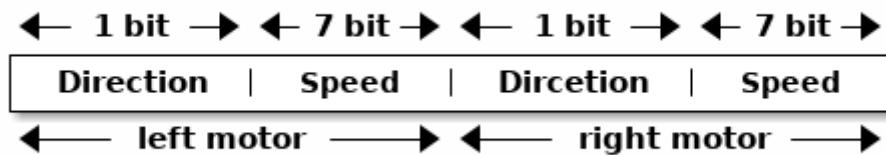


Abbildung 12. UDP-Paketformat

Das **Direction** bit gibt an, ob die Motoren vorwärts (0) oder rückwärst (1) drehen. In den sieben **Speed** bits wird die prozentuale Geschwindigkeit definiert, also ein Wert im Intervall [0, 100]. Um die Drohne um Hindernisse zu manövrieren, muss die Geschwindigkeit der beiden Motoren unterschiedlich hoch definiert werden.

Im Folgenden wird eine Referenzimplementierung der Steuerung der Drohne am Beispiel einer Android App beschrieben.

Referenzimplementierung für Android

Zur Steuerung der Recon Drohne wurde eine simple Android App entwickelt. Sie besteht im Grunde aus zwei Ansichten, eine zum Verbinden mit der Drohne und eine zum Steuern der Drohne. Beide Ansichten sind auf Abbildung 10 gezeigt. Die App sucht nach dem W-Lan Netzwerk, dass von der Drohne bereitgestellt wird. Damit W-Lan Netzwerke gesucht werden können, muss der Standort-Service des Smartphones aktiviert sein. Wurde das Netzwerk gefunden, wird das Passwort zum Verbinden verlagt. Wurde das Passwort eingegeben und der "Verbinden"-Button gedrückt, baut die App die Verbindung zur Drohne auf. Damit die App mit der Drohne kommunizieren kann, müssen die mobile Internet deaktiviert sein. Diese Funktionalität wird durch die *ReconDroneConnection*-Klasse realisiert (vgl. Abbildung 11). Hat sich das Smartphone mit der Drohne verbunden, wird automatisch zur nächsten Ansicht der App gewechselt, welche durch die *ReconDrone*-Klasse realisiert wird.

