

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	6
Kameramodul	7
Bildschirm	8
Stromversorgung	9
Schaltplan und PCB	10
Entwicklung einer Steuerung	11
Referenzimplementierung für Android	11
Ausblick	14
Gehäuse	14
Verpolschutz	14
Raspberry PI zero W	14
Alternative Stromversorgung	14
Externer SPI RAM	15
Glossar	15
Quellen	16
Anhang	17

Vorwort

Im Rahmen der Master-Projektarbeit im Wintersemester 2018/19 wurde eine Modular-Recon-Drone (modulare Aufklärungsdrohne) entwickelt. Auf Abbildung 1 ist der finale Aufbau abgebildet. Bei der Architektur der Hard- und Software wurde auf einen modularen Aufbau geachtet, um die einzelnen Komponenten leicht gegen Alternativen aus tauschen zu können. So wurde zum Beispiel das Ansteuern der Motoren in ein eigenes Modul ausgelagert, das über ein TWI-Interface angesprochen werden kann. Die Drohne bietet eine simple Schnittstelle, über die sie gesteuert werden kann. Auch hier kann das Steuersystem leicht ausgetauscht oder angepasst werden. Im Rahmen dieses Projekts wurden zwei Steuersysteme entwickelt, welche als Basis für die Entwicklung weitere Eingabemethoden genutzt werden können. Um die Drohne auch aus der Ferne zu steuern, verfügt sie über ein Kameramodul.

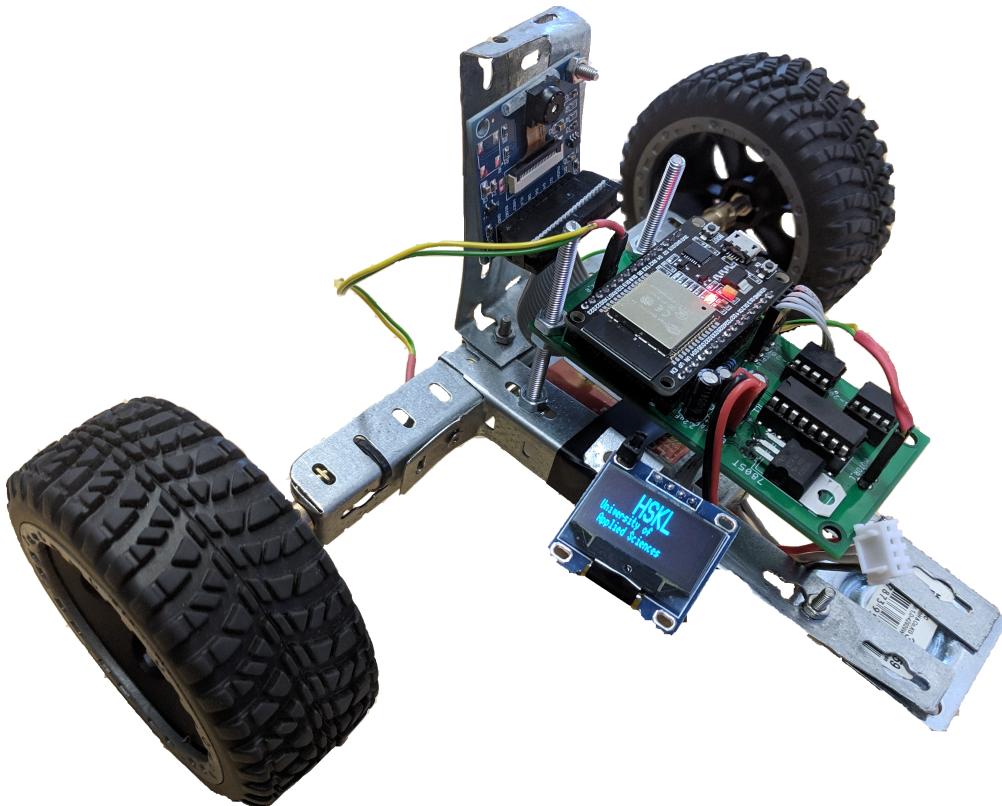


Abbildung 1. Modular-Recon-Drone

Im Folgenden werden die einzelnen Komponenten der Drohne und ihr Aufbau dokumentiert. Anschließend werden die Steuersysteme vorgestellt.

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 [2](#)).

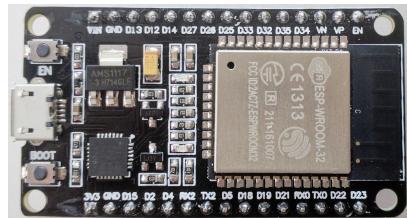


Abbildung 2. Espressif ESP32 DevKit v1 Development Board

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, konfiguiert 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 [3](#) definiert.

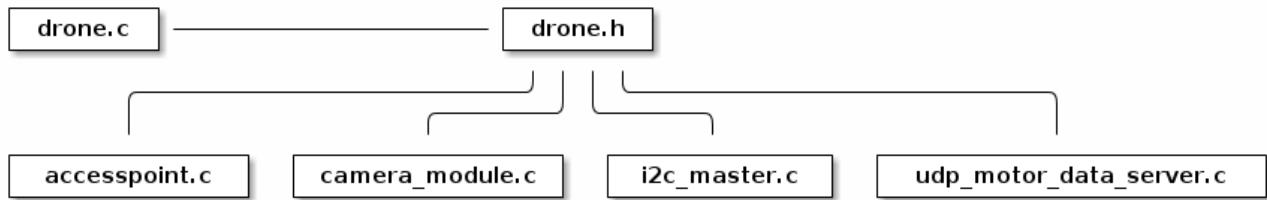


Abbildung 3. MCU Code Struktur

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 verbindet sich, Client trennt sich) und setzt entsprechende Flags in der definierten Systemeventgroup.

