

ARDUINO DROHNE SELBER BAUEN

Vertiefungsarbeit

Autor: Christopher O'Connor, IN18z

Lehrperson: D. Wüst

Abgabe Datum: 27.10.2021

Schule: BBBaden, Wiesenstrasse 32, 5400 Baden

1 INHALTSVERZEICHNIS

2 Vorwort.....	3
3 Einleitung	4
4 Konzept	5
4.1 <i>Teilziele</i>	5
4.2 <i>Hilfsmittel</i>	6
5 Drohne 101.....	7
5.1 <i>Die ersten Drohnen</i>	7
5.2 <i>Zivile Einsatzgebiete</i>	8
5.3 <i>Arbeiter der Zukunft?</i>	10
5.4 <i>Quadrokopter Bewegung</i>	11
6 Analyse der Komponenten	13
6.1 <i>Rahmen</i>	13
6.2 <i>BLDC Motor</i>	14
6.3 <i>Elektronischer Geschwindigkeitsregler</i>	20
6.4 <i>LiPo Batterie</i>	21
6.5 <i>Propeller</i>	24
6.6 <i>RC Sender & Empfänger</i>	25
6.7 <i>Trägheitsmessungseinheit</i>	26
6.8 <i>Arduino</i>	27
6.9 <i>Verwendete Komponenten</i>	28
6.10 <i>Kommunikationsprotokoll PWM</i>	29
7 Schaltplan.....	30
8 Realisierung.....	32
8.1 <i>Prototypen</i>	32
8.2 <i>Montage</i>	49

8.3	<i>Programmieren</i>	62
9	Entstehungszeit	75
10	Schlusswort	76
11	Literaturverzeichnis	78
12	Abbildungsverzeichnis	81
13	Weiterverwendung	87
14	Eigenständigkeitserklärung	88
15	Reflexion	89
15.1	<i>Beurteilung des Arbeitsprozesses</i>	89
15.2	<i>Beurteilung der Arbeit als Ganzes</i>	89
15.3	<i>Beurteilung des Lernerfolgs</i>	89
15.4	<i>Umgang mit Schwierigkeiten</i>	90
15.5	<i>Zeitmanagement</i>	90
16	Anhang	91
16.1	<i>Interviewbestätigungen</i>	91
16.2	<i>Interview</i>	91
16.3	<i>Quellcode</i>	98

2 VORWORT

Von Natur aus bin ich sehr interessiert, wie etwas funktioniert. Es genügt mir meistens auch nicht nur oberflächlich eine Problemstellung zu verstehen. Die Frage «Wie funktioniert eine Drohne?» trage ich im Hinterkopf seit Beginn der Lehre. Ich durfte vor vier Jahren bei einer kleinen Demonstration eines angehenden Mediamatikers dabei sein, wie er mit Unterstützung seines Quadroopters grossartige Luftaufnahmen gemacht hat. Ich war von der Technologie begeistert und wollte mehr über Drohnen erfahren. Am liebsten hätte ich mir sofort selbst eine gebaut, weshalb ich darüber philosophiert habe, wie die Steuerung wohl funktionieren könnte. Mir wurde schnell klar, dass ich das entsprechende Wissen noch nicht beherrsche und habe die Idee verstaut. In den vergangenen vier Jahren konnte ich mir durch meine Ausbildung zum Applikationsentwickler sehr viel Wissen aneignen und möchte es nun endlich wagen, diese Frage in der Praxis zu klären.

An dieser Stelle möchte ich mich bei all denen, die mich bei der Durchführung dieser Arbeit unterstützt haben, herzlich bedanken:

- Bei der Santis Training AG, welche mich zu dieser Idee inspiriert hat und mir für diese Arbeit Zeit zur Verfügung gestellt hat.
- Herrn Simon Wilks, für ein sehr spannendes und lehrreiches Interview.
- Herrn Daniel Wüst, für die kritische Hinterfragung der vielen vorgeschlagenen Themen.

3 EINLEITUNG

Haben Sie sich beim letzten Sport Event, welches Sie vor dem TV angeschaut haben, gefragt, wie diese spektakulären Bilder aus der Luft aufgezeichnet wurden? Oder beim letzten «Züri Fäscht» (2019), wie am Abend die spektakuläre Lichtshow in der Luft entstanden ist?

Unbemannte Luftfahrzeuge, auch Drohne genannt, ermöglichen es! Aber wie funktionieren Sie? Gibt es verschiedene Typen? Wie kann man selbst eine bauen? Und wie funktioniert die Steuerung? Genau mit diesen Fragen und Thematik möchte ich mich befassen.

In dieser Arbeit werde ich einen eigenen Quadrokopter zusammenbauen und ihm das Fliegen beibringen. Dazu benötigt es eine gute Recherche zu den dafür notwendigen einzelnen Komponenten und wie diese miteinander kompatibel sind. Nachdem ich die benötigte Hardware ausgesucht und bestellt habe, werde ich diese miteinander durch drahten, löten und schrauben verbinden. Zuletzt werde ich eine Applikation entwerfen, um den Quadrokopter fernsteuern zu können und ihm das eigenständige Schweben beibringen.

Damit ich mich genügend mit den einzelnen Komponenten befasse, werde ich auf einen Fertigbausatz verzichten. Weil ich mich auch mit der Funktionalität auseinandersetzen möchte, verzichte ich ebenfalls auf einen fertigen Flugkontroller. Dieser wird durch einen programmierbaren Mikrocontroller namens Arduino ersetzt sowie das Programm dazu selbst programmiert.

Wurden die Grundziele vollständig erreicht, kann die Drohne mit zusätzlichen Messinstrumenten (Sensoren, GPS, Kamera etc.) ausgestattet werden. Durch die eigene Programmierung sind die Erweiterungsmöglichkeiten lediglich auf die maximale Tragfähigkeit des Quadroopters beschränkt.

Mit dem gewonnenen Grundwissen aus dieser Arbeit und dem Interview eines Fachspezialisten im Bereich der Autopiloten für Drohnen erhoffe ich mir das Grundsystem in Zukunft weiter entwickeln zu können, welches nicht nur mehr Sensoren beinhaltet, sondern auch autonome Aufgaben für mich und andere selbstständig erledigen kann.

Durch das Publizieren dieser Vertiefungsarbeit wird jeder, welcher sich damit befassen möchte, einen einfachen Einstieg in die Welt der Drohnen erhalten und eigenständig bauen können.

4 KONZEPT

Das Konzept dient als Leitfaden für die Realisierung dieses Projektes.

4.1 Teilziele

Um einen besseren Überblick zu erhalten, wird das Gesamtziel in kleinere überschaubare Ziele unterteilt:

4.1.1 Komponenten

Mit einer tiefgründigen Recherche sollen die benötigten Komponenten ausfindig gemacht werden. Dabei wird besonders ein Einblick in die Funktionsweise erarbeitet.

4.1.2 Schaltplan

Ausgehend von den Datenblättern der einzelnen Komponenten und den deklarierten Quellen kann ein Schaltplan entworfen werden. Dieser dient als Vorlage für die Realisierung.

4.1.3 Prototypen

Damit Probleme frühzeitig erkannt und ausgegrenzt werden können, werden kleinere Subsysteme als Prototypen realisiert. Diese Prototypen dienen der Erforschung, wie und ob die einzelnen Komponenten miteinander kommunizieren können und der frühzeitigen Problemfindung. Die daraus gewonnenen Erfahrungen und Erkenntnisse fließen in die Realisierung des Gesamten.

4.1.4 Montage

Anhand der Schaltpläne und dem Entwurf werden die einzelnen Komponenten zusammengebaut und die Schnittstellen der einzelnen Komponenten miteinander verbunden.

4.1.5 Programmieren

Der Code wird ausgehend von den Prototypen erweitert. Mit dem Wissen, wie die Komponenten untereinander kommunizieren, kann die nötige Business Logik implementiert werden und schlussendlich Schnittstellen für potenzielle Erweiterungen gemacht werden.

Die Programmierung wird in drei verschiedenen Meilensteinen (Releases) erfolgen:

1. Quadrokopter schweben lassen (Altitude, Throttle)
2. Quadrokopter Bewegung (Pitch, Yaw, Roll)
3. Autonomes Schweben

4.2 Hilfsmittel

4.2.1 Sachmittel

ANZAHL	BESCHRIFTUNG
--------	--------------

1X	Laptop
1X	Internetanschluss
1X	Lötutensilien

4.2.2 Programme

Zur Realisierung dieser Arbeit werden folgende Programme verwendet:

NAME	VERWENDUNGSZWECK
------	------------------

MICROSOFT WORD	Dokumentation
FRITZING	Design Software für das Erstellen von Schaltungen
DRAW.IO	Diagramm Erstellung
ARDUINO STUDIO	Entwicklungsumgebung für Arduino Source Code
WINDOWS SNIPPING TOOL	Bildschirmaufnahmen
MICROSOFT PAINT	Bilder zeichnen

4.2.3 Quadrokopter Komponenten Mengengerüst

Die für einen typischen Quadrokopter minimal benötigten Komponenten werden nachfolgend aufgelistet:

ANZAHL	BESCHRIFTUNG
--------	--------------

1X	Rahmen
4X	Motor
4X	Elektronischer Geschwindigkeitsregler (ESC)
4X	Propeller
1X	Batterie
1X	Arduino Uno
1X	RC Empfänger
1X	RC Sender

Eine detailliertere Liste mit den exakt verwendeten Komponenten wird vor der Realisierung erstellt.

5 DROHNE 101

«Eine Drohne ist ein unbemanntes Luft- oder Unterwasserfahrzeug, das entweder von Menschen ferngesteuert oder von einem integrierten oder ausgelagerten Computer gesteuert und damit teil- oder vollautonom wird.» (Prof. Dr. Bendel, 2021)

5.1 Die ersten Drohnen

Die Geschichte der Drohne ist eng mit der Entwicklung der militärischen Luftfahrt in Grossbritannien verbunden. Das Konzept der Funkfernsteuerung eines Flugzeugs wurde durch die Royal Flying Corps (RFC) bereits während des Ersten Weltkriegs eingeleitet.

Am Mittwoch, dem 21. März 1917, startete die RFC ihr erstes unbemanntes Flugzeug, welches auf per Funk vom Boden übertragene Befehle reagierte. Dabei handelte es sich um ein Flugzeug in voller Grösse entworfen von Archibald Low. (Millis, 2019)



Abbildung 1 Erste Flugdrohne (<https://www.iwm.org.uk/history/a-brief-history-of-drones>)

Die britischen Streitkräfte betrieben jahrelang eine grosse Flotte speziell gebauter, ferngesteuerter Flugzeuge. Diese britischen unbemannten Luftfahrzeuge wurden 1935 in Dienst gestellt und bis Ende der 1940er-Jahre von den britischen Streitkräften im In- und Ausland eingesetzt. Diese Doppeldecker in Originalgrösse waren eine Variante der weltberühmten Tiger Moths, die von Operatoren am Boden gesteuert wurden und hauptsächlich als Ziele zur Ausbildung von Flugabwehrschützen verwendet wurden. Sie wurden „Queen Bees“ genannt und waren der Höhepunkt einer langjährigen Entwicklung. (Millis, 2019)



Abbildung 2 De Havilland Queen Bee seaplane L5894 (<https://www.iwm.org.uk/history/a-brief-history-of-drones>)

5.2 Zivile Einsatzgebiete

Drohnen haben ein immer grösser werdendes Einsatzgebiet. Sie werden lange nicht mehr nur für militärische Zwecke oder nur für Video- und Fotoaufnahmen eingesetzt. Nachfolgend einige moderne Einsatzgebiete:

5.2.1 Landwirtschaft

In der Landwirtschaft können Drohnen durch den Einsatz von Nahinfrarotsensoren Temperaturschwankungen im Feld erkennen welche von Wassermangel, Schädlingsbefall oder Düngermangel entwickelt werden. (Brown J., My DroneLab, 2019)

5.2.2 Logistik

Drohnen können, wie unter anderem Amazon bewiesen hat, Pakete an Kunden gezielt ausliefern. Hierzulande verwendet die Schweizer Post seit 2017 den weltweit ersten kommerziellen Betrieb in städtischem Gebiet und transportiert regelmässig medizinische Produkte sowie Blutproben mit autonomen Lieferdrohnen in den Städten Bern, Lugano und Zürich. (Swiss Post, 2021)

5.2.3 Sicherheitsinspektionen

Regelmässige Inspektionen von Infrastrukturen können mithilfe von Drohnen zu einem sehr niedrigen Kostenansatz durchgeführt werden im Vergleich zu herkömmlichen manuellen Inspektionen. (Brown J. , My DroneLab, 2019)

5.2.4 Racing

Drohnen haben auch den Weg auf die Rennstrecke gefunden! Für die noch wenig bekannte Sportart gibt es bereits grosse Turniere, Wettkämpfe und eine eigene Internationale «Drone-Racing League». Dabei wird eine Kamera, welche sich auf der Drohne befindet, mit einem VR Headset verbunden und ein abgesteckter Kurs durchflogen. (Brown J. , My DroneLab, 2019)

5.2.5 Hobby

Hobby Drohnen werden in drei verschiedenen Hauptkategorien unterschieden:

			
<i>Abbildung 3 Flugzeug (https://pixabay.com/vectors/aircraft-drone-usa-airplane-plane-161415/)</i>	<i>Abbildung 4 Helikopter (https://pixabay.com/photos/helicopter-model-propeller-aircraft-1934758/)</i>	<i>Abbildung 5 Multikopter (https://pixabay.com/photos/drone-hexacopter-uav-rpas-aircraft-2168938/)</i>	<i>Abbildung 6 VTOL (https://wingtra.com/mapping-drone-wingtraone/vtol-drone/)</i>
Flugzeuge und Gleiter	Helikopter	Multikopter	VTOL

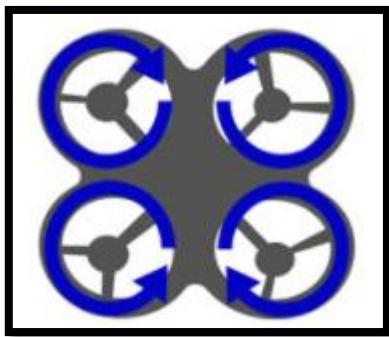
VTOL Drohnen sind eine Mischform. Sie haben die Eigenschaft, vertikal starten sowie landen zu können, aber in der Luft sich wie ein Flug fortzubewegen.

5.3 **Arbeiter der Zukunft?**

Das Beliefern von abgeschnittenen Dörfern durch Naturkatastrophen, Überwachungen von sensiblen Anlagen, Unterstützung bei aufwendiger Handarbeit in der Landwirtschaft usw. Experten sind sich einig, dass der Einsatz von Drohnen in den kommenden Jahren stark zunehmen wird. (Swiss Post, 2021)

Auch mein Interviewpartner ist sich darüber einig: «I see drones everywhere. When we look at things right now, especially crop spraying, which means spraying fertiliser, pesticides and seeding. I see them taking over and replacing all the manual labour. It is happening now with our drones. [...] I think what a big turning point of those types of drones will be, but also all and the biggest barrier at the moment is power source. Our biggest limitation is batteries. If you could have 100 times the power density, then you could fly a lot longer and carry a lot more weight. This will have a huge impact on the industry. Similar to cars as well, they do also a better job than a person can. The sooner you get a human out of the loop the better. Autopilots will do a much more exacter job. What we see with drones as well in terms of collecting data or spraying is that it is much more precise. [...] I think drones will get more efficient and obviously smarter and probably into some areas, you will not be as happy about. A Better opportunity to monitor people and all that kind of thing. But the limits will extend, I mean where do drones stop and aircraft begin? [...] Eventually, at some point in the future, you will not have no more pilots. Same as cars, right? You have self-driving cars because they are proven to be much safer than having a human behind the steering wheel and the same will happen with manned aircraft. You have these two worlds approach each other and merge in the middle».

5.4 Quadrokopter Bewegung



Die dreidimensionale Bewegung eines Quadroopters wird durch den Auftrieb der Rotoren relativ zueinander gesteuert. Damit der Quadroopter sich in der Luft nicht wie ein Helikopter ohne Heckrotor im Kreis dreht, müssen jeweils die gegenüberliegenden Rotoren in die gleiche Richtung Drehen und die anderen beiden Rotoren in die Gegenrichtung. (FPVRacing, 2021)

Abbildung 7 Quadrokopter Rotordrehrichtung (FPVRacing, 2021)

Um die Balance in der Luft zu erhalten, werden von einer Trägheitsmesseinheit (IMU) kontinuierlich Messungen im Dreidimensionalen Raum durchgeführt und die Geschwindigkeit der Rotoren angepasst.

