

Modular Recon Drone

Dokumentation

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

Table of Contents

Vorwort	1
Architektur	1
Main Control Unit	1
Motor Controller	2
Kameramodul	4
Bildschirm	4
Stromversorgung	4
Schaltplan und PCB	5
Android App	5
Ausblick	7
Glossar	8
Quellen	8

Vorwort

Architektur

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 [{counter:fig}](#)).

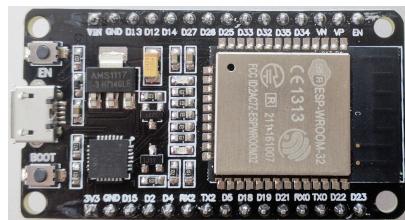


Abbildung 1. Espressif ESP32 DevKit v1 Development Board

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 [Bildschirm](#)) 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, konfiguiert werden. Dies geschieht unter *Serial flasher options* → *Default serial port*.

MCU Software

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 [{counter:fig}](#) 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 [\[attiny254585\]](#)).

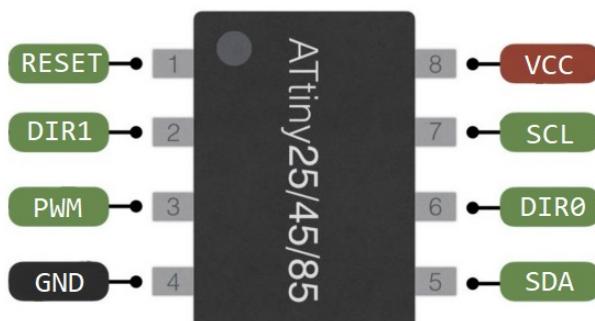


Abbildung 2. ATTiny25 Pinout

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 dein 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 [{counter:fig}](#) gezeigt.