Exemplarisch ist die Ablaufreihenfolge der Accesspoint Events in Abbildung 4.

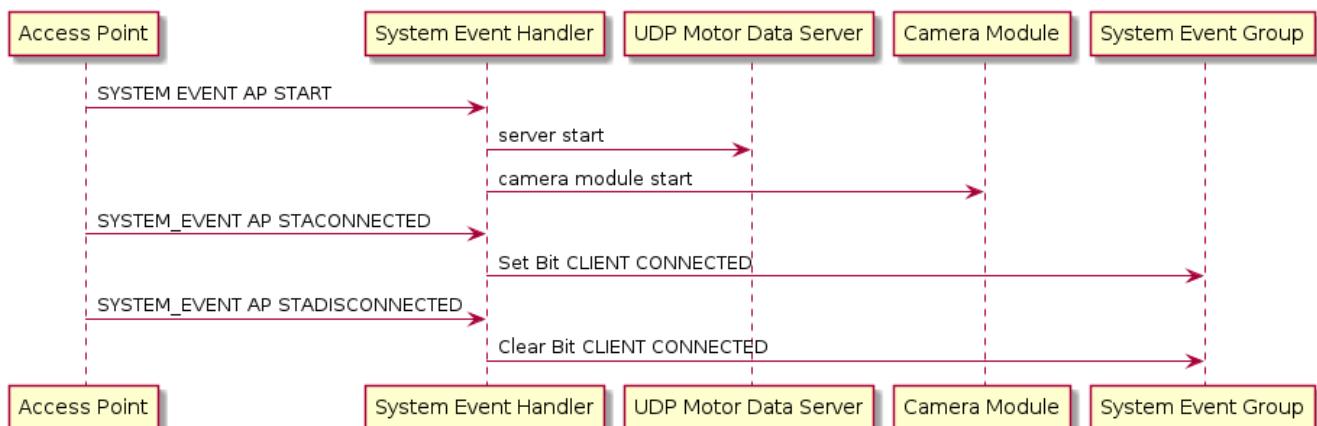


Abbildung 4. Ablauf, Start des AP, Client verbindet, Client trennt sich

Mit Hilfe der *Eventgroup* kann nun entsprechender Text auf dem Display angezeigt werden, abhängig davon, ob ein Nutzer verbunden ist oder nicht (siehe Abbildung 5).

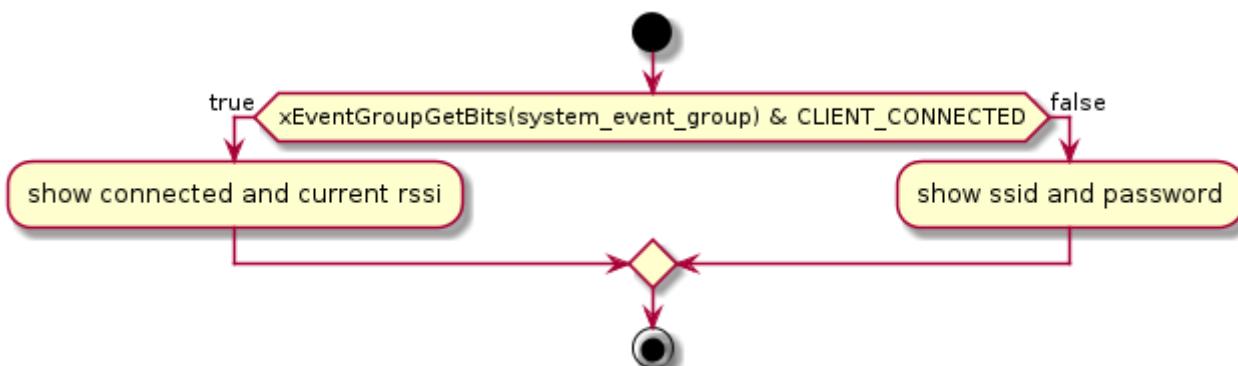


Abbildung 5. Display Textupdate

Access Point Module

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 ist in Tabelle [AP Konfiguration](#) aufgeführt.

Tabelle 1. AP Konfiguration

SSID	Recon Drone
Kanal	0
Authmode	WPA_WPA2_PSK
Versteckt	nein
Max. Verbindungen	1
Beacon Interval	100
IP	192.168.4.1

I²C Master Module

Zum einen initialisiert das Modul den Bus als Master und startet den Task zum Senden der Motorsteuerdaten an die jeweiligen ATTiny (siehe Abschnitt [Motor Controller](#)).

Tabelle 2. TWI Bus Konfiguration

Mode	Master
SDA Leitung	Pin 32
SCL Leitung	Pin 33
Taktfrequenz	100 kHz
TX Puffer	Nein
RX Puffer	Nein

Zusätzlich wird das Display initialisiert und aus dem *Standby* aufgeweckt.