Der Quadroopter kennt folgende vier Bewegungsmöglichkeiten:

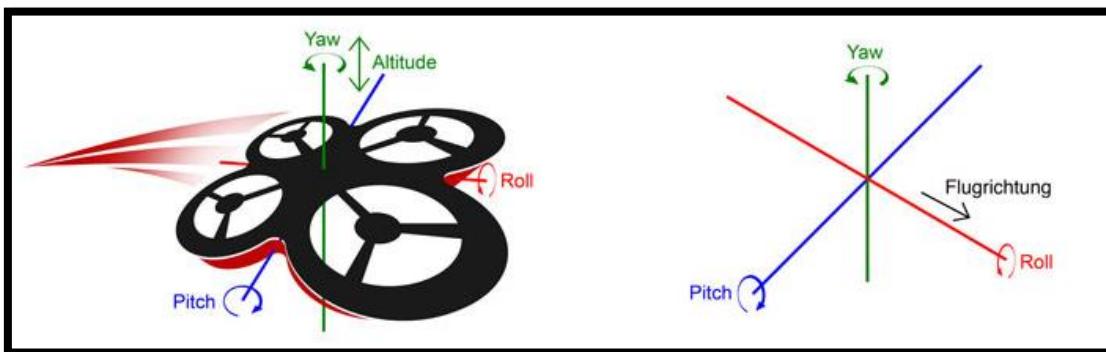


Abbildung 8 Quadrokopter Bewegungsachsen (FPVRacing, 2021)

5.4.1 Altitude

Die Flughöhe wird durch die einheitliche Geschwindigkeit aller Rotoren definiert. (FPVRacing, 2021)

5.4.2 Roll

Eine seitliche Neigung wird durch die Geschwindigkeitserhöhung beider Motoren auf einer Seite erzeugt. (FPVRacing, 2021)

5.4.3 Pitch

Die Neigung in Flugrichtung wird durch die Geschwindigkeitserhöhung der Front- respektive Heckmotoren erzeugt. (FPVRacing, 2021)

5.4.4 Yaw

Die Drehung um die Z-Achse wird durch die Geschwindigkeitserhöhung der diagonalen Rotoren erzeugt. Für eine Drehung im Gegenuhrzeigersinn werden bei den im Uhrzeigersinn drehenden Motoren die Geschwindigkeit erhöht. (FPVRacing, 2021)

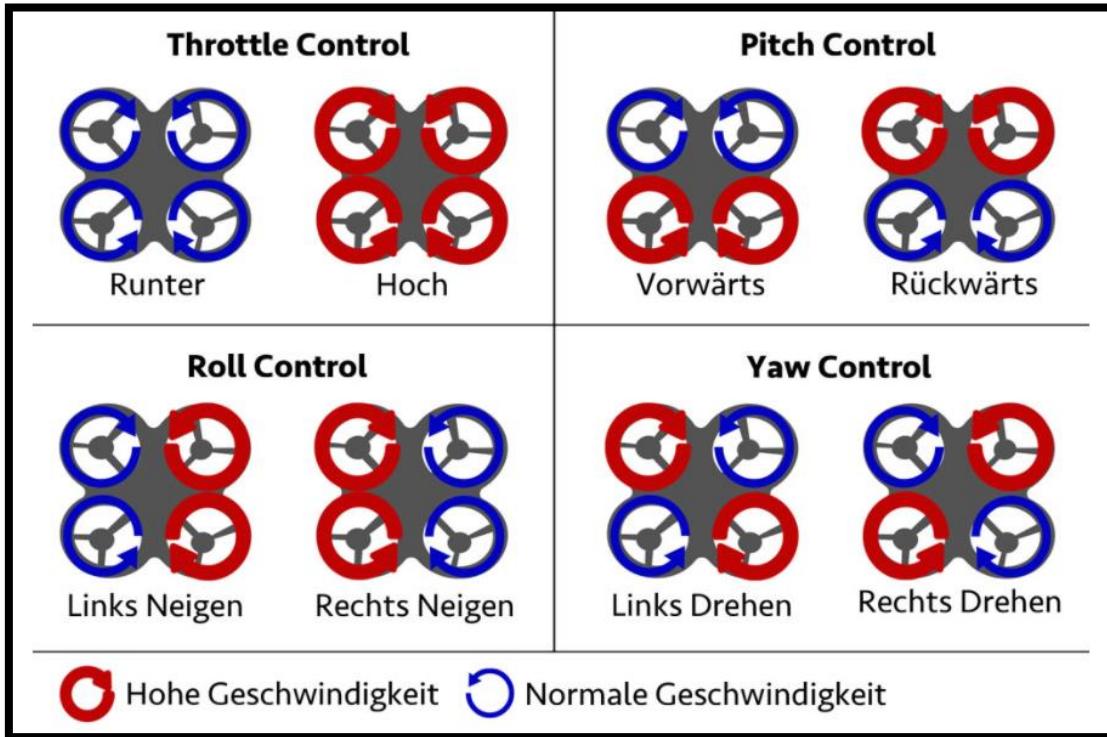


Abbildung 9 Quadrokopter Bewegung durch Rotoren Geschwindigkeitsanpassung (FPVRacing, 2021)

Beim Roll- und Pitch-Manöver muss die Geschwindigkeit erhöht werden bei den gegenüber der gewünschten Bewegungsrichtung liegenden Motoren. Damit eine Flugrichtung beispielsweise nach vorne vonstatten kommt, müssen die Heckmotoren schneller drehen als die vorderen.

6 ANALYSE DER KOMPONENTEN

In diesem Kapitel werden die einzelnen Komponenten genauer betrachtet. Unter anderem soll ein Einblick in die Funktionsweise aufgezeigt werden und was es beim Kauf zu beachten gibt.

6.1 Rahmen

Ein guter Quadrokopter Rahmen ist essenziell. Er muss stark, aber gleichzeitig auch leicht sein. Auf dem Markt gibt es viele verschiedene Arten von Rahmen. Sie werden nach Form, Material und Abmessungen klassifiziert.

6.1.1 Steifheit

Ein steifer Rahmen kann die Flugstabilität enorm erhöhen. Er sollte aber auch flexibel genug sein, um Vibrationen der Motoren kompensieren zu können. Dies ist für die Trägheitsmessungseinheit enorm wichtig, damit die genauen Messungen nicht zu stark verfälscht werden. (Brown J., My DroneLab, 2019)

6.1.2 Material

Zu den meistverwendeten Materialien gehören Aluminium, Kohlefaser, Holz und Glasfaser. Der ungeschlagene König der Materialien vor allem im Bereich des FPV Sports ist Kohlefaser. Durch seine leichte Struktur erhöht er die enorm wichtige Agilität. (Brown J., My DroneLab, 2019)

6.1.3 Grösse

Die Rahmengrösse wird standardisiert in Zoll angegeben und ist aussagekräftig über die Grösse der zu verwendeten Propeller. Als zweiten Indikator über die Dimensionen dient der Radstand der Motoren. Je kleiner ein Rahmen ist, desto schneller reagieren Befehle und entsprechend ist er schwieriger zum Fliegen. Für den erstmaligen Eigenbau eignen sich grössere Rahmen, weil es einfacher ist, die einzelnen Komponenten zu montieren und die zusätzliche Stabilität führt zu weniger Bruchlandungen in den ersten Fluglektionen.

6.1.4 Verwendungszweck

Der Verwendungszweck spielt beim Rahmen einer der grössten Rolle. Drohnen, welche hauptsächlich für Photographie oder Transport verwendet werden, favorisieren Stabilität über schnelle Richtungswechsel. Das genaue Gegenteil findet man im FPV Sport, oftmals werden dafür sogar Trägheitsmessungseinheiten absichtlich weggelassen, um die maximale Kontrolle der Drohne zu erhalten. (Brown J., My DroneLab, 2019)

6.2 BLDC Motor

Im nachfolgenden Kapitel wird die Funktionsweise eines «Brushless Direct Current Motors» als Innenläufer erklärt.

6.2.1 Funktionsweise

Ein BLDC Motor besteht aus einem Stator und einem Rotor. Der Rotor ist ein zweipoliger Permanentmagnet im inneren Kern und das drehende Element eines Motors. Der Stator besteht aus mehreren Spulen, welche an den Rotor angrenzen. (Nedelkovski, 2019)

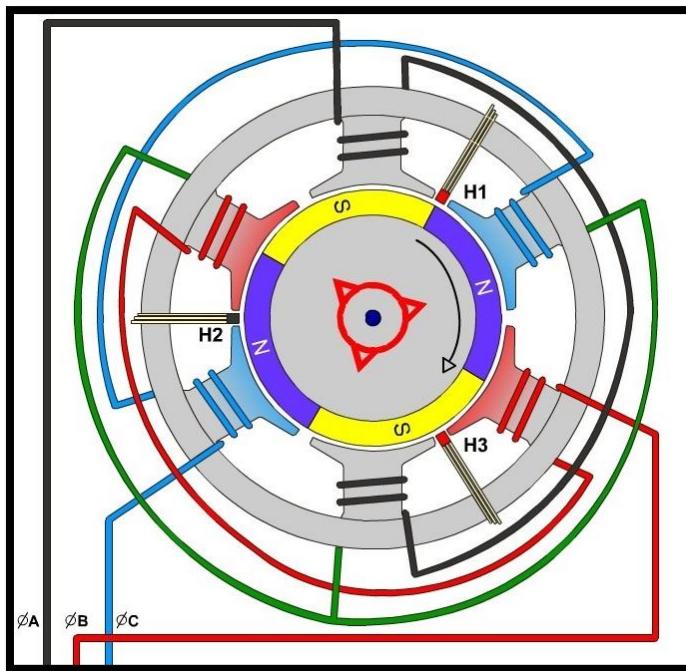


Abbildung 10 BLDC Motor Konzept (<https://electricalbaba.com/brushless-dc-bldc-motor/>)

Mit dem entsprechenden Stromfluss erzeugt die Spule ein Magnetfeld, welches den Permanentmagneten des Rotors anzieht. Wird jede Spule der Reihe nach aktiviert, dreht sich der Rotor aufgrund der Kraftwechselwirkung zwischen Permanentmagnet und Elektromagnet weiter. Fließt der Strom in umgekehrter Richtung durch die Spule, wird der Nord- und Südpol des Magneten vertauscht. (Nedelkovski, 2019)

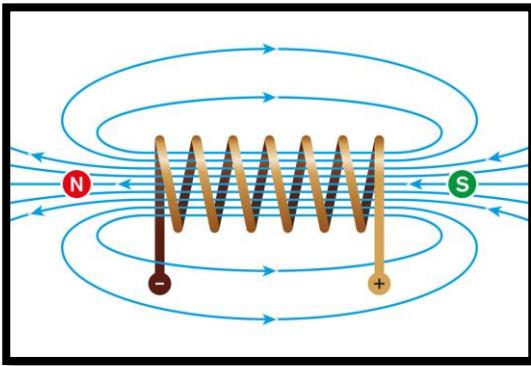


Abbildung 11 Magnetfeld einer Spule (<https://quizlet.com/306997277/magnetic-field-around-a-solenoid-diagram/>)

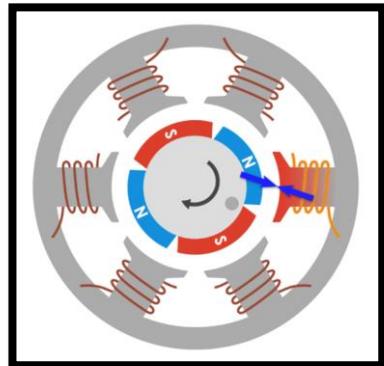


Abbildung 12 Kraftwechselwirkung (Nedelkovski, 2019)

Die Effizienz des Motors kann erhöht sowie die Komplexität der Ansteuerung minimiert werden, wenn die zwei gegenüberliegenden Spulen so verbunden werden, dass entgegengesetzte Pole erzeugt werden. Somit kann von den ursprünglich sechs Spulen, welche einzeln angesteuert werden müssen, auf lediglich drei halbiert werden. Bei jedem Intervall ist eine Phase ausgeschaltet, eine Phase mit positivem und eine mit negativem Strom geladen. Die Spulen beim BLDC Motor können auf ein beliebig-Faches von drei erweitert werden und untereinander so verbunden werden, dass immer drei Phasen entstehen. Die drei Phasen werden mit Drähten aus dem Motor zu einem elektronischen Geschwindigkeitsregler geführt. (Nedelkovski, 2019)

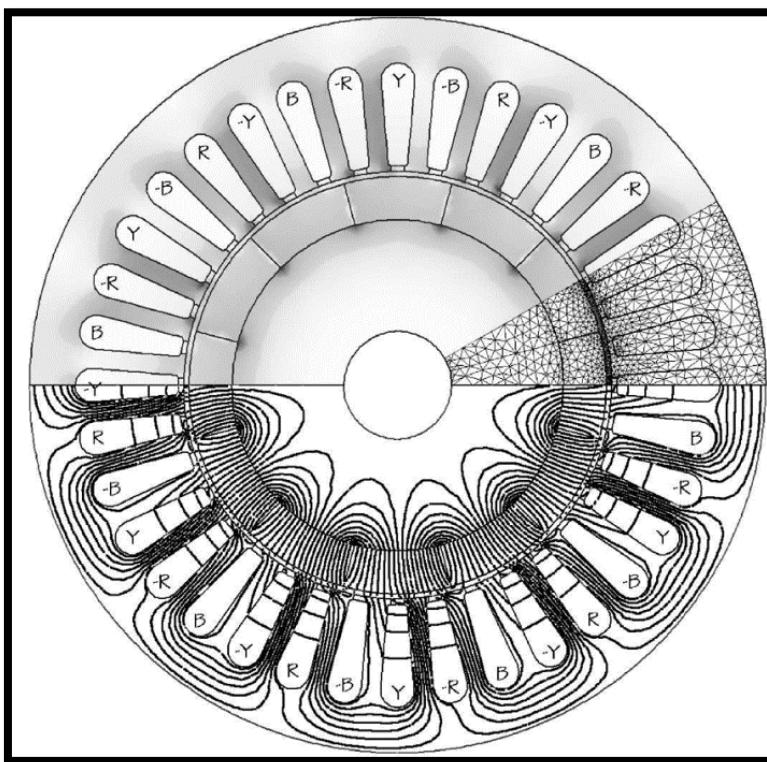


Abbildung 13 Visualisierung der Magnetfelder (https://www.researchgate.net/figure/FE-model-of-a-12-pole-BLDC-prototype-motor-with-a-distributed-finely-pitched-stator_fig5_4099599)

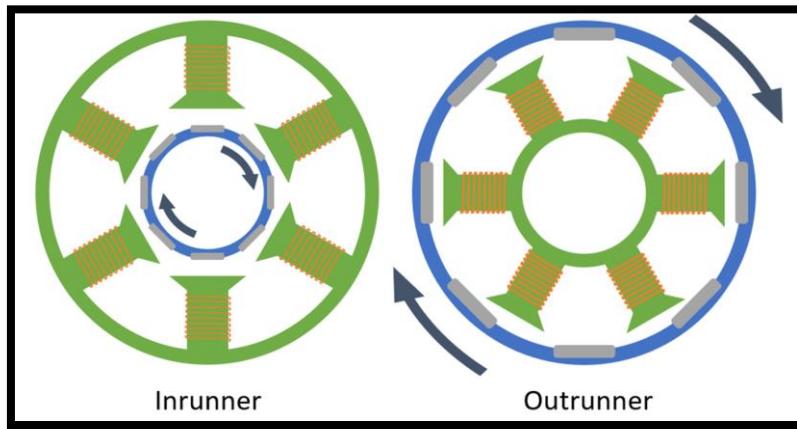


Abbildung 14 Innen- und Aussenläufer BLDC Motor (<https://www.tytorobotics.com/blogs/articles/how-brushless-motors-work>)

BLDC Motoren können Innen- sowie als auch Aussenläufer sein. Innenläufer drehen sich um die eigene Achse innerhalb des Stators. Beim Aussenläufer wird das Permanentmagnet nach aussen gesetzt und dreht sich um den Stator.

6.2.2 Verhältnis von Schubkraft zu Gewicht

Der von allen Motoren erzeugte maximale Auftrieb sollte mindestens das Doppelte des Gesamtgewichts des Quadroopters betragen.