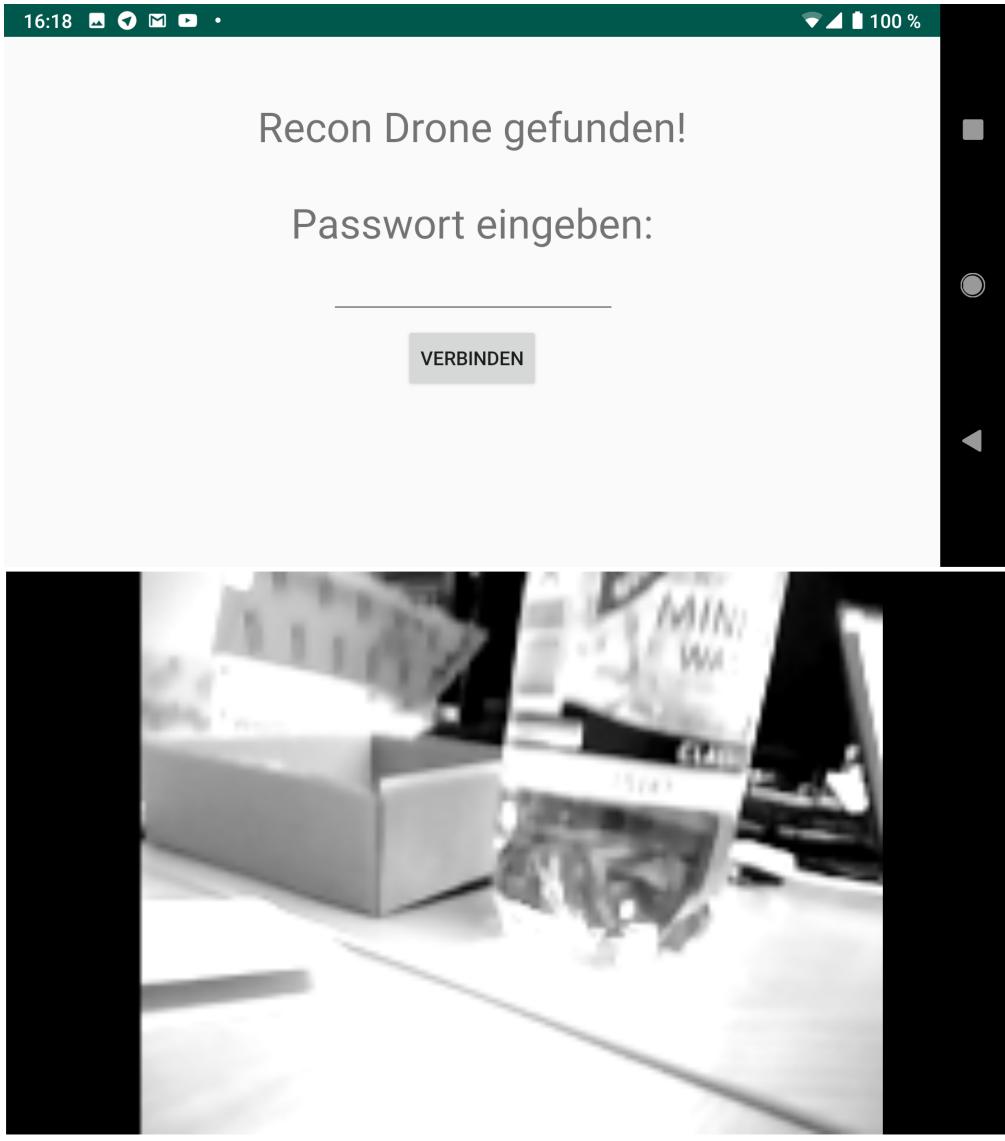


Abbildung 13. Ansichten der App

Aus dieser zweiten Ansicht kann die Drohne gesteuert werden und das Kamerasignal eingesehen werden. Die Steuerung wird von der *TouchController*-Klasse implementiert. Die Touchevents des Benutzer werden in der *onTouch()*-Methode verarbeitet. In ihr werden die Startposition und aktuelle Position der Touchbewegungen verarbeitet und in den Attributen *startingPositions* und *currentPositions* zwischengespeichert. Die komplette rechte Hälfte des Bildschirms dient zur Beschleunigung der Drohne. Je weiter der Finger vom Startpunkt der Touchbewegung entfernt ist, desto schneller bewegt sich die Drohne. Die linke Hälfte des Bildschirms dient zur Lenkung der Drohne. In den Attributen *acceleration* und *steering* wird die ID des jeweiligen Touchevents gespeichert. Wenn kein Touchevent vorliegt, enthalten die beiden Attribute den Wert -1. Alle 50 Millisekunden wird die Methode *sendControllerUpdate()* aufgerufen, welche der Drohne die Steuersignale sendet. Es werden zwei Bytes übertragen, die jeweils die Geschwindigkeit und Richtung für einen Motor definieren. Die beiden Bytes werden in den Methoden *setAcceleration()*, *setSteering()*, *setDirection()* und *adjustForInPlaceTurn()* befühlt.