Camera Module

Neben der Initialisierung der Kamera mit Hilfe der externen Bibliothek (siehe Abschnitt [Kameramodul](#)), wird im Modul noch ein Task gestartet, der mittels TCP-Server auf einkommende Verbindungen von Seiten der Steuereinheit wartet. Nach Verbindungsaufbau sendet der TCP-Server konstant die aus der Kamera ausgelesenen Bilddaten an den Client. Sobald über die *Eventgroup* signalisiert wurde, dass der Client die Verbindung unterbrochen hat, wartet der TCP-Server auf einen erneuten Verbindungsaufbau. Dies geschieht zum einen über die *Eventgroup* und zum anderen über das Timeout der Schreiboperation des Sockets.

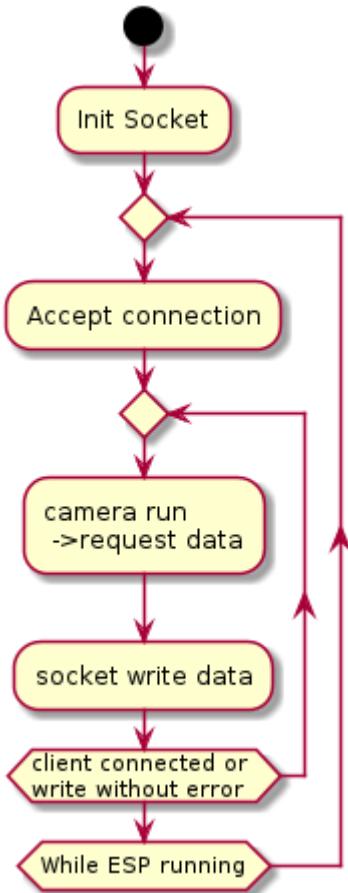


Abbildung 6. Kamera TCP-Server Ablauf

Tabelle 3. Kamera und TCP Socket Konfiguration

Pixel Format	Grayscale
Auflösung	160 x 120
TCP Port	1234

NOTE

Graustufen und die geringe Auflösung (in Graustufen jedoch auch 320 x 240 möglich) sind darauf zurückzuführen, dass der ESP32 nicht über ausreichend internen SRAM verfügt, um die Bilddaten vor dem Senden über TCP zwischenzupuffern (siehe Abschnitt [Externer SPI RAM](#)).

UDP Motor Data Server Module

Initialisiert und startet einen UDP Socket, der Daten

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 8 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 [\[attiny254585\]](#)).

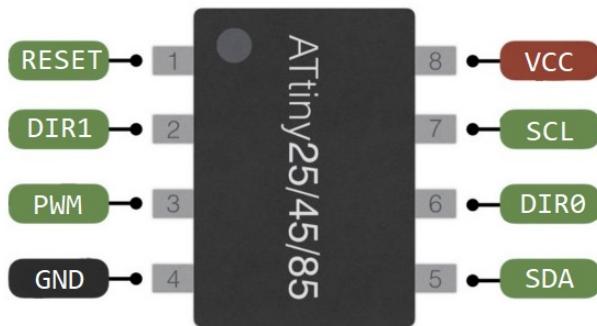


Abbildung 7. ATTiny25 Pinout

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 Pulsweitenmodulation, welche 127 Schritte unterstützt. Dieses Signal wird über den Pin *PWM* ausgegeben. Zwischen den ATTinys und dein Motoren wurde eine H-Brücken 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 18 im Anhang gezeigt. 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 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 verwendete Kameramodul basiert auf dem OV7725 VGA Sensor von Omnivision. Dieses Modul erfüllt die gängigen Marktansprüche an PC-Multimedia- und Smartphone-Kameras. Es kann in Temperaturen von -20°C bis 70°C betrieben werden und passt sich automatisch an schlechte Lichtverhältnisse an. Es unterstützt eine maximale Auflösung von 640x480 Pixel und unterstützt bis zu 60 FPS. Die Bilddaten können in verschiedenen Formaten über das SCCB Interface erhalten werden (vgl. [\[ov7725\]](#)).

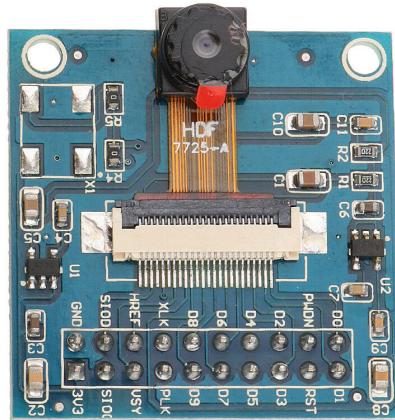


Abbildung 8. Kameramodul OV7725

Zum Ansprechen des Kameramoduls wurde eine bereits existierende Bibliothek verwendet (vgl. [\[esp_cam\]](#)). In der folgenden Tabelle ist beschrieben, wie das Kameramodul mit dem ESP32 verbunden wurde. Es ist zu bemerken, dass die Verbindung zwischen dem Kameramodul und dem ESP32 möglichst kurz zu halten ist. Wird die Verbindung zu lang, wird jediglich Rauschen vom ESP32 empfangen.