Berechnung der minimalen Schubkraft:

$$\frac{100\% \text{ Throttle Pull} * \text{anzahl Motoren}}{2} \geq \text{Totalgewicht}$$

Das 2:1 Verhältnis gilt als absolutes Minimum. Für schnelleres Fliegen wie bei Drohnenrennen sollte das Verhältnis von Schub zu Gewicht viel höher sein. Beim Profisport beispielsweise sind Verhältnisse von bis zu 13:1 zu sehen. Für die aller erste Drohne sollte ein Verhältnis von 4:1 angestrebt werden. Dies bringt eine gute Kontrolle und bietet in Zukunft auch Platz für zusätzliche Nutzlast. (Liang, OscarLiang, 2021)

Zuerst sollte das geplante Gesamtgewicht ungefähr ermittelt werden:

KOMPONENTE	GEWICHT
RAHMEN	128g
ARDUINO UNO	44g
BATTERIE	135g
KABEL	65g
ESC	4 x 39g = 156g
MOTOR	4 x 30g = 120g
PROPELLE	4 x 4g = 16g
RC - EMPFÄNGER	8g
TOTALGEWICHT	672g

Anschliessend kann aus dem Datenblatt des gewünschten Motors das «Pull» Gewicht ermittelt werden und mit dem Gesamtgewicht mittels der Formel abgeglichen werden.

Prop (inch)	Voltages (v)	Throttle (%)	Load Current (A)	Pull(g)	Power(W)	Efficiency (g/W)	Temperature(in full throttle load 1 min)
5.1x3.1x3	16	50%	9.4	589	150.4	3.916	66C°
		60%	14.07	767	225.1	3.407	
		70%	20.15	966	322.4	2.996	
		80%	27.28	1118	436.5	2.561	
		90%	34.68	1261	554.9	2.273	
		100%	43.23	1522	691.7	2.200	

Abbildung 15 Datenblatt Motor-Pull-Gewicht (<https://www.amazon.com/iFlight-2750KV-Brushless-Racing-Quadcopter/dp/B07Y9JK2MW>)

6.2.3 KV

«KV» ist eine Richtlinie dafür, wie viele Umdrehungen ein Motor macht. Wenn der Motor dreht, entsteht ein elektromagnetisches Feld. Diese erzeugte Spannung wird als Gegen-EMK bezeichnet. Da die Gegen-EMK proportional zur Drehzahl des Motors ist, wird sie in U/min/V ausgedrückt. KV gibt die Beziehung zwischen der Gegen-EMK-Konstante und der Drehzahl des Motors. (IntoFPV, 2018)

Ein 2800KV Motor gibt 1 Volt Gegen-EMK, wenn er sich mit 2800 U / min dreht. Die maximale Drehzahl ist abhängig von der zugeführten Stromspannung und kann grob errechnet werden: (Liang, OscarLiang, 2021)

$$KV * V = U/min$$

Beispielsweise bei einem 2800KV Motor und einem 3S-LiPo-Akku (11,1V) dreht sich der Motor mit etwa 31'080 U / min ohne Propeller. Sobald ein Propeller am Motor montiert wird, nimmt die Drehzahl aufgrund des Luftwiderstands drastisch ab.

6.2.4 Motorgrösse

Die Grösse von BLDC Motoren wird durch eine 4-stellige Zahl angegeben – AABB. «AA» ist die Statorbreite, während «BB» die Statorhöhe ist. (Liang, OscarLiang, 2021)

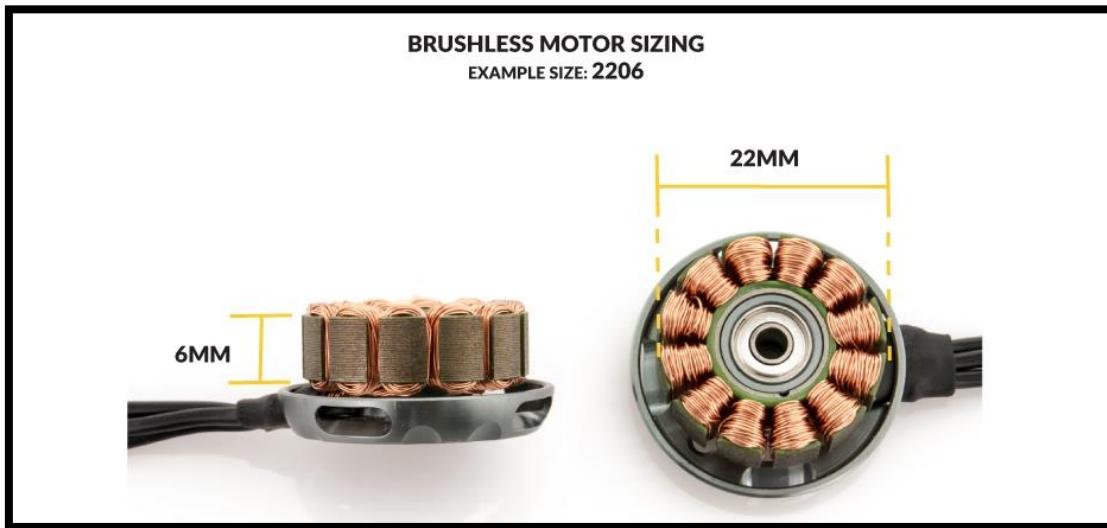


Abbildung 16 BLDC Motorgrösse (<https://www.getfpv.com/motors/what-is-a-drone-motor.html>)

Nach allgemeinem Konsens:

- Höherer Stator = mehr Leistung bei höheren Drehzahlen
- Breiterer Stator = mehr Drehmoment bei niedrigeren Drehzahlen

6.2.5 Größenrichtwert

Als Hilfsmittel für die richtigen Größen und Kombinationen des Rahmens, Propeller, Motorgrösse und KV dient folgende Tabelle:

Rahmengrösse	Propellergrösse	Motorgrösse	KV
> 150mm	3"	1105 – 1306	3000KV +
180mm	4"	1806, 2204	2600KV – 3000KV
210mm	5"	2205 - 2208, 2305	2300KV – 2600KV
250mm	6"	2206 - 2208, 2306	2000KV – 2300KV
350mm	7"	2506 – 2508	1200KV – 1600KV

(Liang, OscarLiang, 2021)

6.2.6 CW und CCW Motoren

Die Drehrichtung bei BLDC Motoren spielt keine Rolle. Um einen Motor in eine gewünschte Richtung drehen zu lassen, werden die Verbindungskabel vom ESC unterschiedlich angebracht:

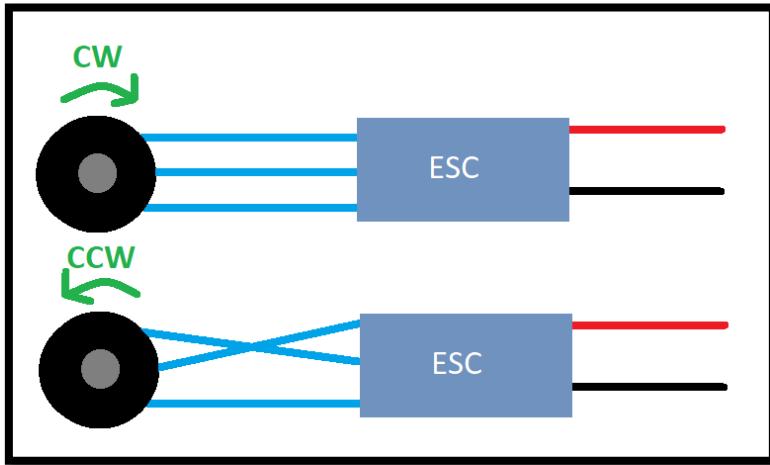


Abbildung 17 CW und CCW Verbindung (O'Connor)

6.3 Elektronischer Geschwindigkeitsregler

Ein elektronischer Geschwindigkeitsregler (ESC) steuert die Geschwindigkeit des BLDC Motors, indem er die entsprechende Schaltungsvariation aktiviert, um das rotierende Magnetfeld zu erzeugen, damit sich der Motor dreht. Je höher die Frequenz oder je schneller der ESC die sechs Intervalle durchläuft, desto höher ist die Drehzahl des Motors. (Nedelkovski, 2019)

Damit eine Phase Negativ, Positiv oder Ausgeschalten sein kann, ergibt sich nachfolgender Schaltplan sowie der Schaltzyklus für eine 360° Drehung:

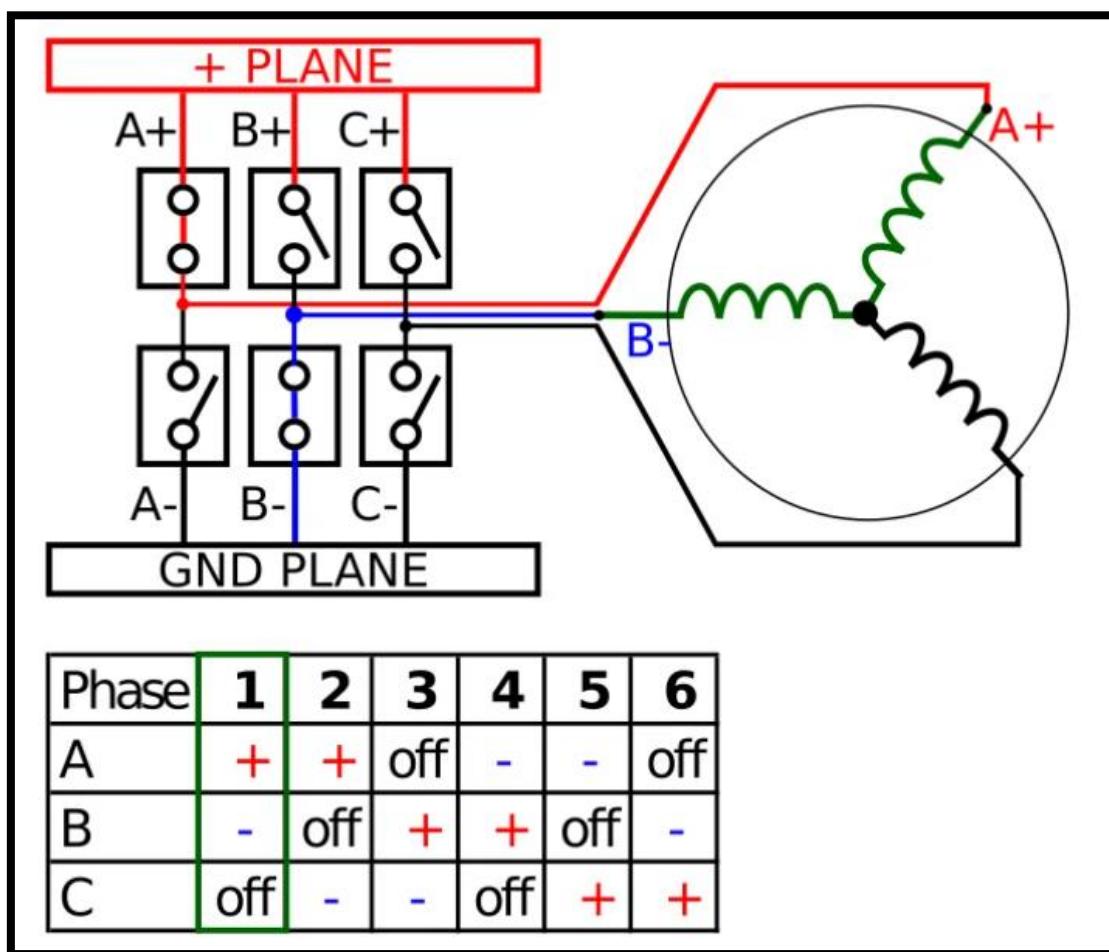


Abbildung 18 ESC Schaltplan und 360° Intervall (<https://e-surfer.com/de/selbstbau-eines-elektro-longboards/>)

6.3.1 Kaufentscheidung

Bei der Auswahl eines Geschwindigkeitsreglers muss darauf geachtet werden, dass dieser den Motor mit genügend Ampere versorgen kann und die Anzahl „S“ der Batterie passend ist. Das Gewicht sollte auch nicht vernachlässigt werden.

6.4 LiPo Batterie

Lithium-Polymer-Akkus, allgemein bekannt als LiPo, haben eine hohe Energiedichte, eine hohe Entladerate und ein geringes Gewicht, was sie ideal für RC-Anwendungen macht.

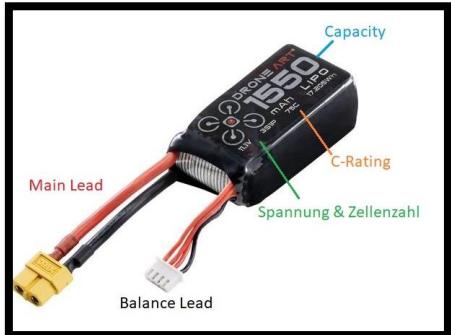


Abbildung 19 LiPo Batterie (<https://www.conrad.ch/de/p/droneart-modellbau-akkupack-lipo-11-1-v-1550-mah-zellen-zahl-3-75-c-softcase-xt60-1784861.html>)

6.4.1 Batteriespannung und Zellenzahl

LiPo Batterien bestehen aus Zellen, jede Zelle hat eine Stromspannung von 3,7V. Wird eine höhere Spannung benötigt, werden die Zellen zu einer einzigen Batterie in Reihe geschaltet.

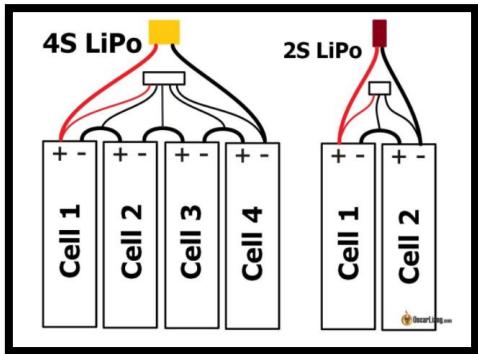


Abbildung 20 LiPo-Zellen (Liang, OscarLiang, 2021)

LiPo Batterien werden nicht durch die Batteriespannung gekennzeichnet, sondern wie viele Zellen «S» sie hat.

S	Anzahl Zellen	Stromspannung
1S	1	3.7V
2S	2	7.4V
3S	3	11.1V
4S	4	14.8V
5S	5	18.5V

(Liang, OscarLiang, 2021)

6.4.2 Kapazität und Grösse

Die Kapazität eines LiPo-Akkus wird in Milliamperestunden (mAh) gekennzeichnet. Es zeigt im Grunde an, wie viel Strom eine Stunde lang aus dem Akku gezogen werden kann, bis dieser leer ist. Wird die Akkukapazität erhöht, kann die Flugzeit verlängert werden, aber das Gewicht und die physische Grösse werden auch grösser. Es gilt dabei einen Kompromiss zwischen Kapazität und Gewicht zu finden. (Liang, OscarLiang, 2021)

6.4.3 Entladerate (C-Rating)

Mit dem C-Wert und der Kapazität (Ah) einer Batterie kann der theoretisch maximal kontinuierliche Entladestrom berechnet werden.

$$\text{Maximaler Entladestrom}(A) = C \text{ Bewertung} * \text{Kapazität}(Ah)$$

Wenn die C-Bewertung zu niedrig ist, wird ungenügend Strom an die Motoren geliefert und der Quadrokopter wird unversorgt. Die Batterie kann beschädigt werden, wenn die Stromaufnahme den Sicherheitswert überschreitet. (Liang, OscarLiang, 2021)

6.4.4 Stromanschlüsse

Alle LiPo Akkus haben zwei Anschlüsse: Ein Balancekabel und ein Entladekabel. Das Entladekabel wird für den Stromfluss verwendet. Sei dies für den Strom der Motoren oder für die Ladung. Das Balancekabel wird nur verwendet während der Ladung der Batterie damit alle Zellen gleichmässig geladen werden. (Liang, OscarLiang, 2021)

6.4.5 Bedarf ermitteln

Um die passende Batterie für den Quadrokopter zu finden, muss der Bedarf ausfindig gemacht werden. Dies kann erst passieren, wenn bereits der Motor und die Propeller ausgesucht wurden. Im Datenblatt des Motors befindet sich eine Tabelle mit der Anzahl Ampere, welche benötigt werden mit der Kombination von Propeller:

Prop (inch)	Voltages (V)	Throttle (%)	Load Currency (A)	Pull(g)	Power(W)	Efficiency (g/W)	Temperature(in full throttle load 1 min)
5.1x3.1x3	16	50%	9.4	589	150.4	3.916	66C*
		60%	14.07	767	225.1	3.407	
		70%	20.15	966	322.4	2.996	
		80%	27.28	1118	436.5	2.561	
		90%	34.68	1261	554.9	2.273	
		100%	43.23	1522	691.7	2.200	

Abbildung 21 BLDC Motor Datenblatt Bedarfsermittlung (<https://www.amazon.com/iFlight-2750KV-Brushless-Racing-Quadcopter/dp/B07Y9JK2MW>)

Die maximale Gesamtstromaufnahme für meine Drohne mit vier Motoren und 5.1x3.1x3 Zoll Propellern wäre $43.23A * 4$ bei 100% Drosselung = **172.92A**. Andere Komponenten benötigen auch Strom, diese können aber bei der Berechnung ignoriert werden, da sie im Vergleich zu den Motoren nur minimal Strom benötigen.