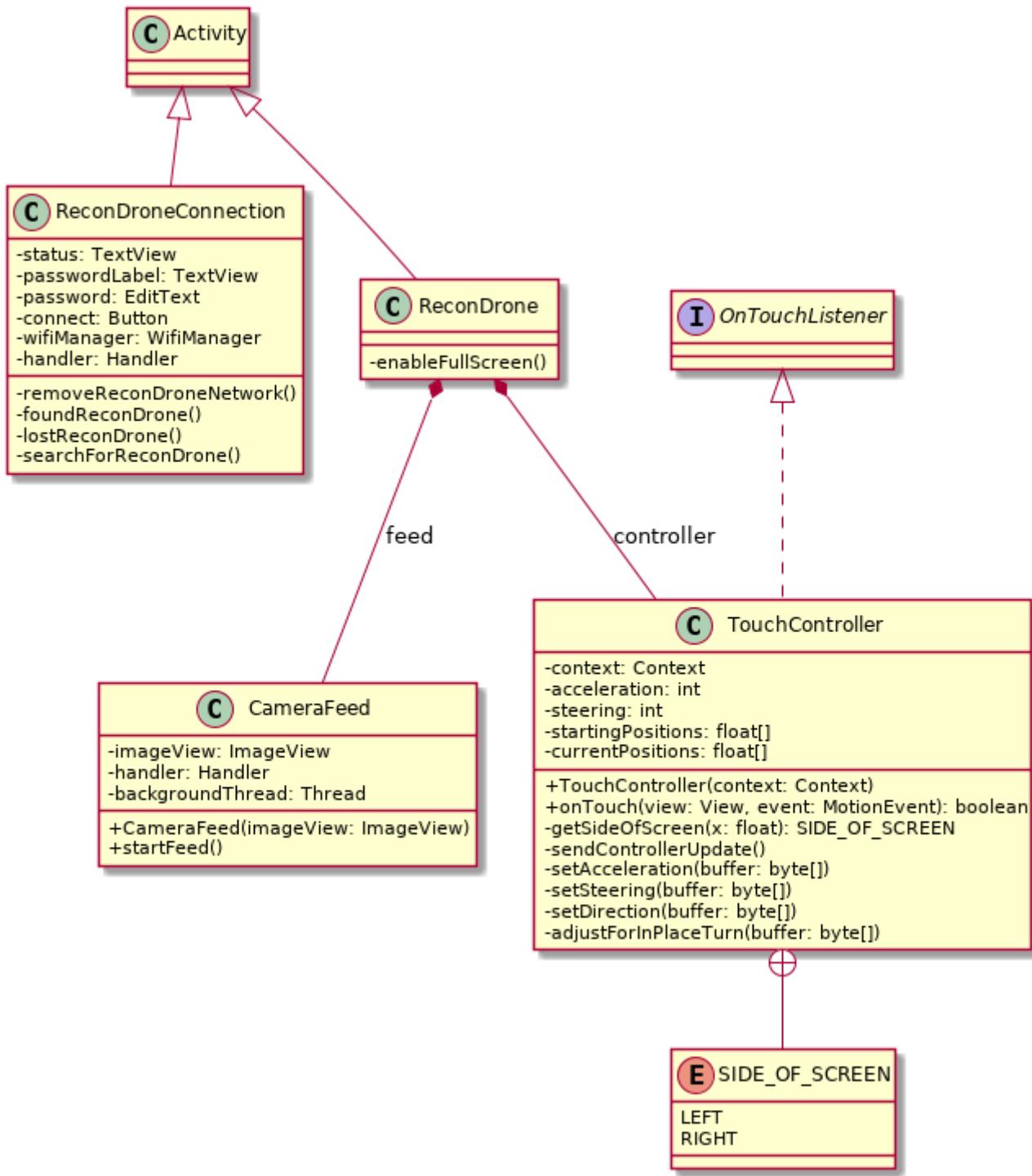


Abbildung 14. Klassendiagramm

Das Kamerasignal der Drohne wird in der *CameraFeed*-Klasse verarbeitet. Sie nimmt in ihrem Konstruktor das *ImageView*-Objekt entgegen, in dem sie das Kamerasignal anzeigen soll. Sie startet einen Hintergrundthread, in dem die einzelnen Bilder empfangen werden und wandelt die Byte-Daten in ein Bitmap um. Da in Android das User-Interface nicht aus einem Hintergrundthread angepasst werden kann, wird ein *Handler* verwendet, der das Bitmap im *ImageView* anzeigt.