Tabelle 4. 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
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

Bildschirm

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



Abbildung 9. Oled Diplay 128x64, SSD1306

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

- GND: Ground, Masse
- VCC: Spannung, 3,3V - 5V
- SDA: TWI, Datenleitung
- SCL: TWI, Taktleitung

Zur Ansteuerung des Displays wird auf die **u8g2** Bibliothek für monochrome LCD- und OLED Bildschirme zurückgegriffen. Diese unterstützt eine vielzahl an Display Controllern. Die Standardimplementierung ist für die Verwendung unter Arduinos vorgesehen. Jedoch ist die Bibliothek mit hoher Abstraktion aufgebaut und bietet eine Schnittstelle um hardwarespezifische Funktionalitäten anderer Microcontroller wie dem ESP32 zu implementieren. Für den ESP32 existiert diese Implementierung bereits unter dem Namen *u8g2_esp* (siehe [\[u8g2_esp\]](#)). Jedoch wurde die Implementierung so vorgesehen, dass sie neben dem Ansteuern des Bildschirms auch die Initialisierung des TWI-Controllers übernimmt. Da dies im Projekt bereits geschieht, wurde der entsprechende Teil in der Implementierung entfernt. Auch wurde der verwendete TWI-Controller angepasst. Ursprünglich sah die Implementierung TWI-Controller 2 vor, die Drohne nutzt jedoch TWI-Controller 1.

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 18 im Anhang ist gezeigt, wie die einzelnen Versorgungsspannungen von der Batterie erhalten werden. 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 vom ESP32 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 11 zu sehen.

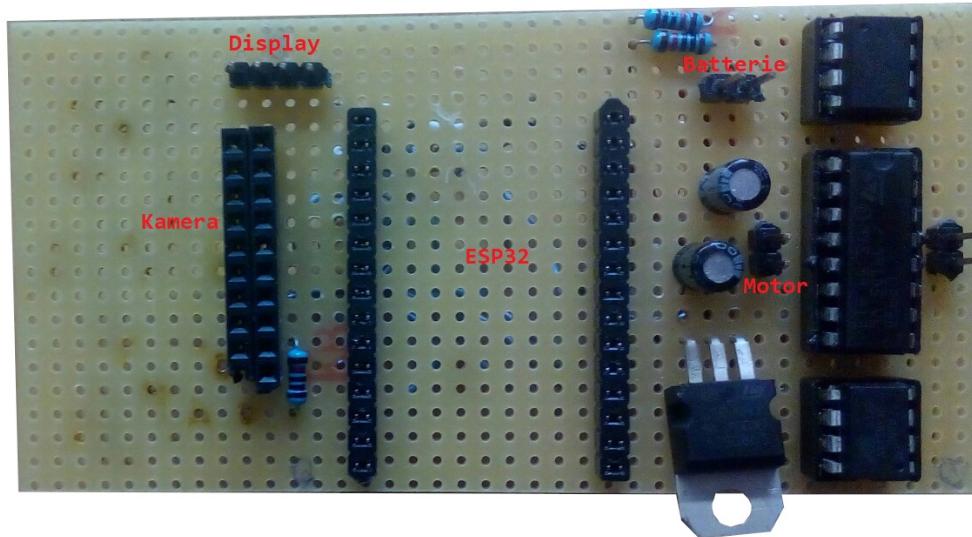


Abbildung 10. Lochrasterplatine

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 [\[jlcpcb\]](#)) mit der Fertigung des PCBs beauftragt. Das fertig bestückte PCB ist auf Abbildung 12 gezeigt.

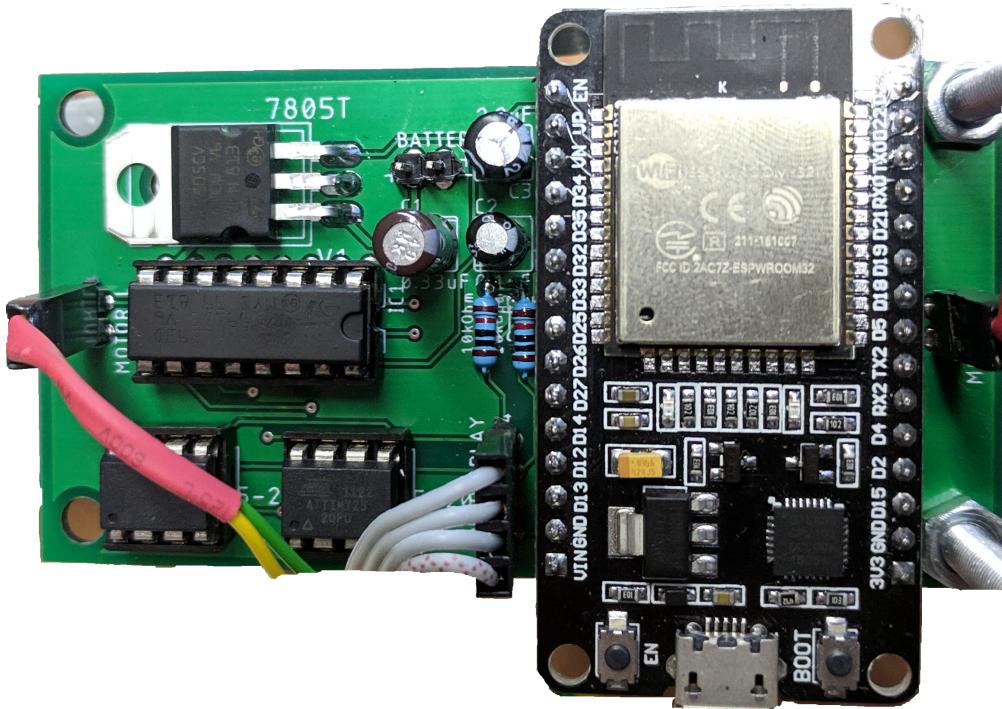


Abbildung 11. Fertige bestücktes PCB

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 bereitgestellten 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 [Abb. 1](#)).