6.4.5.1 Kapazität

Um die Batteriekapazität (mAh) für die jeweilige Grösse des Quadroopters ausfindig zu machen, gibt es eine allgemeine Richtlinie:

Rahmengrösse	Von	Bis
6 Zoll	1500 mAh	2200 mAh
5 Zoll	1300 mAh	1800 mAh
4 Zoll	850 mAh	1300 mAh
3 Zoll	650 mAh	1000 mAh

(Liang, OscarLiang, 2021)

6.4.5.2 Errechnung C-Rating

Mit nachfolgender Formel kann das C-Rating ermittelt werden. Dabei gilt zu beachten:
1000mAh = 1Ah:

$$\frac{\frac{\text{Maximale Gesamtstromaufnahme (A)}}{\text{Batteriekapazität (Ah)}}}{2} = CRating$$

(Liang, OscarLiang, 2021)

Für meine Drohne werde ich eine Batteriekapazität von 1550 mAh verwenden. Dies ergibt folgende Rechnung:

$$\frac{\frac{172.92A}{1.55Ah}}{2} = 55.75 \text{ C Rating}$$

Dies zufolge benötige ich eine Batterie mit 1550mAh und einem C-Rating von mindestens 55C.

6.4.5.3 Einfluss Flugstil

Der Flugstil beeinflusst die Wahl des Akkus. Eine höhere C-Bewertung muss eingeplant werden, wenn man konstant mit mehr als 50% Schubkraft fliegen möchte. (Liang, OscarLiang, 2021)

6.5 Propeller

In diesem Kapitel werden die Propeller näher angeschaut.

6.5.1 Propellerrichtungen

An jedem Quadrocopter gibt es zwei Motoren, welche im Uhrzeigersinn (CW) und zwei gegen den Uhrzeigersinn (CCW) drehen. Daher sind passende Propeller erforderlich, um genügend Schub zu erzeugen. In einem Packet befinden sich diesbezüglich zwei mal zwei unterschiedliche Propeller. Die Drehrichtung des Propellers kann bestimmt werden, indem die Vorderkante ermittelt wird. Der Propeller sollte sich zur höheren Vorderkante drehen, um Abwärtschub zu erzeugen. (Liang, OscarLiang, 2020)

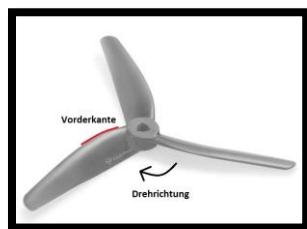


Abbildung 22 Propeller Drehrichtung (<https://www.dronefactory.ch/produkt/azure-vanover-limited-edition-prop-iron-grey/>)

6.5.2 Propeller Grösse und Steigung

Propeller erzeugen Schub durch das Bewegen von Luft. Wird ein Propeller schneller gedreht, kann er mehr Luft bewegen und damit mehr Schub erzeugen. Durch das Vergrössern der Länge oder der Steigung des Propellers bei gleicher Umdrehungsgeschwindigkeit kann mehr Schub erzeugt werden. Dies führt jedoch zu einer höheren Leistungsaufnahme. (Liang, OscarLiang, 2020)

6.5.3 Anzahl der Klingen

Grundsätzlich wird durch das Hinzufügen von mehr Klingen mehr Oberfläche hinzugefügt und daher mehr Schub auf Kosten einer höheren Stromaufnahme und eines höheren Widerstands erzeugt. Je mehr Klingen, desto ineffizienter. (Liang, OscarLiang, 2020)

6.5.4 Propellerspezifikationen

Es gibt zwei Arten von Formaten, die Hersteller verwenden für die Propeller Spezifikationen:

$$L * P * B \text{ oder } LLPP * B$$

- L- Länge
- P – Steigung
- B – Anzahl der Blätter

(Liang, OscarLiang, 2020)

6.6 RC Sender & Empfänger

Der RC-Sender sendet ein Radiosignal an den Empfänger. Dieser wandelt das erhaltene Signal um zu einem PWM-Signal und leitet es via den einzelnen Channels weiter.

6.6.1 Rechtslage

In der Schweiz stehen exklusiv für Flugmodellfernsteuerungen 23 Kanäle im Bereich 35.000 MHz bis 35.220 MHz zur Verfügung. Der Frequenzbereich von 40.715 MHz bis 40.985 MHz darf ausschliesslich nur für bodengebundene Modelle verwendet werden. Beim Kauf muss darauf geachtet werden. (Bundesamt für Kommunikation, 2021)

6.6.2 Channels

Die beiden Joysticks des Senders senden für die Horizontale- und Vertikale-Achse jeweils ein Radio Signal an einen dedizierten Kanal [CH1:4] des Empfängers.



Abbildung 23 RC-Empfänger (<https://www.conrad.ch/de/p/reely-hr-4-hand-fernsteuerung-2-4-ghz-anzahl-kanaele-4-inkl-empfaenger-1310036.html>)

6.7 Trägheitsmessungseinheit

Die Trägheitsmessungseinheit (IMU) MPU-6050 besteht unter anderem aus einem Beschleunigungsmesser und einem Gyroskop. Mit der Kombination dieser beiden Messinstrumente lässt sich die räumliche Befindlichkeit bestimmen. Sie ist essenziell für das autonome Schweben.

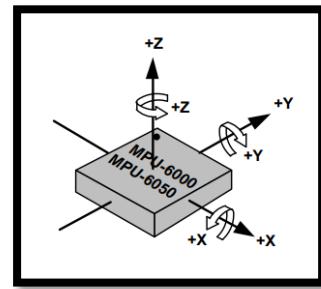


Abbildung 24 MPU-6050 Achsenbeschriftung (InvenSense.inc, 2013)

6.7.1 Beschleunigungsmesser

Der Beschleunigungsmesser misst die Beschleunigung (G-Kraft) von den drei Achsen X, Y und Z. Die Werte der jeweiligen Achse werden als eine 16Bit grosse Zahl gespeichert. Der MPU-6050 arbeitet nur mit 8Bit Registern und diesbezüglich wird jeder Beschleunigungs-wert in zwei Registern gespeichert, einem Low- und einem High-Byte. Der maximal gemes-sene Beschleunigungswert von 2^{16} Bit wäre 65'536. Im Datenblatt wird definiert, dass ein Wert von $4096 = 1\text{g-Kraft}$ entspricht, wenn der Messbereich von +/- 8g eingestellt wurde. Der zu messende Bereich kann definiert werden bis zu maximal 16g. Je höher der mögliche Messbereich ist, desto ungenauer wird die Messung.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY Full-Scale Range	AFS_SEL=0 AFS_SEL=1 AFS_SEL=2 AFS_SEL=3		± 2 ± 4 ± 8 ± 16		g g g g	
ADC Word Length Sensitivity Scale Factor	Output in two's complement format AFS_SEL=0 AFS_SEL=1 AFS_SEL=2		16 16,384 8,192 4,096		bits LSB/g LSB/g LSB/g	

Abbildung 25 MPU-6050 Datenblatt Beschleunigungsmesser Sensibilität (InvenSense.inc, 2013)

6.7.2 Gyroskop

Das Gyroskop misst die Winkelgeschwindigkeit. Die Winkelgeschwindigkeit gibt Auskunft über die Anzahl Grad Bewegung pro Sekunde für jede Achse. Im Datenblatt wird definiert, dass ein gemessener Wert von $65.5 = 1^\circ/\text{s}$ entspricht, wenn die maximal zu messende Gradänderung auf $500^\circ/\text{s}$ eingestellt wurde.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY Full-Scale Range	FS_SEL=0 FS_SEL=1 FS_SEL=2 FS_SEL=3		± 250 ± 500 ± 1000 ± 2000		%/s %/s %/s %/s	
Gyroscope ADC Word Length Sensitivity Scale Factor	FS_SEL=0 FS_SEL=1 FS_SEL=2 FS_SEL=3		16 131 65.5 32.8 16.4		bits LSB/(%/s) LSB/(%/s) LSB/(%/s) LSB/(%/s)	

Abbildung 26 MPU-6050 Datenblatt Gyroskop Sensibilität (InvenSense.inc, 2013)

6.8 Arduino

Arduino ist eine Open-Source-Plattform, die zum Erstellen von Elektronikprojekten verwendet wird. Arduino besteht sowohl aus einer physischen programmierbaren Platine und einer Software, die auf dem Computer ausgeführt wird zum Schreiben sowie Hochladen von Computercode auf die physische Platine verwendet wird. (Sparkfun, 2021)

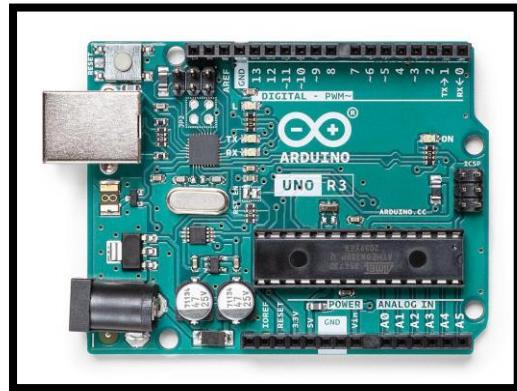


Abbildung 27 Arduino Uno (<https://store.arduino.cc/products/arduino-uno-rev3/>)

6.8.1 Arduino Auswahl

Mittlerweile gibt es viele verschiedene Arduino-Boards. Einige mit bestimmten und andere für allgemeine Aufgabengebiete. Ich habe mich bei diesem Projekt für einen Standard Arduino Uno entschieden. Dieser beinhaltet alle nötigen Schnittstellen und die einfache Handhabung mit dem Verbinden anderer Komponenten mit einem Stecker System ermöglicht ein viel einfacheres Umstecken bei den Prototypen oder falls ein Kabel fälschlicherweise vertauscht wurde als bei anderen Boards, welche gelötet werden müssen wie beispielsweise beim Arduino Nano.

6.8.2 Arduino Uno Schnittstellen

Im nachfolgenden Bild sind alle Arduino Uno Pins ausführlich beschriftet:

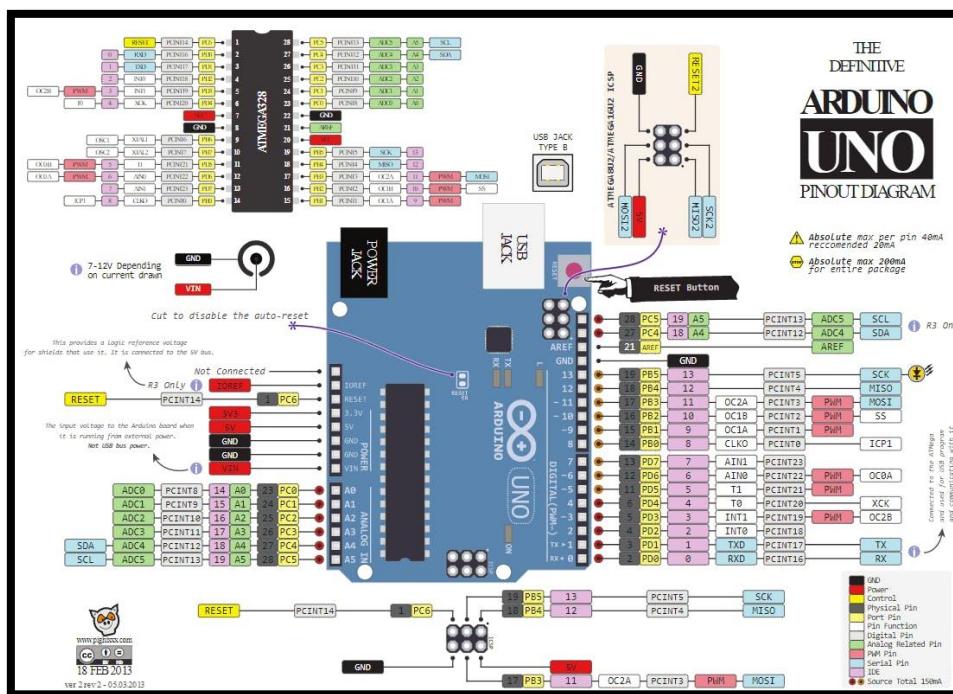


Abbildung 28 Arduino Uno Pinbelegung (<https://www.circuito.io/blog/arduino-uno-pinout/>)

6.9 Verwendete Komponenten

Nachfolgend eine detaillierte Liste mit den in diesem Projekt verwendeten Komponenten:

Bild	Komponente	Name	Anzahl	Einzel-preis CHF	Quelle
	Rahmen	Haoye X1	1x	35.90	https://optimum-racing.ch/de/frame/1361-haoye-rc-x1-5-inch-free-style-fpv-frame-kit.html
	Motor	TBS MasterPilot PRO 2750KV	4x	19.90	https://www.dronefactory.ch/produkt/tbs-masterpilot-pro-2750kv/
	ESC	Skywalker 40A	4x	17.95	https://www.conrad.ch/de/p/hobbywing-skywalker-40a-flugmodell-brushless-flugregler-be-lastbarkeit-max-a-55-a-2108342.html
	Propeller	Ethix S5	1x	3.90	https://www.dronefactory.ch/produkt/ethix-s5-props/
	Batterie	DroneArt 11.1V 1550 mAh 75C	1x	18.95	https://www.conrad.ch/de/p/droneart-modellbau-akkupack-lipo-11-1-v-1550-mah-zellen-zahl-3-75-c-softcase-xt60-1784861.html
	Flugkontroller	Arduino Uno Rev3	1x	25.20	https://www.digitec.ch/en/s1/product/arduino-uno-atmega328-development-boards-kits-5764177
	RC-Empfänger & Sender	Reely HT-4	1x	49.95	https://www.conrad.ch/de/p/reely-ht-4-handfernsteuerung-2-4-ghz-anzahl-kanaele-4-inkl-empfaenger-1310036.html
	LiPo Ladegerät	Absima Cube 2.0	1x	33.95	https://www.conrad.ch/de/p/absima-cube-2-0-modellbau-ladegeraet-5000-ma-liion-lipo-nicd-nimh-2238976.html
	Ladekabel	Reely Ladekabel	2x	6.95	https://www.conrad.ch/de/p/reely-ladekabel-1x-bananenstecker-4-mm-1x-xt60-stecker-30-00-cm-re-6799038-2266346.html

	IMU	GY MPU-6050	1x	29.90	https://www.play-zone.ch/de/mpu-6050-accelerometer-gyro.html
	Jumperkabel	Play-Zone M/F 24AWG	1x	11.90	https://www.play-zone.ch/de/jumperkabel-verbindungsleitungskabel-20cm-m-f-40-stk-24awg.html
	Verbindungs-kabel	Hook-up Wire (Draht)	1x	25.90	https://www.play-zone.ch/de/verbindungs-kabel-hook-up-wire-sortiment-draht.html
	Schrumpf-schläuche	Delock Box 100-t	1x	5.10	https://www.brack.ch/de-lock-schrumpfschlauch-100-teilig-sortimentskoffer-1064038

6.10 Kommunikationsprotokoll PWM

«Pulse Width Modulation» (PWM) funktioniert durch das Pulsieren von Gleichstrom und Variieren der Zeit, die jeder Impuls an bleibt. PWM ist ein digitales Signal, es gibt zwei Zustände an und aus. In Arduino Code wird dies als «HIGH» und «LOW» definiert. Die Zeitmessung des Signals wird in Mikrosekunden gemessen. (Brown N., 2021)

Ein Signal von Beispielsweise 1400us sieht folglich aus:

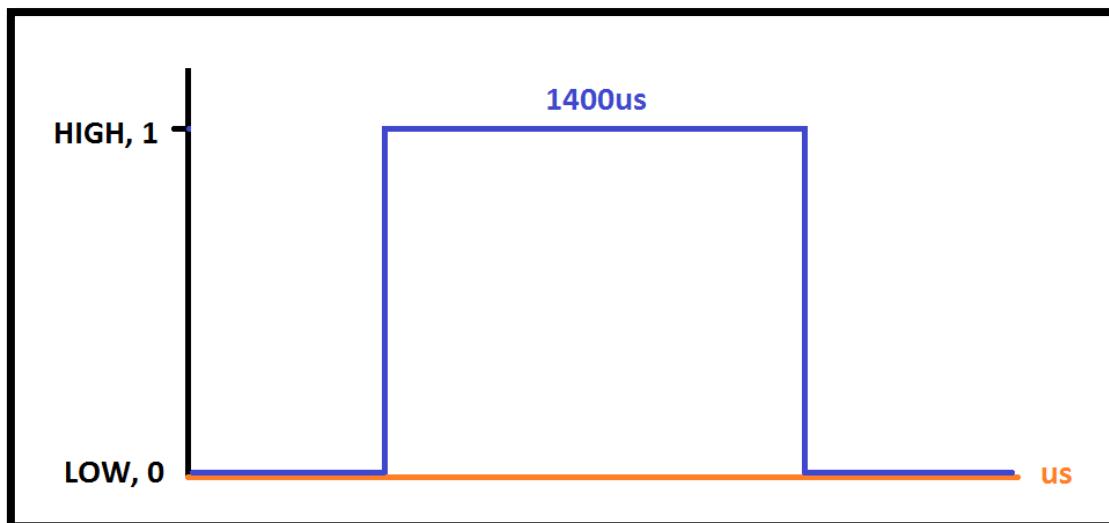


Abbildung 29 PWM Signal (O'Connor)

7 SCHALTPLAN

Damit die einzelnen Komponenten miteinander kommunizieren können, müssen diese korrekt miteinander verbunden werden. Dies wird mit Löten und Drahten ermöglicht. Für die Verbindungen habe ich folgenden Schaltplan aus den zur Verfügung gestellten Datenblättern der einzelnen Komponenten erstellt:

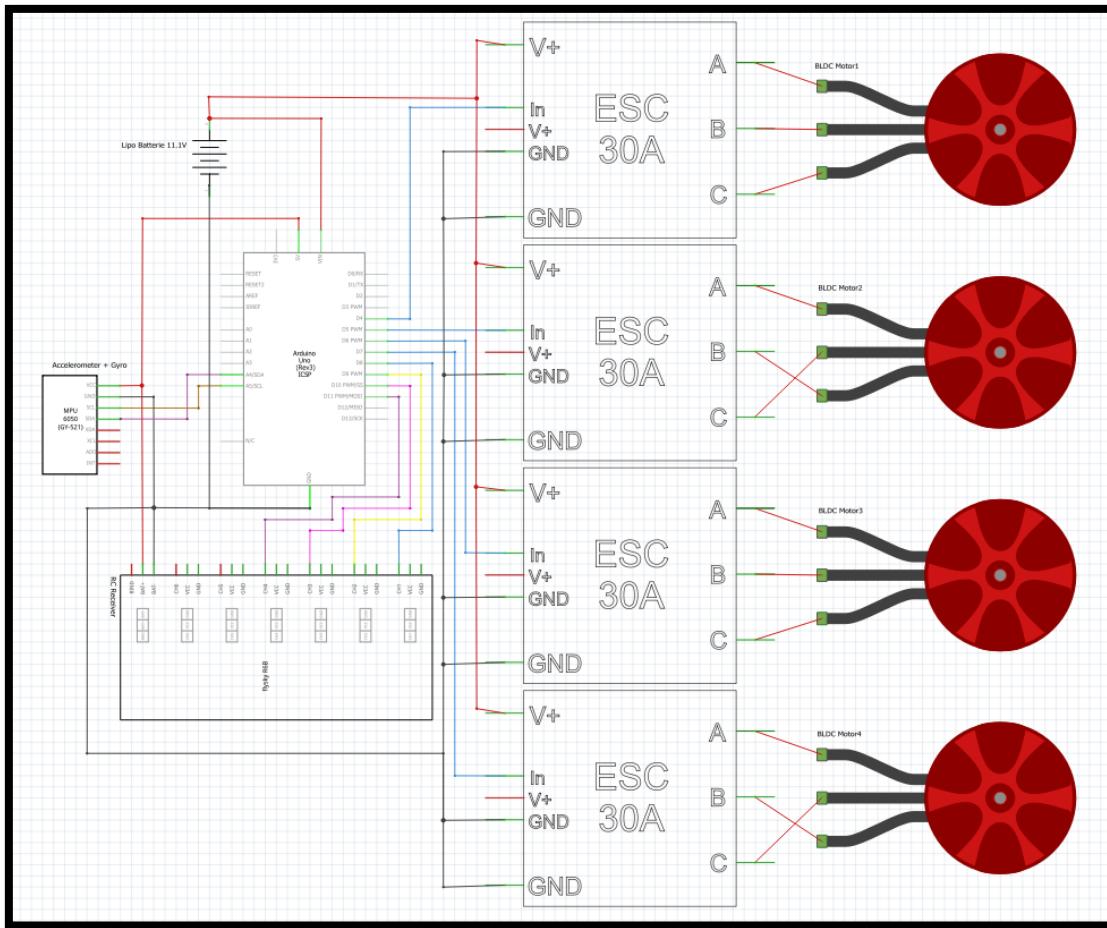


Abbildung 30 Schaltplan (O'Connor)

Arduino Drohne selber bauen

Ein detaillierter Ausschnitt der wichtigsten Schnittstellen:

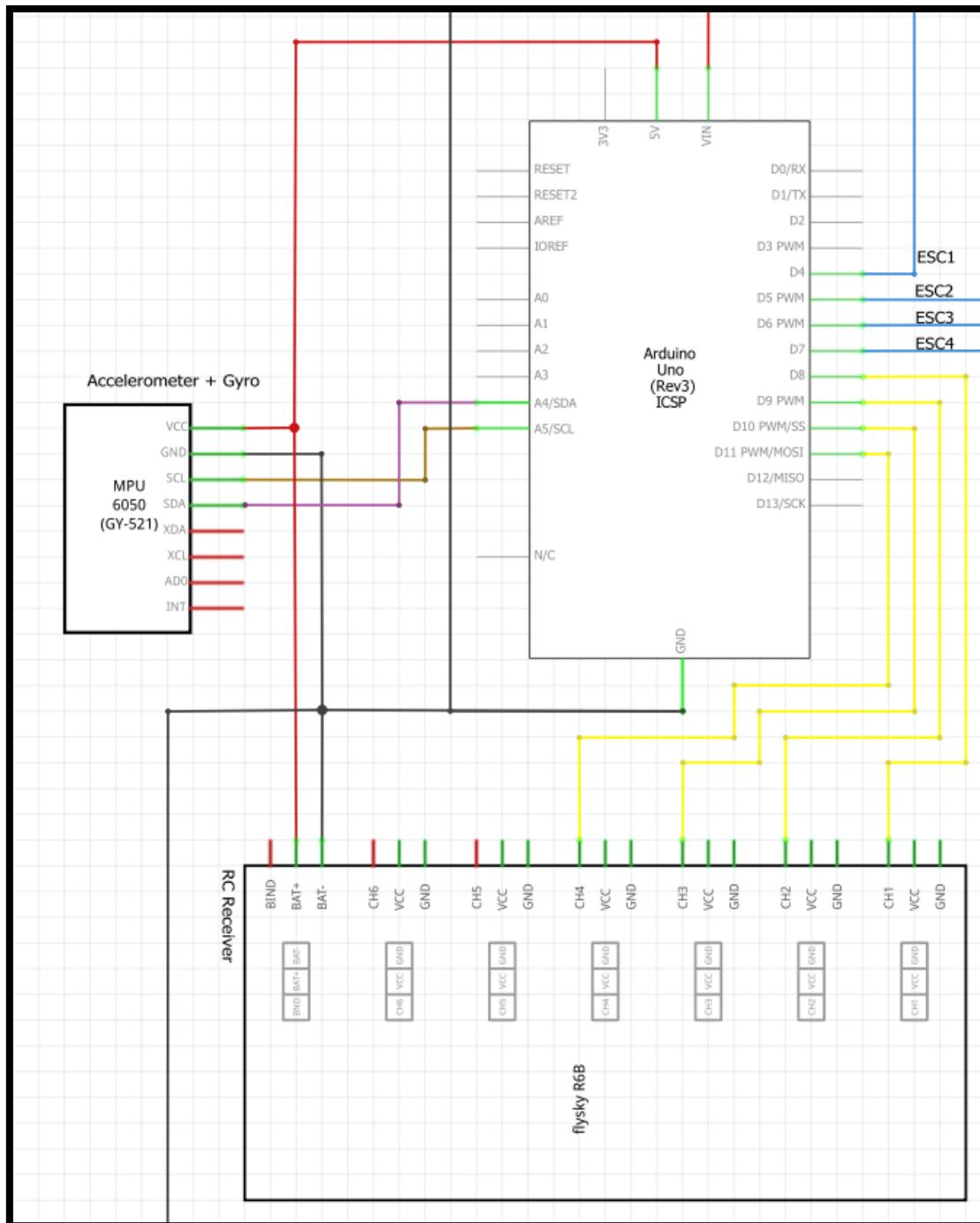


Abbildung 31 Schaltplan Zoom (O'Connor)

8 REALISIERUNG

Das nachfolgende Kapitel behandelt die Realisierung des Gesamten.

8.1 Prototypen

Die einzelnen Prototypen dienen der Unterteilung des Gesamten in kleinere Subsysteme mit dem Ziel der Erforschung, wie das Subsystem funktioniert und ob sich frühzeitig Fehler herauskristallisieren.

8.1.1 Prototyp 1: RC Sender und Empfänger

Anhand dieses Prototyps soll festgestellt werden, ob der RC-Sender und Empfänger bereits gekoppelt sind und untereinander kommunizieren können, wie die einzelnen Kabel vom Empfänger mit dem Arduino verbunden werden müssen und wie die Daten eingelesen werden können.

8.1.1.1 Kopplung

In meinem Fall waren die beiden Komponenten bereits miteinander verkoppelt. Sollte dies nicht der Fall sein, kann wie dies gemacht wird, aus der entsprechenden Anleitung entnommen werden.

8.1.1.2 Empfängeranschluss

Der Empfänger hat für den Strom (angeschrieben als BAT) sowie für jeden der total sechs Kanäle drei Anschlussmöglichkeiten. Am Anschluss «BAT» wird der mittlere Port mit dem Arduino 5V verbunden sowie der rechte Port mit GRND.

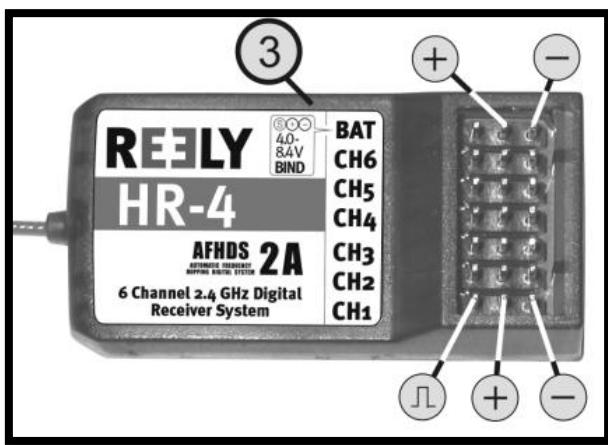


Abbildung 32 RC-Empfänger (Reely HR-4 Manual)

Für dieses Projekt werden lediglich die Kanäle 1-4 benötigt. Bei diesen wird jeweils nur das Signalkabel, der linke Port, mit den Pins D8-11 auf dem Arduino verbunden wie im Schaltplan dargestellt. Im nachfolgenden Bild ist der Kanal 1 mit Pin D8 verbunden:

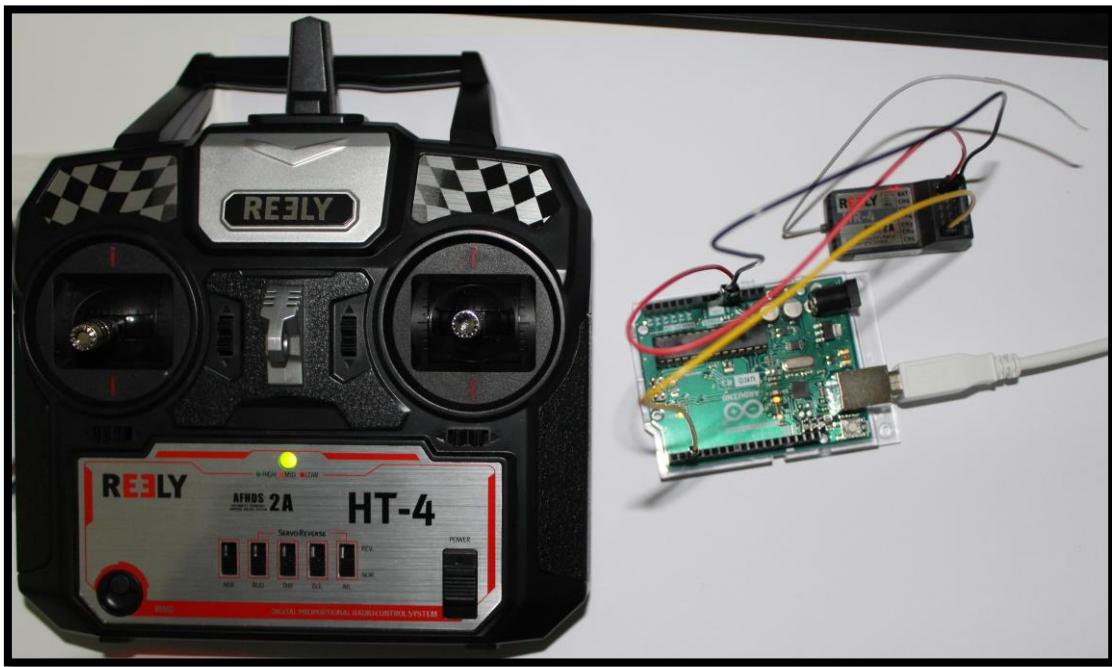


Abbildung 33 Prototyp 1 (O'Connor)

8.1.1.3 Einlesen PWM Signal

Durch meine intensive Recherche konnte ich mehrere Möglichkeiten ausfindig machen, wie der Arduino die Daten Einlesen kann:

1. Servo Input Library

Die Servo Input Library verwendet «Timer Interrupts». Diese ermöglichen es, die normale Abfolge von Ereignissen, die in der Main loop()-Funktion in zeitlich genau festgelegten Intervallen stattfinden, vorübergehend zu unterbrechen. Dadurch muss nicht auf das PWM Signal gewartet werden, bis jede einzelne Übertragung der vier Kanäle abgeschlossen wurde. Diese Funktionalität ähnelt dem sogenannten Multithreading und verkürzt die benötigte Zeit für einen Durchgang der Main loop() massiv. Voraussetzung für die Verwendung dieser Library ist es, dass alle Signale an einem Timer-Interrupt Pin vom Arduino angeschlossen werden. (Poole, 2012)

Der Arduino Uno besitzt leider jedoch nur zwei der vier benötigten Anschlüsse und der Arduino Uno müsste mit einem anderen Arduino Mikrochip ersetzt werden.

2. Arduino PulseIn() Funktion

Bei der Standard Arduino PulseIn() Funktion wird bei jedem Durchgang der Main loop() jedem einzelnen Kanal des Empfängers nachfolgend das Signal eingelesen. Bei RC-Empfängern sowie bei ESCs ist ein Signal von $1000\mu\text{s}$ bis $2000\mu\text{s}$ üblich. Somit wäre die Main loop() mindestens 8ms lang und im schlimmsten Szenario 16ms. (Arduino, 2021)

3. Pin-Change-Interrupt

Im Gegensatz zum internen «Timer Interrupts» wird bei einem Pin-Change-Interrupt der Durchgang der Main loop() unterbrochen, sobald eine Änderung am Wert eines vordefinier-ten Pins von aussen entsteht.