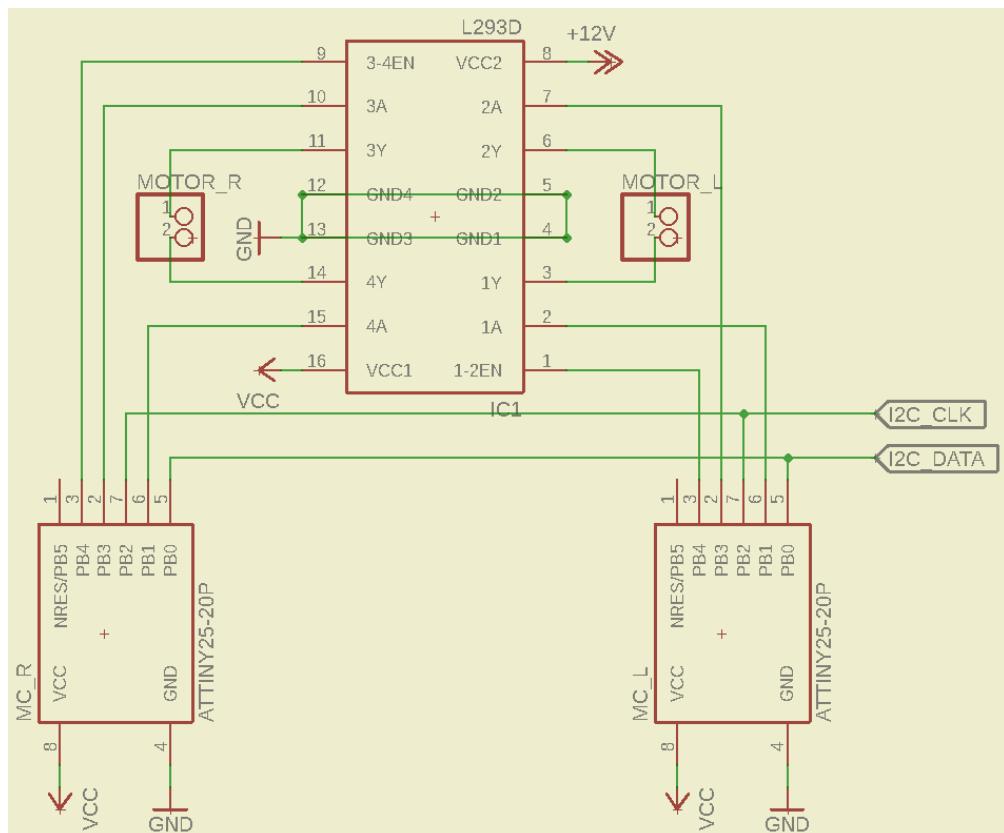


Abbildung 3. Schaltplan der Motorsteuerung

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 Programme 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 ATTinys 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 Omnivision.

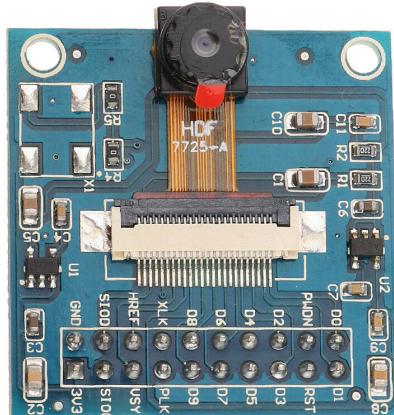


Abbildung 4. Kameramodul OV7725

Bildschirm

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



Abbildung 5. 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
- SDA: TWI, Datenleitung
- SCL: TWI, Taktleitung

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 {counter:fig} wird gezeigt, wie die einzelnen Versorgungsspannungen von der Batterie erhalten werden.

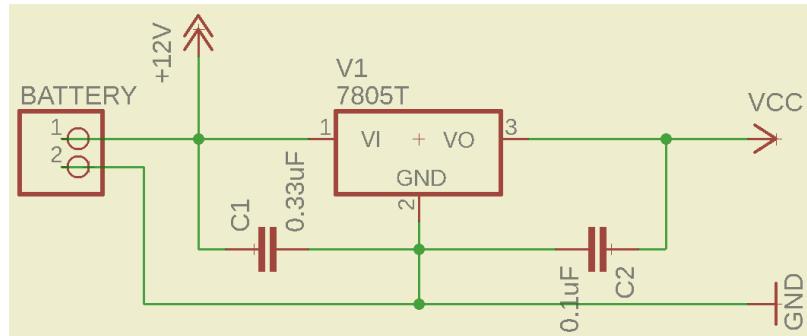


Abbildung 6. Stromversorgung

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 Ausgungsspannung 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 [{counter:fig}](#) zu sehen.

[lochrasterplatine] | *lochrasterplatine.png*

Abbildung 7. 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 [\[jlcpcb\]](#)) mit der Fertigung des PCBs beauftragt. Das fertig bestückte PCB ist auf Abbildung [{counter:fig}](#) gezeigt.

[pcb] | *pcb.png*

Abbildung 8. Fertige bestücktes PCB

Android App

Zur Steuerung der Recon Drone wurde eine simple Android App entwickelt. Sie besteht im Grunde aus zwei Ansichten, eine zum Verbinden mit der Drone und eine zum Steuern der Drone. Beide Ansichten sind auf Abbildung [{counter:fig}](#) gezeigt. Die App sucht nach dem W-Lan Netzwerk, dass von der Drone 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 Drone auf. Damit die App mit der Drone kommunizieren kann, müssen die mobile Internet deaktiviert sein. Diese Funktionalität wird durch die *ReconDroneConnection*-Klasse realisiert (vgl. Abbildung [{counter:fig}](#)). Hat sich das Smartphone mit der Drone verbunden, wird automatisch zur nächsten Ansicht der App gewechselt, welche durch

die *ReconDrone*-Klasse realisiert wird.