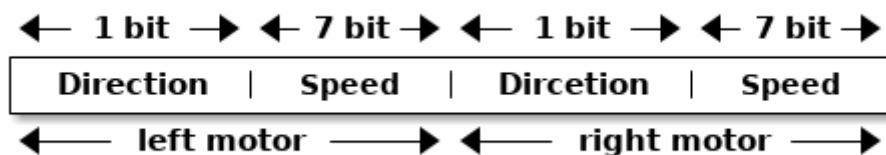


Abbildung 12. UDP-Paketformat

Das *Direction* Bit gibt an, ob die Motoren vorwärts (0) oder rückwärts (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 [13](#) 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 verlagert. 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 [14](#)). Hat sich das Smartphone mit der Drohne verbunden, wird automatisch zur nächsten Ansicht der App gewechselt, welche durch die *ReconDrone*-Klasse realisiert wird.

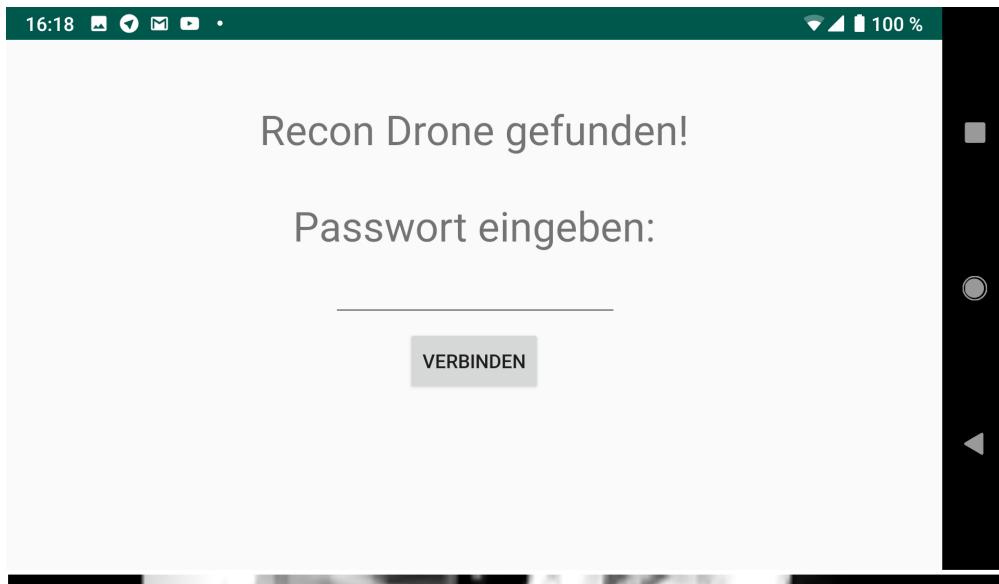


Abbildung 13. Ansichten der App

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.