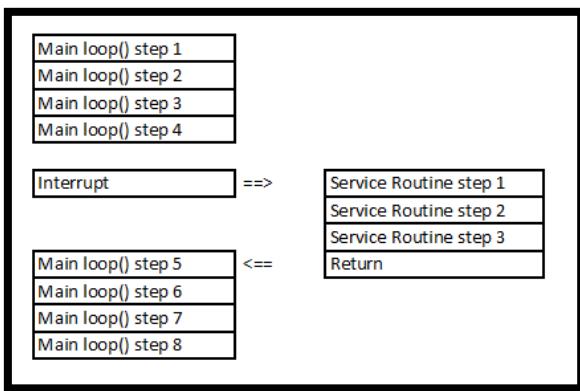


Abbildung 34 Interrupt (O'Connor)

Beim Arduino Uno können fast alle Pins dafür verwendet werden. Die einzige Einschränkung in Bezug auf diese Methode ist, jeweils 6-8 Pins teilen sich die gleiche Service-Routine:

- ISR(PCINT0_vect) pin change interrupt für D8 bis D13
- ISR(PCINT1_vect) pin change interrupt für A0 bis A5
- ISR(PCINT2_vect) pin change interrupt für D0 bis D7

Wird ein Pin-Change-Interrupt von D8 oder von D9 ausgelöst, wird die gleiche Service Routine aufgerufen und innerhalb dieser Routine muss dann ausfindig gemacht werden, welcher Pin geändert wurde. Dies bedeutet, der alte Stand aller Pins der gleichen Gruppierung muss gespeichert werden und beim Ausführen der Service Routine mit dem aktuellen Stand verglichen werden, um den veränderten Pin ausfindig machen zu können. (Arduino, 2021)

Mein Interview Partner hat Folgendes zum Problem der Verzögerung der Main loop() von der PulseIn() Funktion geäussert: «It depends on what you are looking at. In terms of commercial drones like we have here, it's not a problem because usually what you are dealing with by just looking at a simple multirotor: you have big props, so these here are around 25-30 inches in size with a low KV motor that are efficient but turn relatively slowly. Which means, every time you give it a change, a command to speed up or slow down, you got to deal with inertia. The time it needs to respond is much longer than the actual control time. Therefore, getting an even faster control will not even do you much. So, where it does play a role, is when you are flying on tiny FPV drones. The little ones with 2k - 3k RPM motors, they are extremely agile and there, you notice the difference. There it counts. [...].».

Aufgrund dessen habe ich mich für die 3. Option entschieden und werde eine Pin-Change-Interrupt Service-Routine implementieren.

8.1.1.4 Pin-Change-Interrupt

Der Pin-Change-Interrupt PCI0 wird ausgelöst, wenn ein aktiver PCINT[7:0]-Pin umschaltet. Auf dem Arduino sind das die dazugehörigen Pins 8-13, sowie 20 und 21. (Microchip Technology Inc, 2018)

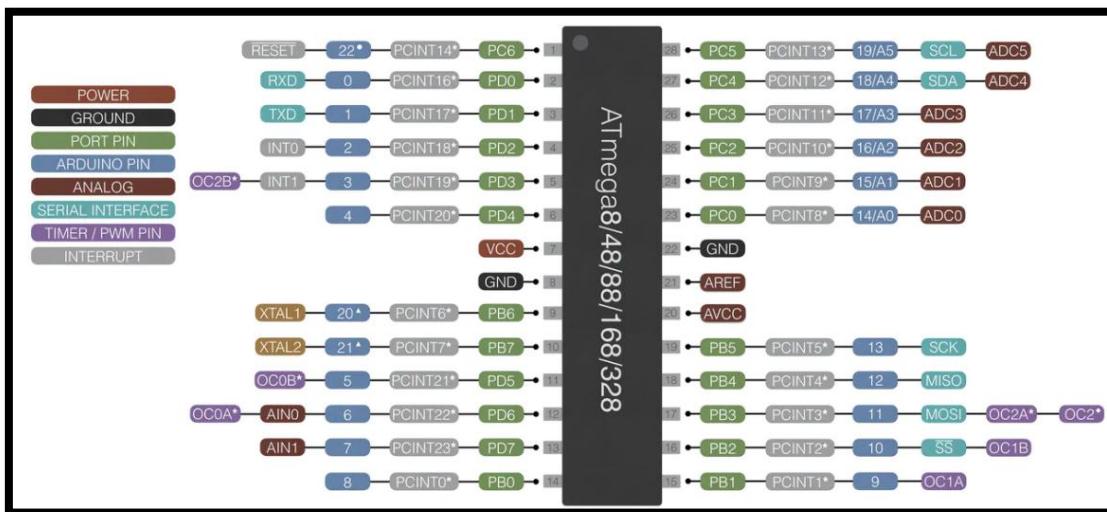


Abbildung 35 Arduino Uno Pinbelegung (<https://create.arduino.cc/projecthub/Alojz/arduino-pocket-game-console-a-maze-game-63c225>)

Um den Pin-Change-Interrupt einzuschalten für alle Pins vom PCIE0, muss dieser im Pin-Change-Interrupt-Controller Register aktiviert werden. Danach müssen alle Pins, welche ein Interrupt erhalten können, zusätzlich noch einzeln aktiviert werden im Pin-Change-Mask-Register 0. (Microchip Technology Inc, 2018)

PCMSK0 – Pin Change Mask Register 0								PCMSK0	
Bit (0x6B)	7	6	5	4	3	2	1	0	PCMSK0
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 36 PCMSK0 Register (Microchip Technology Inc, 2018)

Im Arduino Source-Code sieht dies folgendermassen aus:

```
PCICR |= 0b00000001;      // PCIE0 for PCMSK0
PCMSK0 |= 0b00001111;    // PCINT 0:3 => Input 8:11
```

Abbildung 37 PCIE0 Pin-Change-Interrupt (O'Connor)

Die Service Routine, welche aufgerufen wird bei einem Pin-Change-Interrupt, wird folgendemassen für den PCIE0 definiert im Arduino Code:

```
ISR(PCINT0_vect) {  
    ...  
}
```

Abbildung 38 PCINT0 Service Routine (O'Connor)

Alle Pins vom PCIE0 Register können durch die Input Port B Adresse ausgelesen werden. Die Input Port B Adresse ist ebenfalls im ATmega328 Pinout-Diagramm beschriftet. Mit der entsprechenden Bit-Stelle aus dem PINB Array kann mit Bit Operationen der aktuelle Wert des Pins ermittelt werden: (Microchip Technology Inc, 2018)

PINB – The Port B Input Pins Address ⁽¹⁾								
Bit	7	6	5	4	3	2	1	0
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Read/Write	R/W							
Initial Value	N/A							

Abbildung 39 PINB Adressen (Microchip Technology Inc, 2018)

Um nachzusehen ob PINB0 (Pin 8) ein HIGH Signal hat, kann folgende Syntax bezüglich verwendet werden:

```
if(PINB & B00000001) {  
}
```

Abbildung 40 Pin 8 HIGH? (O'Connor)

Dies entspricht dem Pin 8 auf dem Arduino. Für den Pin 9 (PINB1) muss nun lediglich das entsprechende Bit um eine Stelle nach links verschoben werden:

```
if(PINB & B00000010) { ... }
```

Abbildung 41 Pin 9 HIGH? (O'Connor)

Dies wird wiederholt für alle Kanäle des Empfängers angeschlossenen Pins:

```
ISR(PCINT0_vect) {  
    //CH 1  
    if(PINB & B00000001){      //input 8 high?  
    }  
    //CH 2  
    if(PINB & B00000010 ){     //input 9 high?  
    }  
    //CH 3  
    if(PINB & B00000100 ){    //input 10 high?  
    }  
    //CH 4  
    if(PINB & B00001000 ){   //input 11 high?  
    }  
}
```

Abbildung 42 Aktive Input-Pins ermitteln (O'Connor)

Die vom Arduino dafür vorgefertigte Funktion wäre: «digitalRead(pin)». Diese Funktion benötigt aber mehr Leistung des Arduinos als die Low-Level Bit-Operationen. Die Interrupt Service Routine sollte so klein wie möglich bleiben.

Um die Pulslänge des PWM Signals ermitteln zu können, muss für jeden Channel der letzte Zustand sowie die Zeit des letzten Wechsels gespeichert werden. Dafür wird ein Array verwendet. Der Index des Arrays gibt Auskunft über die Kanalnummer beginnend mit für Pin 8 = Index 0, Pin 9 = Index 1 usw. Da dies in einer Interrupt-Service-Routine passiert, muss zusätzlich «volatile» angegeben werden, damit der Compiler die Werte nicht vom Speicher sondern vom Memory holt.

```
volatile bool PCI_channel_state[4] = {false, false, false, false};  
volatile int  PWM_channel_input[4];  
volatile long PCI_current_time, PCI_channel_timer[4];
```

Abbildung 43 Notwendige Interrupt-Deklarationen (O'Connor)

Im nachfolgenden Beispiel wird das PWM Signal für den Kanal 1 des Empfängers auf den aktuellen Zustand überprüft.

Wurde der Pin von LOW auf HIGH gewechselt, wird dies im Array PCI_channel_state[0] gespeichert sowie die aktuelle Systemzeit im Array PCI_channel_timer[0] in μs .

Der Wechsel von HIGH zu LOW kann erkannt werden, wenn der Zustand vom Pin LOW ist und der zuletzt gespeicherte Channel-State auf HIGH. Das PWM Signal ist somit abgeschlossen und die verstrichene Zeit kann errechnet werden mit der Startzeit des Wechsels von LOW auf HIGH und der aktuellen Systemzeit:

```

/* 
 * PinChangeInterrupt
 */
ISR(PCINT0_vect) {
    PCI_current_time = micros();

    // CH 1
    if(PINB & B00000001) {
        if(!PCI_channel_state[0]) {
            PCI_channel_state[0] = 1;
            PCI_channel_timer[0] = PCI_current_time;
        }
    }
    else if(PCI_channel_state[0]) {
        PCI_channel_state[0] = 0;
        PWM_channel_input[0] = PCI_current_time - PCI_channel_timer[0]; // Calculate Time (PWM)
    }
    // CH 2
    // CH 3
    // CH 4
}

```

Abbildung 44 PWM-Signalmessung (O'Connor)

8.1.1.5 Ausgabe des PWM Signals im Serien-Monitor

Als ersten Test werden die übermittelten Werte vom RC-Empfänger auf dem Serien-Monitor ausgegeben. Dies ist wie eine Konsolenausgabe auf dem Entwicklungsrechner. Dazu muss dieser initialisiert werden in der setup() Funktion:

```

void setup() {
    Serial.begin(9600);
    ...
}

```

Abbildung 45 Serienmonitor Initialisierung (O'Connor)

In der Main loop() kann dann einfach jeweils der gespeicherte Wert aus dem Array ausgelesen werden:

```

void loop() {

    Serial.print(receiver_input[0]);
    Serial.print(" | ");
    Serial.print(receiver_input[1]);
    Serial.print(" | ");
    Serial.print(receiver_input[2]);
    Serial.print(" | ");
    Serial.println(receiver_input[3]);
}

```

Serien-Monitor			
1500	1500	1008	1500
1504	1500	1008	1504
1500	1496	1008	1500
1504	1500	1012	1504
1504	1500	1012	1504
1500	1500	1008	1500
1504	1500	1008	1504
1500	1496	1008	1500
1504	1500	1012	1504
1504	1500	1012	1504

Abbildung 46 Ausgabe vom PWM Signal des Empfängers auf dem Serienmonitor (O'Connor)

Die Zahlen haben zwischen 1000µs und 2000µs variiert, je nach dem, in welcher Position sich der entsprechende Joystick befunden hat. Die Zahl 1500µs bedeutet, dass dieser Joystick sich aktuell in einer zentralen Position befindet. Auch ersichtlich ist es, dass die Zahlen in 4er Sprüngen wechseln. Der Arduino Uno hat zu wenig Rechenleistung und kann nur ein Vielfaches von 4 im Mikrosekundenbereich ermitteln.

Durch das ausgiebige Testen der einzelnen Joysticks konnte ich ausfindig machen, welcher für welchen Channel verantwortlich ist und konnte dies wie folgt festhalten:

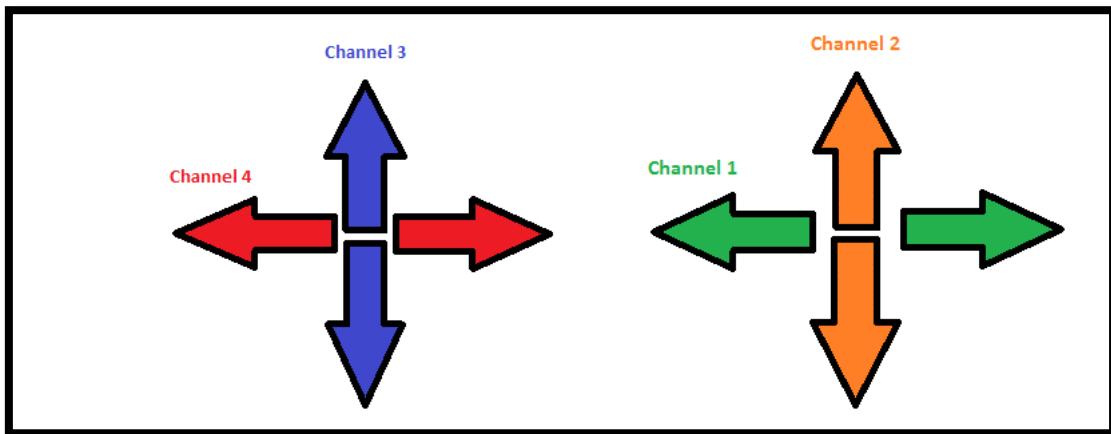


Abbildung 47 RC-Kanalzuweisung (O'Connor)

8.1.2 Prototyp 2: Elektronischer Geschwindigkeitsregler & Motor

Um die Ansteuerung eines elektronischen Geschwindigkeitsreglers (ESC) ausfindig zu machen, habe ich einen Motor sowie einen ESC provisorisch mit dem Arduino und einer Batterie verbunden.

Das Ziel dieses Prototyps ist es, die Geschwindigkeit des Motors mittels des RC-Senders via dem Arduino steuern zu können. Ein ESC erwartet ebenfalls wie der RC-Empfänger ein PWM Signal zwischen 1000µs - 2000µs. Durch die Erkenntnisse und Erarbeitung des ersten Prototypen kann ausgesagt werden, dass lediglich die Ausgabe anstatt an den Serien-Monitor, an den ESC weiter gegeben werden soll in der Main loop() Funktion.

Dasselbe Problem wie bei der PulseIn() Funktion vom RC-Empfänger betrifft auch die Ansteuerung der vier Geschwindigkeitsregler. Wird jeder ESC nacheinander angesteuert, fallen bis zu 8ms Verzögerung an. Die Lösung zu diesem Problem ist eine eigene Methode welche gleich alle vier ESCs mit dem entsprechenden PWM-Signal anspricht zu implementieren. Dadurch kann die maximale Zeit auf lediglich 2ms reduziert werden.