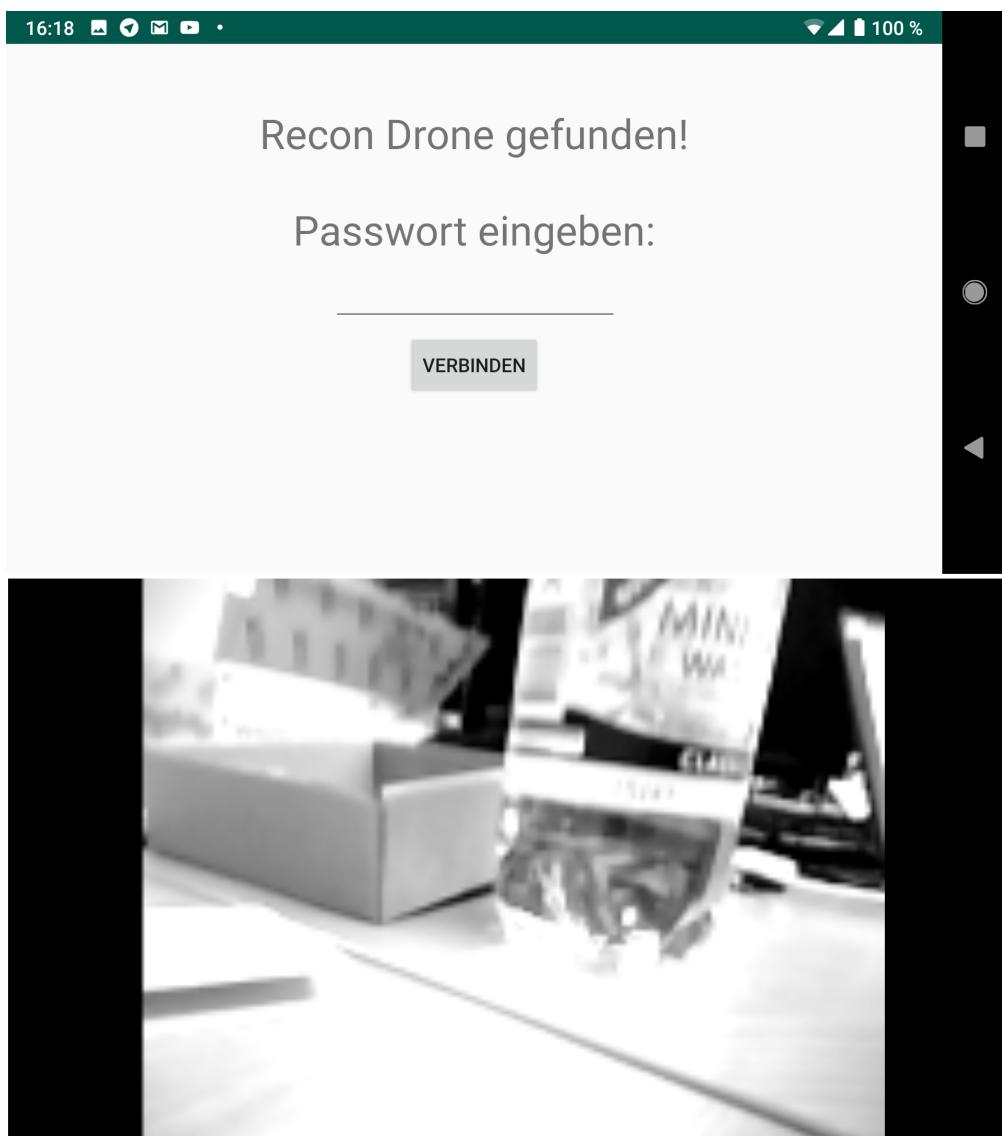


Abbildung 9. Ansichten der App

Aus dieser zweiten Ansicht kann die Drone 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 Drone. 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 Drone 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.