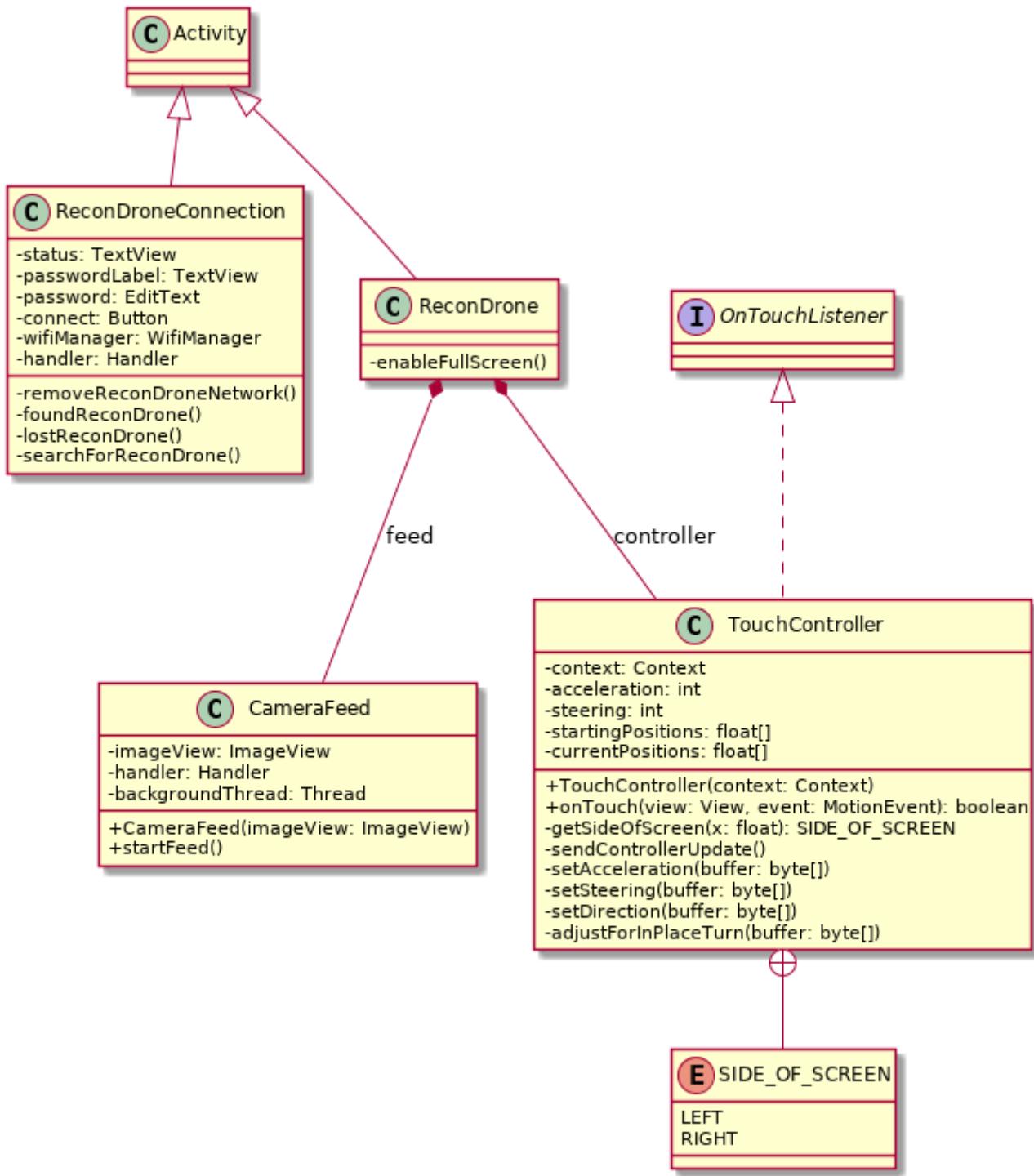


Abbildung 14. Klassendiagramm

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.

Um die Android App zu installieren, muss die APK-Datei auf das Smartphone übertragen werden. Sie befindet sich im Ordner *android_app/app/build/outputs/apk/debug*. Um sie zu installieren muss in den Einstellungen des Smartphones die Installation von Apps aus unsicheren Quellen erlaubt werden. Auf dem Smartphone muss mindestens Android 5.0 installiert sein. Die App wurde unter Android 9.0 getestet.

Ausblick

Auf Basis des derzeitigen Entwicklungsstands 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. Zum Anderen kann mit Hilfe eines Brückengleichrichters (siehe Abbildung 15) hinter dem Akkuanschluss ein Schutz vor Verpolung garantiert werden.

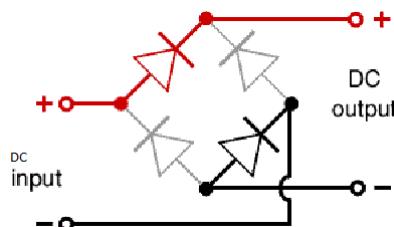


Abbildung 15. Diodenbrückengleichrichter

Die Funktionsweise der Brücke ist relativ simple. Unabhängig davon, wie der Strom angelegt wird, schließt 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 [\[sec::camera_module\]](#)).

Alternative Stromversorgung

Aktuell wird ein linearer Spannungsregler verwendet, um die hohe Batteriespannung von 11.1V auf die 5V zu reduzieren, die von der MCU und den ATTinys benötigt werden. Hierbei wird sehr viel Energie in Form von Wärme verschwendet. Dies beeinflusst die Batterielaufzeit der Drone negativ. Weiterhin besteht die Möglichkeit, dass der Spannungsregler überhitzt, in welchem Fall er sich abschalten würde. Erst wenn er weit genug abgekühlt ist, gibt er wieder 5V aus. Dies würde

zum Abschalten der Drone führen. Alternativ könnte ein *Step-Down-Converter* oder auch *Buck-Converter* verwendet werden. Im Vergleich zu einem linearen Spannungsregler wird so viel weniger Energie verschwendet und somit die Batterielaufzeit der Drone erhöht. Allerdings wird die Schaltung etwas komplexer und die benötigten Komponenten kosten etwas mehr als ein linearer Spannungsregler.

Externer SPI RAM

Das Kamerabild, das von der Drohne übertragen wird, kann aktuell eine maximale Auflösung von 320x240 Pixeln unterstützen, wenn es im Grayscale-Format übertragen wird. Diese Einschränkungen sind auf den begrenzten internen RAM des ESP32 zurückzuführen. Er besitzt eine insgesamte Größe von 320KB, welche sich in zwei Teile unterteilt. Aufgrund von technischen Einschränkungen können maximal 160KB statisch allokiert werden. Die restlichen 160KB können nur während der Laufzeit dynamisch allokiert werden (vgl. [\[esp32_kevin\]](#)). Bei einer Auflösung von 320x240 Pixeln und einem Byte für den Grauwert eines Pixels benötigt ein Bild bereits 76,8KB RAM, um es vor dem Senden zwischenzuspeichern. Ein farbiges Bild ist nicht möglich, da dies aufgrund der 3 Bytes pro Pixel, eines pro Farbkanal, bei einer Größe von 230,4KB liegen würde. Wird versucht ein Byte-Array dieser Größe zu allokiieren, gibt der ESP32 eine Fehlermeldung aus. Es ist jedoch möglich den internen RAM des ESP32 um einen externen RAM zu erweitern, der über das SPI-Interface angeschlossen wird. Ist der gewählte externe RAM groß genug, kann sogar ein farbiges Bild in der maximalen Auflösung des Sensors von 640x480 Pixeln aufgenommen werden. Hierfür werden 921,6KB benötigt.