Bereits in der Service Routine vom Pin-Change-Interrupt wurden Bit-Operationen verwendet. Diese finden hier auch ihren Platz. Die nötigen Pins 4-7 können vom PORTD (PD) angesprochen werden mit den vier höchstwertigen Bit: (Microchip Technology Inc, 2018)

PORTD – The Port D Data Register								
Bit	7	6	5	4	3	2	1	0
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
ReadWrite	R/W							
Initial Value	0	0	0	0	0	0	0	0
								PORTD

Abbildung 48 PORTD Register (Microchip Technology Inc, 2018)

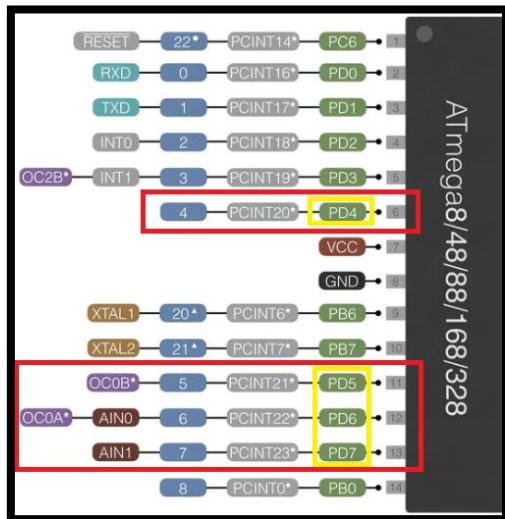


Abbildung 49 Arduino Pinbelegung (<https://create.arduino.cc/projecthub/Alojz/arduino-pocket-game-console-a-maze-maze-game-63c225>)

Die Pins 4:7 müssen, um als Ausgabepins funktionieren zu können, in der setup() Funktion konfiguriert werden:

```
void setup() {  
    DDRD |= B11110000; //Configure digital pins 4, 5, 6 & 7 as output.  
    ...  
}
```

Abbildung 50 Pins 4:7 als Ausgabepins definieren (O'Connor)

Als Erstes in der Main loop() muss die Zielzeit ermittelt werden. Dafür kann die aktuelle Systemuhrzeit in μs mit dem Gas- (Throttle-) Wert vom RC-Empfänger Channel 3 addiert werden:

```
current_time = micros();  
  
ESC_PWM_target_time[0] = PWM_channel_input[2] + current_time;  
ESC_PWM_target_time[1] = PWM_channel_input[2] + current_time;  
ESC_PWM_target_time[2] = PWM_channel_input[2] + current_time;  
ESC_PWM_target_time[3] = PWM_channel_input[2] + current_time;
```

Abbildung 51 Zielzeitermittlung (O'Connor)

Anschliessend wird das HIGH Signal auf allen ESCs ausgesendet:

```
PORTD |= B11110000; // SET HIGH on all ESC
```

Abbildung 52 PORTD HIGH-Signal für alle Motoren (O'Connor)

Um das High Signal wieder um zu kehren, wird die aktuelle Systemuhrzeit mit der Zielzeit für jeden ESC verglichen. Ist dieser Wert identisch, dann wird nur beim entsprechendem ESC das Signal auf LOW gesetzt. Dies passiert so lange bis alle vier höchstwertigen Bit im PORTD Register auf 0 gesetzt sind, oder anders gesagt, der dezimale Wert von PORTD kleiner ist wie 16:

```
while(PORTD >= 16) {  
  
    ESC_loop_time = micros();  
  
    if(ESC_loop_time >= ESC_PWM_target_time[0]) PORTD &= B11101111;  
    if(ESC_loop_time >= ESC_PWM_target_time[1]) PORTD &= B11011111;  
    if(ESC_loop_time >= ESC_PWM_target_time[2]) PORTD &= B10111111;  
    if(ESC_loop_time >= ESC_PWM_target_time[3]) PORTD &= B01111111;  
}
```

Abbildung 53 Beendung des PWM Signals (O'Connor)

Die ESCs erwarten eine Signalfrequenz von 250mHz. Um diese Frequenz stabil beizubehalten wird in der Main loop() eine while()-Schlaufe hinzugefügt, bis die 4000µs vorbei sind, bevor sie sich wiederholt:

```
void loop() {  
  
    while(micros() <= current_time + 4000);
```

Abbildung 54 250Hz Timer (O'Connor)

8.1.2.1 ESC Kalibrierung

Bevor die ESCs in Betrieb genommen werden können, müssen diese kalibriert werden. Dazu wird vor der erstmaligen Stromzufuhr das Gaspedal vom RC-Sender auf maximale Stufe eingestellt. Danach kann der ESC an die Batterie angeschlossen werden. Nach drei kurzen BEEP Tönen wird das Gaspedal in die unterste Position geführt und ein erneutes BEEP bestätigt die neue Konfiguration vom Hoch- und Tiefpunkt.



Abbildung 55 ESC Kalibrierung (O'Connor)

8.1.3 Prototyp 3: Trägheitsmessungseinheit (IMU)

Das Ziel dieses Prototyps ist, die Daten der IMU mit dem Arduino einzulesen, die Daten zu untersuchen und falls notwendig zu kalibrieren.

8.1.3.1 Setup

Für das Einlesen der Daten mit dem I2C-Protokoll wird die Library «Wire.h» benötigt. Diese ermöglicht die Kommunikation zwischen Arduino und IMU. In der `setup()`-Funktion muss diese initialisiert werden und der I2C Clock-Speed auf 400kHz festgelegt werden: (Arduino, 2021)

```
void setup() {  
    Wire.begin();  
    TWBR = 12;  
}
```

Abbildung 56 Wire.h Initialisierung (O'Connor)

Damit der MPU6050 verwendet werden kann, muss dieser auch initialisiert werden. Dieser verwendet standardmäßig die Adresse 0x68. Das PWR_MGMT_1 Register ist zuständig für den aktuellen Wach- oder Schlaf-Zustand. Um den IMU anzustellen, müssen alle Bits in diesem Register auf 0 geschalten werden: (InvenSense.inc, 2013)

PWR_MGMT_1									
Type: Read/Write									
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

Abbildung 57 PWR_MGMT_1 Register (InvenSense.inc, 2013)

```
void setup_mpu_6050_registers(){  
    Wire.beginTransmission(0x68);  
    Wire.write(0x6B);  
    Wire.write(0x00);  
    Wire.endTransmission();
```

Abbildung 58 PWM_MGMT_1 Register verändern (O'Connor)

Arduino Drohne selber bauen

Im zweiten Schritt wird im GYRO_CONFIG Register ein Messbereich von bis zu 500°/sec Messung des Gyroskops eingestellt: (InvenSense.inc, 2013)

GYRO_CONFIG									
Type: Read/Write									
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]	-	-	-	-

Abbildung 59 Gyroskop Register (InvenSense.inc, 2013)

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

Abbildung 60 Gyroskop Gradbereich (InvenSense.inc, 2013)

In Arduino Code übertragen:

```
Wire.beginTransmission(0x68);
Wire.write(0x1B);
Wire.write(0x08);
Wire.endTransmission();
```

Abbildung 61 Gyroskop Messbereich definieren (O'Connor)

Im dritten Schritt muss dem ACCEL_CONFIG Register ebenfalls mitgeteilt werden, dass die volle Breite von +/- 8G gemessen werden soll. (InvenSense.inc, 2013)

ACCEL_CONFIG									
Type: Read/Write									
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]	-	-	-	-

Abbildung 62 ACCEL Konfigurationsregister (InvenSense.inc, 2013)

```
Wire.beginTransmission(0x68);
Wire.write(0x1C);
Wire.write(0x10);
Wire.endTransmission();
```

Abbildung 63 Messbereich Beschleunigungsmesser (O'Connor)

Arduino Drohne selber bauen

Der MPU-6050 hat einen Tiefpassfilter, dieser kann das Rauschen vom Signal etwas herausfiltern. Der DLPF wird auf 44Hz gesetzt im Register 1A. (Buschbaum, 2015)

CONFIG									
Type: Read/Write									
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]	DLPF_CFG[2:0]				

Abbildung 64 DLPF Konfigurationsregister (InvenSense.inc, 2013)

DLPF_CFG	Accelerometer ($F_s = 1\text{kHz}$)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	F_s (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Abbildung 65 DLPF Bandbreite (InvenSense.inc, 2013)

In Arduino-Code übertragen:

```
Wire.beginTransmission(0x68);
Wire.write(0x1A);
Wire.write(0x03);
Wire.endTransmission();
```

Abbildung 66 DLPF definieren (O'Connor)

Das Setup ist somit abgeschlossen und der MPU-6050 ist betriebsbereit.

8.1.3.2 Einlesen

Die gemessenen Werte des Beschleunigungssensors und vom Gyroskop in der X-, Y- und Z-Achse sind jeweils 16Bit gross. Der Arduino und der MPU haben beide nur eine 8Bit Memory- sowie Registergrösse. Dies bedeutet, dass alle Zahlen in der Mitte halbiert und an jeweils zwei Speicherorten gespeichert werden. Von der Tabelle aus dem Datenblatt vom MPU6050 können die einzelnen Speicheradressen nachgeschlagen werden:

(InvenSense.inc, 2013)

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R								ACCEL_XOUT[15:8]
3C	60	ACCEL_XOUT_L	R								ACCEL_XOUT[7:0]
3D	61	ACCEL_YOUT_H	R								ACCEL_YOUT[15:8]
3E	62	ACCEL_YOUT_L	R								ACCEL_YOUT[7:0]
3F	63	ACCEL_ZOUT_H	R								ACCEL_ZOUT[15:8]
40	64	ACCEL_ZOUT_L	R								ACCEL_ZOUT[7:0]
41	65	TEMP_OUT_H	R								TEMP_OUT[15:8]
42	66	TEMP_OUT_L	R								TEMP_OUT[7:0]
43	67	GYRO_XOUT_H	R								GYRO_XOUT[15:8]
44	68	GYRO_XOUT_L	R								GYRO_XOUT[7:0]
45	69	GYRO_YOUT_H	R								GYRO_YOUT[15:8]
46	70	GYRO_YOUT_L	R								GYRO_YOUT[7:0]
47	71	GYRO_ZOUT_H	R								GYRO_ZOUT[15:8]
48	72	GYRO_ZOUT_L	R								GYRO_ZOUT[7:0]

Abbildung 67 Speicheradressen der Messwerte (InvenSense.inc, 2013)

Die benötigten Daten starten von Speicheradresse 0x3B bis 0x48. Mit der «Wire.h Library» kann der Lesestartpunkt gesetzt werden und anschliessend mit einer Request eine Anzahl Bytes eingelesen werden. Um alle Einträge einlesen zu können werden folglich alle 14 Zeilen der Reihe nach eingelesen: (Arduino, 2021)

```
void read_mpu_6050_data() {
    Wire.beginTransmission(0x68);
    Wire.write(0x3B);
    Wire.endTransmission();
    Wire.requestFrom(0x68, 14);
```

Abbildung 68 MPU-6050 Daten Einlesen (O'Connor)

Anschliessend müssen alle Bytes in der richtigen Reihenfolge der richtigen variable zusammengefügt und gespeichert werden. (electrooobs, 2021)

```
acc_axis[0] = Wire.read()<<8|Wire.read();
acc_axis[1] = Wire.read()<<8|Wire.read();
acc_axis[2] = Wire.read()<<8|Wire.read();
temp       = Wire.read()<<8|Wire.read();
gyro_axis[0] = Wire.read()<<8|Wire.read();
gyro_axis[1] = Wire.read()<<8|Wire.read();
gyro_axis[2] = Wire.read()<<8|Wire.read();
```

Abbildung 69 MPU-6050 Daten speichern (O'Connor)

Um die Daten beispielsweise vom Beschleunigungssensor von der X-Achse zu erhalten, wird zuerst das Byte von der ersten Speicherstelle gelesen und der Variable acc_axis[1] zugewiesen. Danach wird das Byte um acht stellen nach links verschoben und die 8Bit von der zweiten Speicherstelle angehängt:

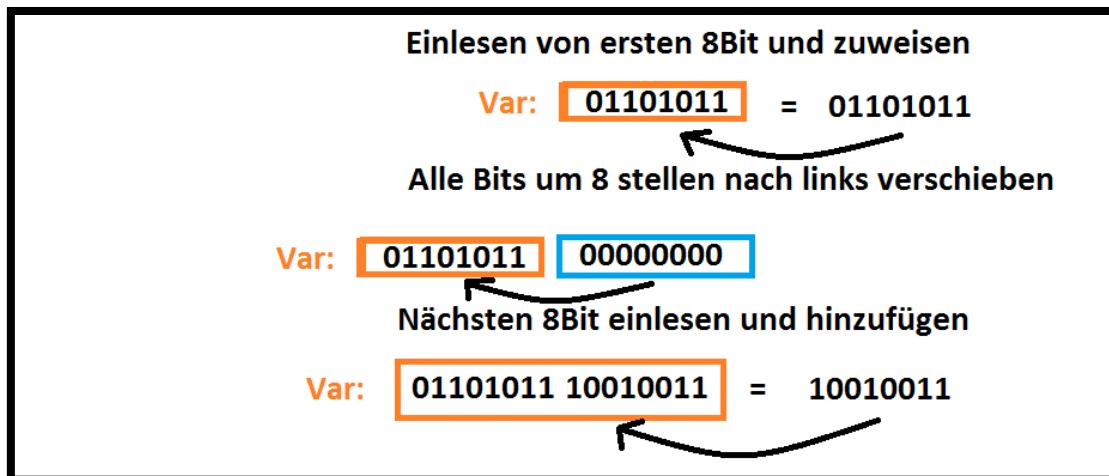


Abbildung 70 Einlesen MPU6050 Bit-Schiftung (O'Connor)

Dieser Vorgang wird für alle sechs Achsen durchgeführt. Anschliessend ist das Einlesen abgeschlossen und die Daten können ausgegeben werden im Serien-Monitor:

```
Serial.print("1: "); Serial.println(gyro_axis[0]);
Serial.print("2: "); Serial.println(gyro_axis[1]);
Serial.print("3: "); Serial.println(gyro_axis[2]);
```

Abbildung 71 Ausgabe von Gyroskop-Daten (O'Connor)

8.1.3.3 Sensordatenkalibrierung

```
void setup() {  
    Wire.begin();  
    TWBR = 12;  
  
    for (int i = 0; i < 2000 ; i++) {  
        read_mpu_6050_data();  
  
        gyro_axis_cal[0] += gyro_axis[0];  
        gyro_axis_cal[1] += gyro_axis[1];  
        gyro_axis_cal[2] += gyro_axis[2];  
  
        delayMicroseconds(4000);  
    }  
  
    gyro_axis_cal[0] /= 2000;  
    gyro_axis_cal[1] /= 2000;  
    gyro_axis_cal[2] /= 2000;
```

Im Gegensatz zum Beschleunigungsmesser ist die Ausgabe des Gyroskops nicht um den Nullpunkt kalibriert. Damit die Daten verwendet werden können, müssen diese im Ruhezustand auf null heruntergerechnet werden. Dazu werden 2000 Datensätze eingelesen und die Durchschnittsabweichung ausgerechnet. Um die 250Hz einzuhalten, gibt es am Ende der Schlaufe jeweils eine Verzögerung. Die Kalibrierung benötigt total 8 Sekunden.

Abbildung 72 Sensordatenkalibrierung ausrechnen (O'Connor)

Für die Verwendung der Gyroskopdaten, kann die Durchschnittsabweichung bei jedem Neuen einlesen direkt abgezogen werden.

```
Serial.print("1: "); Serial.print(gyro_axis[0] - gyro_axis_cal[0]);  
Serial.print("2: "); Serial.print(gyro_axis[1] - gyro_axis_cal[1]);  
Serial.print("3: "); Serial.print(gyro_axis[2] - gyro_axis_cal[2]);
```

Abbildung 73 Sensordatenkalibrierung anwenden (O'Connor)

Direktvergleich Ausgaben mit und ohne Kalibrierung:

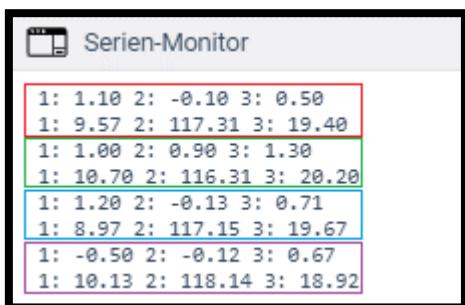


Abbildung 74 Direktvergleich mit und ohne Kalibrierung (O'Connor)

8.2 Montage

Dieses Kapitel befasst sich mit der Montage der Drohne.