Ausblick

Auf Basis des derzeitigen Entwicklungsstandes der Drohne, lassen sich weitere Zusatzfunktionen und Verbesserungen implementieren. Einige Ideen sollen im Folgenden präsentiert werden.

Gehäuse

Derzeit ist das Chassis der Drohne noch im Prototypenzustand, gefertigt aus Stahllochblechprofilen. Diese Bauart ist zum einen relativ schwer und bietet keinen Rundumschutz. Es bietet sich deshalb an, ein Gehäuse mittels 3D-Druck anzufertigen. Während des Projektes wurde bereits eine erste Version des Gehäuses mittels CAD Software (siehe [openscad](#)) modelliert. Dieses kann evaluiert und gedruckt werden oder durch ein anderes völlig neues Design ersetzt werden.

Verpolschutz

Im aktuellen Stand wird der LiPo Akku mittels *Male Headern* am PCB der Drohne angeschlossen. Der Akku selbst verfügt über eine Buchse vom Typ *JST RCY*. Zum einen könnte ein entsprechender *Female* Verbinder für THT Platinen verbaut werden um eine richtige Polung beim Anschließen zu garantieren. Des weiteren kann mit Hilfe eines Brückengleichrichters (siehe Abbildung 12) hinter dem Akkuanschluss ein Schutz vor Verpolung garantiert werden.

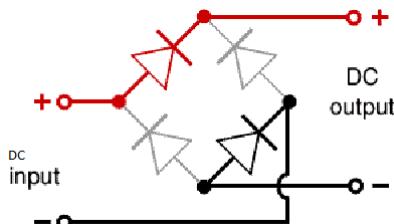


Abbildung 15. Diodenbrückengleichrichter

Die Funktionsweise der Brücke ist relativ simple. Unabhängig davon, wie der Strom angelegt wird, schließen die Konfiguration der Dioden stets so, dass am Ausgang der Brücke Plus und Minus immer gleich anliegen. Eine Diode hat entsprechend die Eigenschaft, Strom in eine Richtung hin passieren zu lassen, während sie in entgegengesetzter Richtung den Stromfluss sperrt.

Raspberry PI zero W

Alternativ kann als MCU der Raspberry PI zero W evaluiert werden. Vorteil dieses Einplatinencomputers im Gegensatz zum ESP32 ist der wesentlich größere RAM, der das Puffern größerer Kamerabilddaten ermöglicht (siehe Abschnitt [\[sec::camera_module\]](#)).

Glossar

TWI

Two Wire Interface (bekannt als I²C), zweiadriger Master-Slave Bus

MCII

Main Control Unit

PCB

Printed Circuit Board

USI

Universal Serial Interface

BLE

Bluetooth Low Energy

ADC

Analog Digital Converter

DAC

Digital Analog Converter

PWM

Pulse Width Modulation

AP

Access Point

RNG

Random Number Generator

SDK

Software Development Kit

THT

Through Hole Technology, Durchsteckmontage

Quellen

- [u8g2] Graustufenbildschirm Bibliothek: <https://github.com/olikraus/u8g2>
- [u8g2_esp] esp idf wrapper: <https://github.com/nkolban/esp32-snippets/tree/master/hardware/displays/U8G2>
- [esp_cam] esp idf Kamera Bibliothek: <https://github.com/igrr/esp32-cam-demo.git>
- [arduino_programmer] Arduino Uno als Programmer: <https://create.arduino.cc/projecthub/arjun/programming-attiny85-with-arduino-uno-afb829>
- [eagle] <https://www.autodesk.co.uk/products/eagle/overview>
- [twi_implementation] <https://github.com/rambo/TinyWire>
- [attiny254585] https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf
- [esp_idf_install] ESP-IDF Getting Started: <https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html>
- [freertos] FreeRTOS, Dokumentation: https://www.freertos.org/Documentation/RTOS_book.html
- [lwip] LwIP TCP/IP Stack: <https://savannah.nongnu.org/projects/lwip/>

- [openscad] OpenSCAD: <http://www.openscad.org/>