Glossar

TWI

Two Wire Interface (bekannt als I²C), zweidriger 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

THT

Through Hole Technology, Durchsteckmontage

FPS

Frames per Second

SCCB

Serial Camera Control Bus

SPI

Serial Peripheral Interface

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] Eagle Software: <https://www.autodesk.co.uk/products/eagle/overview>
- [twi_implementation] TWI Implementation: <https://github.com/rambo/TinyWire>
- [attiny254585] ATTiny25/45/85 Datenblatt: 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/>
- [ov7725] OV7725 Sensor: <https://www.ovt.com/sensors/OV7725>
- [esp32_kevin] Projekterfahrung EspressifESP32 –Fallbeispiel "Kevin", arendi, Benno Trutmann,

Anhang

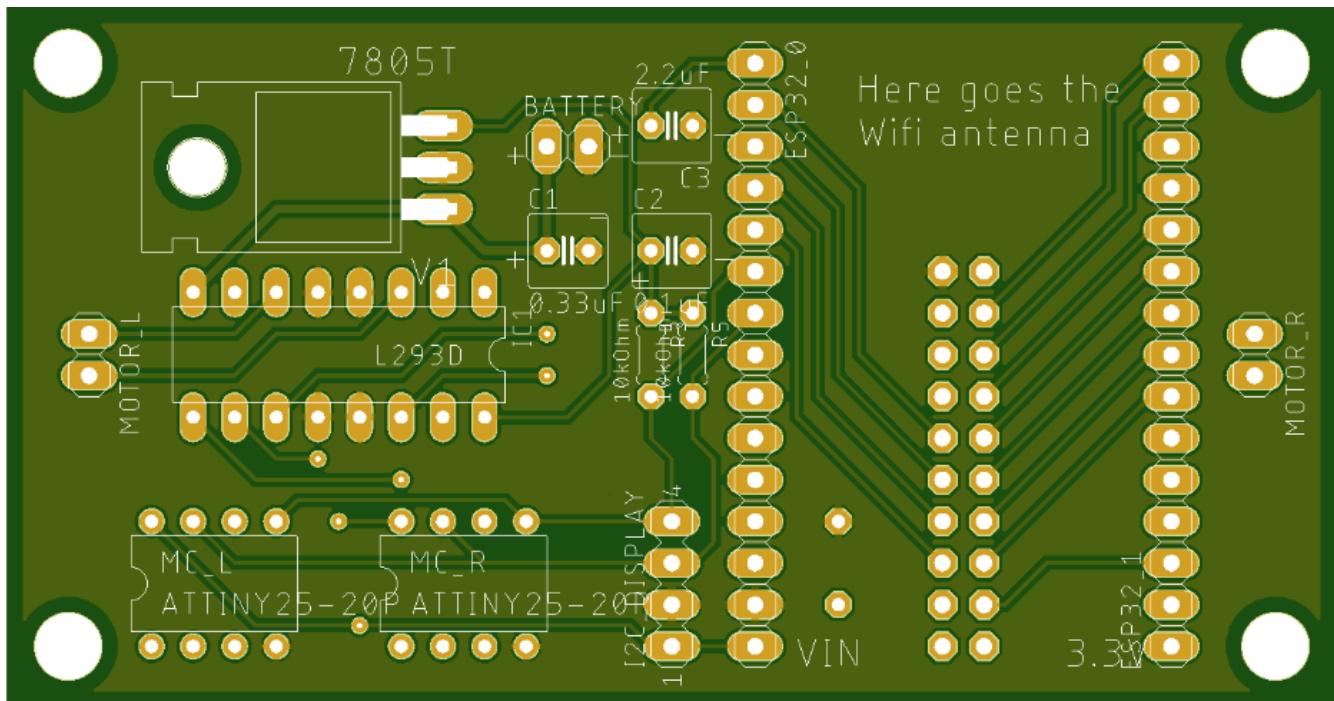


Abbildung 16. PCB Oberseite

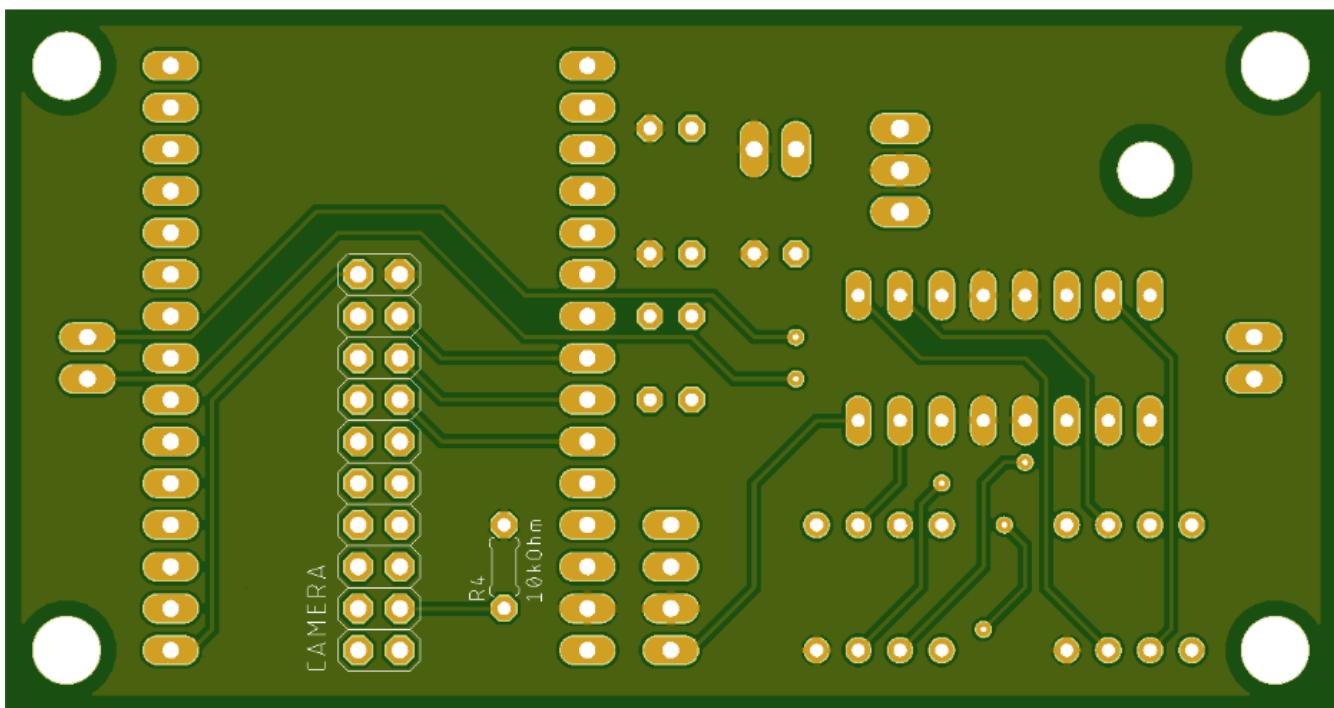


Abbildung 17. PCB Unterseite

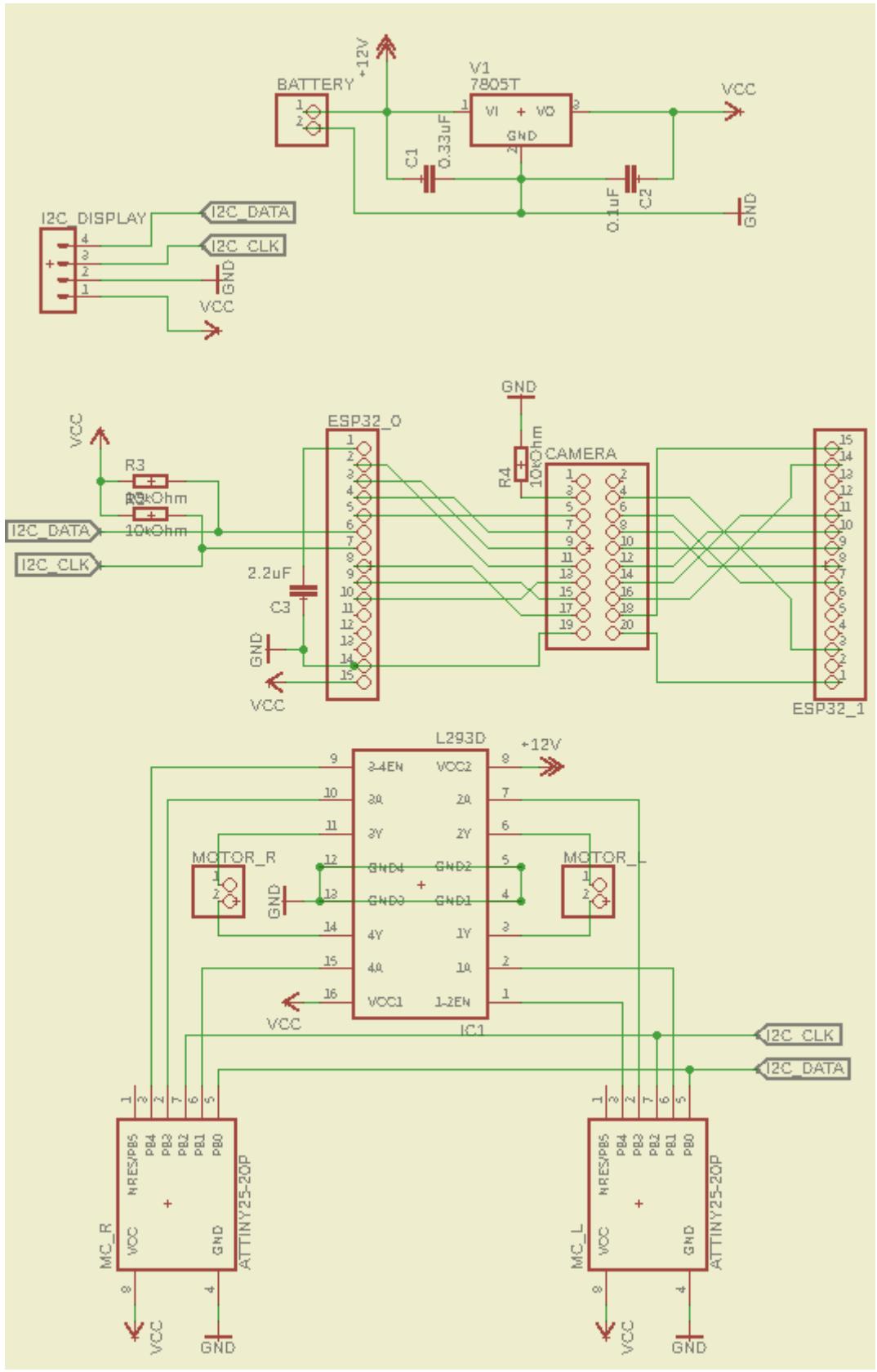


Abbildung 18. Kompletter Schaltplan