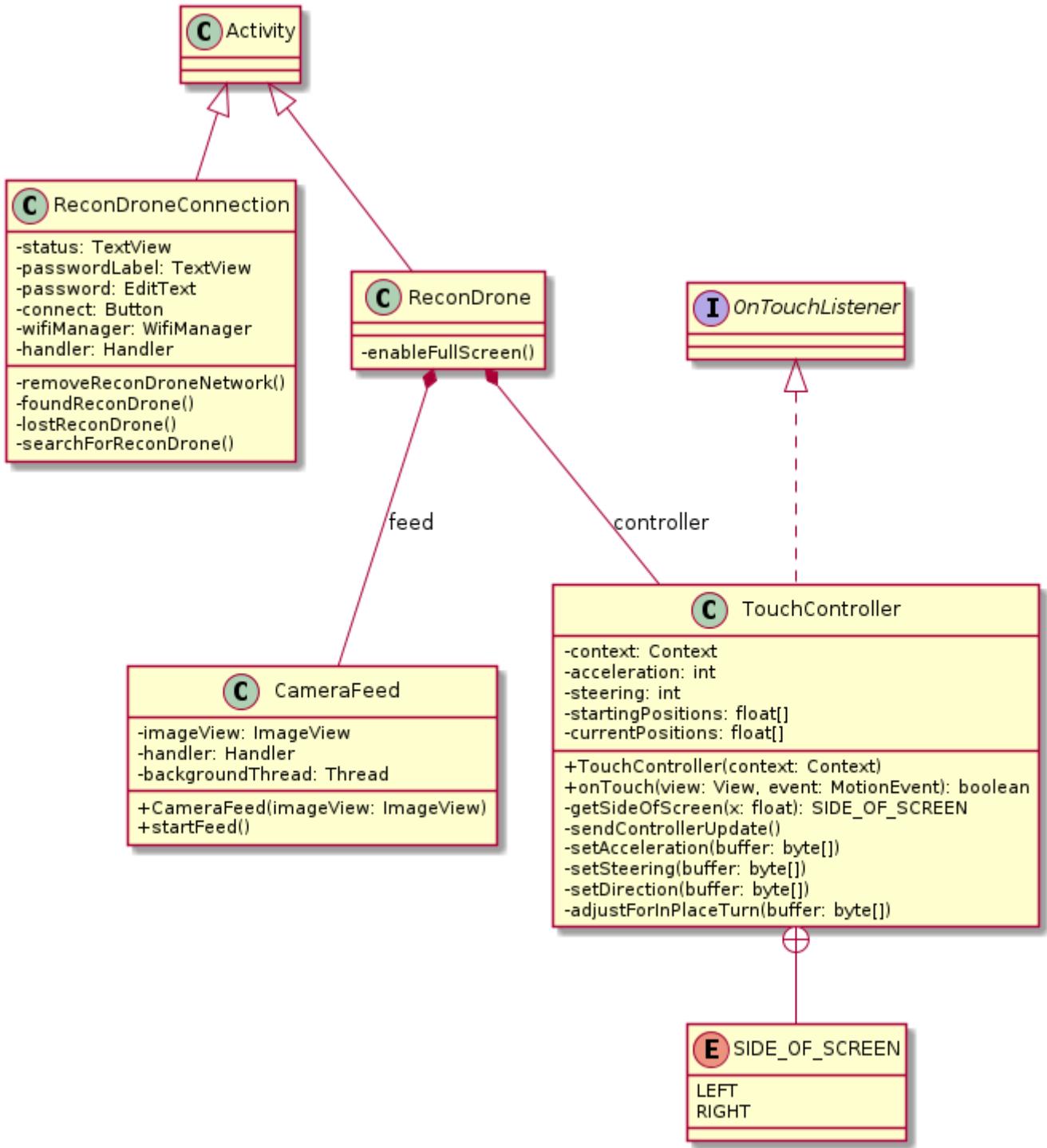


Abbildung 11. Klassendiagramm

Das Kamerasignal der Drone 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

Glossar

TWI

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

MCU

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

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>