8.2.1 Rahmen

Übersicht der einzelnen Teile des Rahmens sowie der Schrauben:

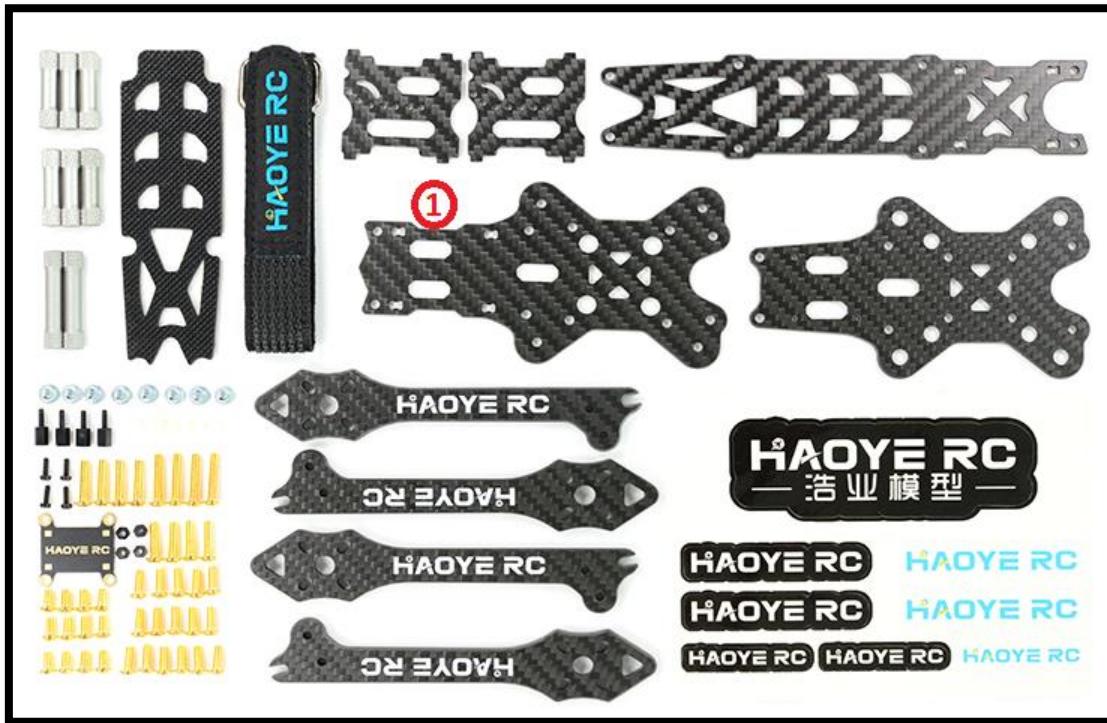


Abbildung 75 Rahmenkomponenten (<https://optimum-racing.ch/de/frame/1361-haoye-rc-x1-5-inch-freestyle-fpv-frame-kit.html>)

Wie aus der Abbildung erkennbar, besteht dieser Rahmen aus zwei verschiedenen Böden. Der Boden, gekennzeichnet mit der (1) ist der Untere. Als ersten Schritt werden alle 12 zentralen Schrauben von unten hereingeführt. Die Arme können anschliessend eingefädelt werden und der zweite Boden daraufgelegt werden. Fest angebracht wird das Ganze mit den Rundmuttern sowie den Platzhaltern, wie im nachfolgenden Bild ersichtlich.



Abbildung 76 Rahmen (O'Connor)

Als letztes Element fehlt noch der Deckel, dieser wird nach Beendung des Innenlebens montiert.

8.2.2 Motoren

Die vier Motoren werden jeweils am Ende jedes Armes von unten mit vier Schrauben ange- schraubt. Dabei sollte beachtet werden, dass die Kabel zur Mitte gerichtet sind.



Abbildung 77 Befestigung der Motoren am Rahmen (O'Connor)

8.2.3 Elektronischer Geschwindigkeitsregler

Nachfolgend die Montage des elektronischen Geschwindigkeitsreglers sowie der Stromversorgung.

8.2.3.1 ESC mit Motor verbinden

Die Kabelenden vom Motor sind mit einem Stecker ausgestattet. Dieser kann abgeschnitten werden, weil die Verbindung gelötet wird. Die Kabelenden vom Motor sowie vom ESC müssen dafür zuerst abisoliert werden. Danach wird jeweils ein Kabel vom Motor mit einem des ESCs verlötet, bis alle drei Kabel verbunden sind.

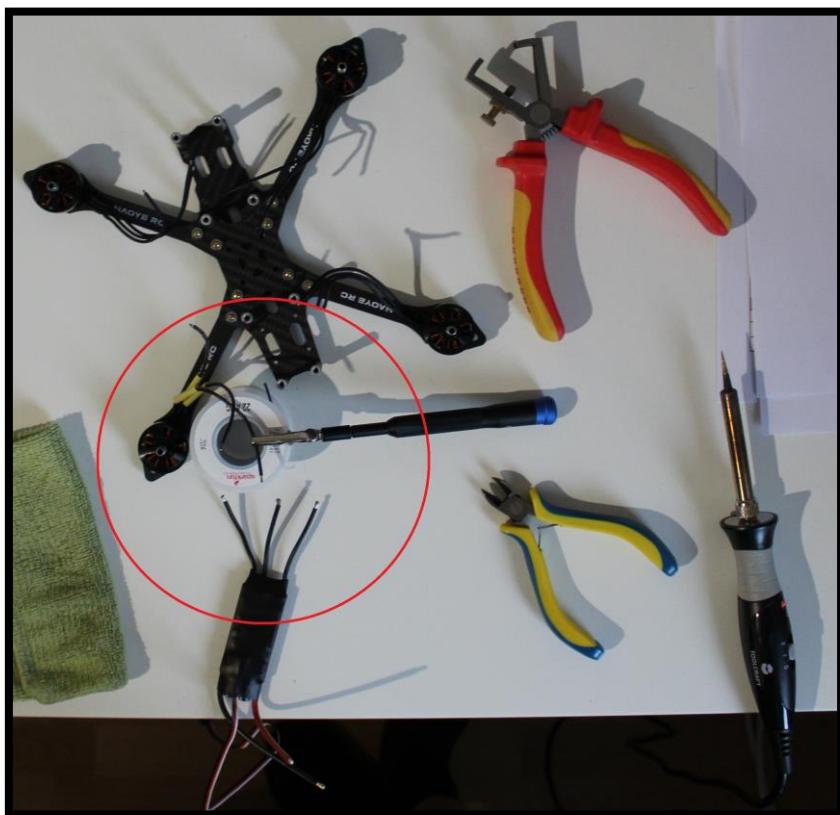


Abbildung 78 ESC & Motor Kabelverbindung (O'Connor)

Die offenen Lötstellen müssen vor der Inbetriebnahme mit einem Schrumpfschlauch isoliert werden.



Abbildung 79 ESC & Motor Kabelverbindung Isolation (O'Connor)

Besonders zu beachten ist die Drehrichtung des BDLC Motors. In meinem Fall konnte ich aus dem Prototyp Nr. 2 feststellen, wenn alle Kabel ohne überkreuzen angeschlossen werden, entsteht eine Drehrichtung im Uhrzeigersinn.

Ich habe die Motoren mit der Reihenfolge beschriftet, wie ich gedenke, diese am Arduino anzuschliessen:

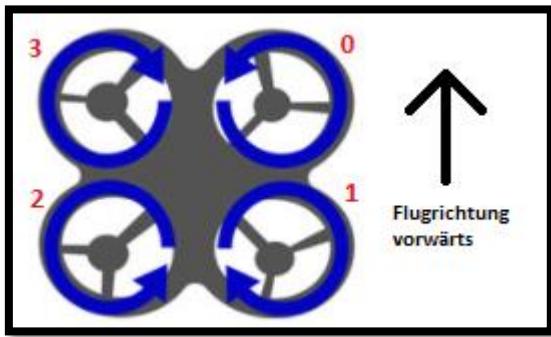


Abbildung 80 Motorenkennzeichnung (FPVRacing, 2021)

Für das Anschliessen der Motoren am ESC müssen entsprechend Motor 3 & 1 normal angeschlossen werden für eine Drehrichtung im Uhrzeigersinn. Bei den Motoren 0 & 2 müssen zwei Kabel überkreuzt werden, damit eine gegenentsetzte Drehrichtung entsteht.

8.2.3.2 ESC am Rahmen befestigen

Für die Befestigung des ESCs am Rahmen eignet sich jeweils die Unterseite des Arms. Mit doppelseitigem Klebeband und einem Kabelbinder wird der ESC befestigt. Ein besonders Auge muss auf die Balance der gesamten Drohne geworfen werden.

8.2.3.3 ESC mit Batterie verbinden

Nach dem Anschliessen der ersten beiden Motoren und ESCs kümmern wir uns um die Strom-zu- bzw. abfuhr. Dabei werden die beiden roten Kabel von den beiden ESCs zusammengeführt und mit einem dritten Kabel verlängert. Danach wird die Stelle zusammengelötet und mit einem Schrumpfschlauch isoliert. Dasselbe wird auch mit den beiden schwarzen GRND Kabeln und einem Erweiterungskabel durchgeführt.

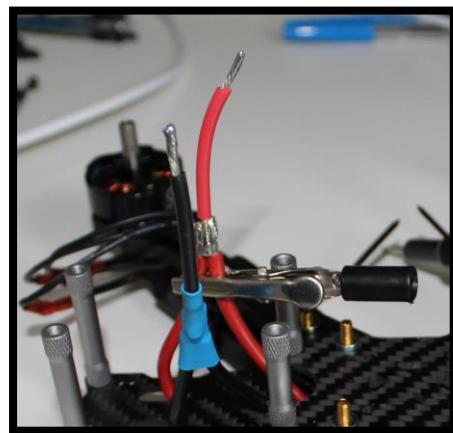


Abbildung 81 ESC Stromanschlussverbindung (O'Connor)

Arduino Drohne selber bauen

Durch das Zusammenführen der Kabel und dem provisorischen Verbinden mit der Batterie entsteht ein kleiner Schaltkreis und die ersten beiden ESCs sowie Motoren können mit dem Arduino und mit der Batterie verbunden werden. Beim Testen sollte auf die richtige Umdrehungsrichtung geachtet werden sowie ein gutes Auge auf die Batterie geworfen werden.

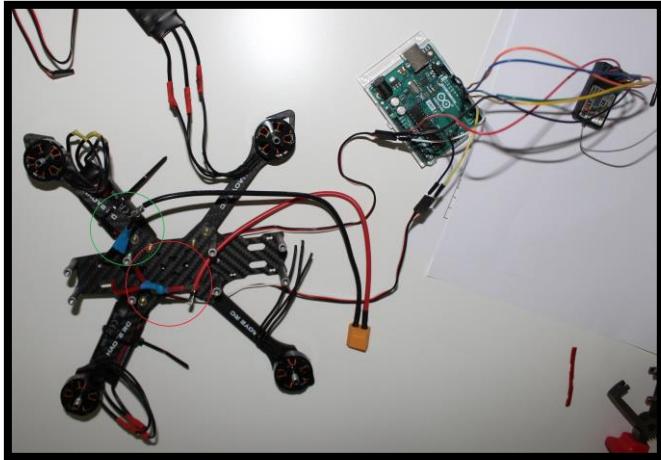


Abbildung 82 Erstes Testen (O'Connor)

Nach dem erfolgreichen Testen kann dasselbe Vorgehen mit den beiden übrigen ESCs und Motoren auf der gegenüberliegenden Seite durchgeführt werden.

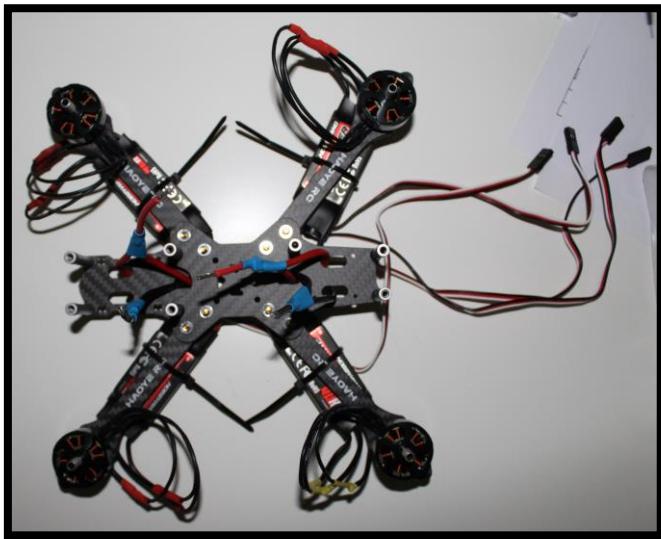


Abbildung 83 Zweites Testen (O'Connor)

Die beiden entstandenen Enden vom Positiv respektive Negativ-Stromanschluss werden nun im nächsten Schritt wieder miteinander verbunden und gelötet. Um einen Kurzschluss zu vermeiden, gilt weiterhin: Rot mit Rot und Schwarz nur mit Schwarz verbinden. Die Kabellänge ist bewusst etwas länger gewählt, damit noch genügend Platz vorhanden ist, um den RC-Empfänger zwischen Boden und Stromanschluss im Zentrum der Drohne anzubringen.

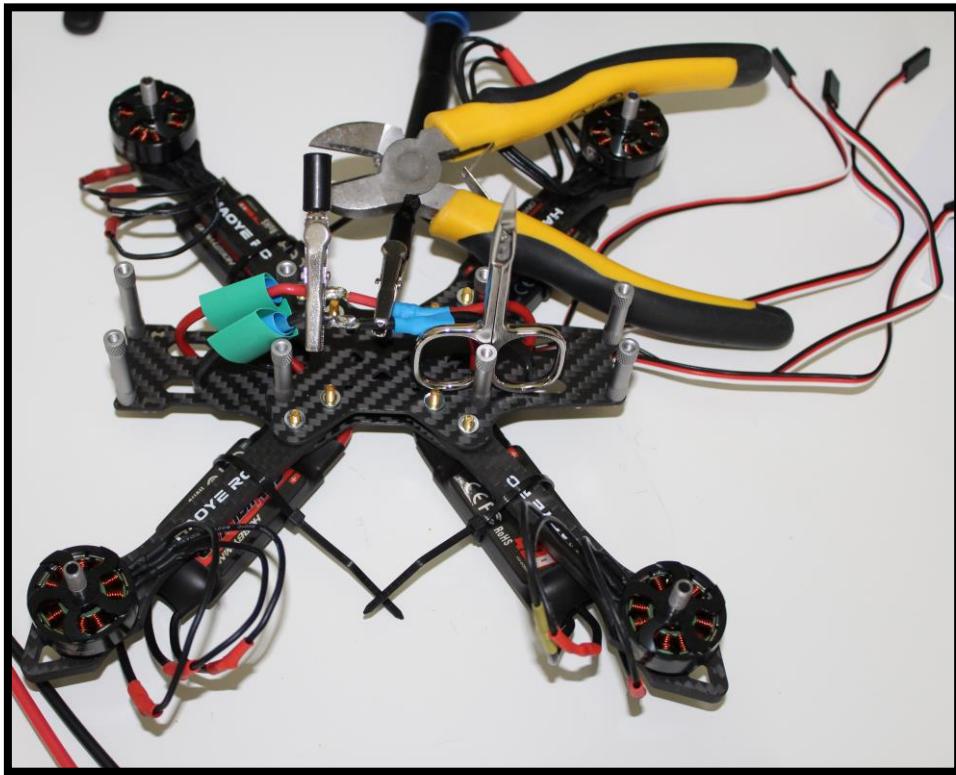


Abbildung 84 Stromkreis von allen ESCs zusammenführen (O'Connor)

Für die Stromzufuhr des Arduinos wird jeweils ein weiterer Draht an die Lötstelle hinzugefügt, welcher an den Arduino VCC für Rot respektive GRND für den schwarzen Draht angegeschlossen wird. Zuletzt werden noch die Kabel des Batterieanschlusses an der gleichen Lötstelle angelötet:

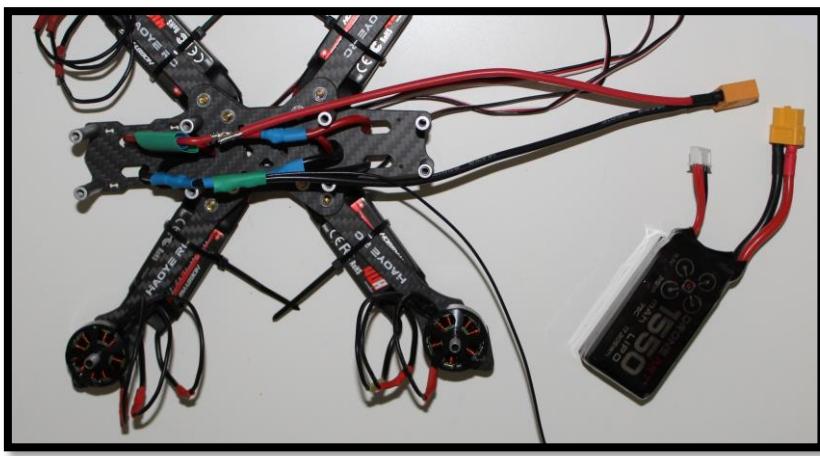


Abbildung 85 Stromkreis Batterie- und Arduino-Anschluss (O'Connor)

Weil dies die zentrale Stelle für die Stromzufuhr ist, werden beide Stellen mehrfach mit Schrumpfschläuchen isoliert.

Um auf der sicheren Seite zu sein, wollte ich das Ganze zuerst testen. LiPo Batterien können extrem gefährlich sein! Diesbezüglich habe ich erstmals nur den GRND Anschluss zusammengelötet damit im Falle des Falles, wenn ein Kurzschluss entstehen sollte oder die Batterie überhitzt, der Stromfluss sehr schnell unterbrochen werden kann durch die Trennung des roten Batterieanschlusskabels. Die Signalkabel habe ich wieder mit dem Arduino provisorisch verbunden:

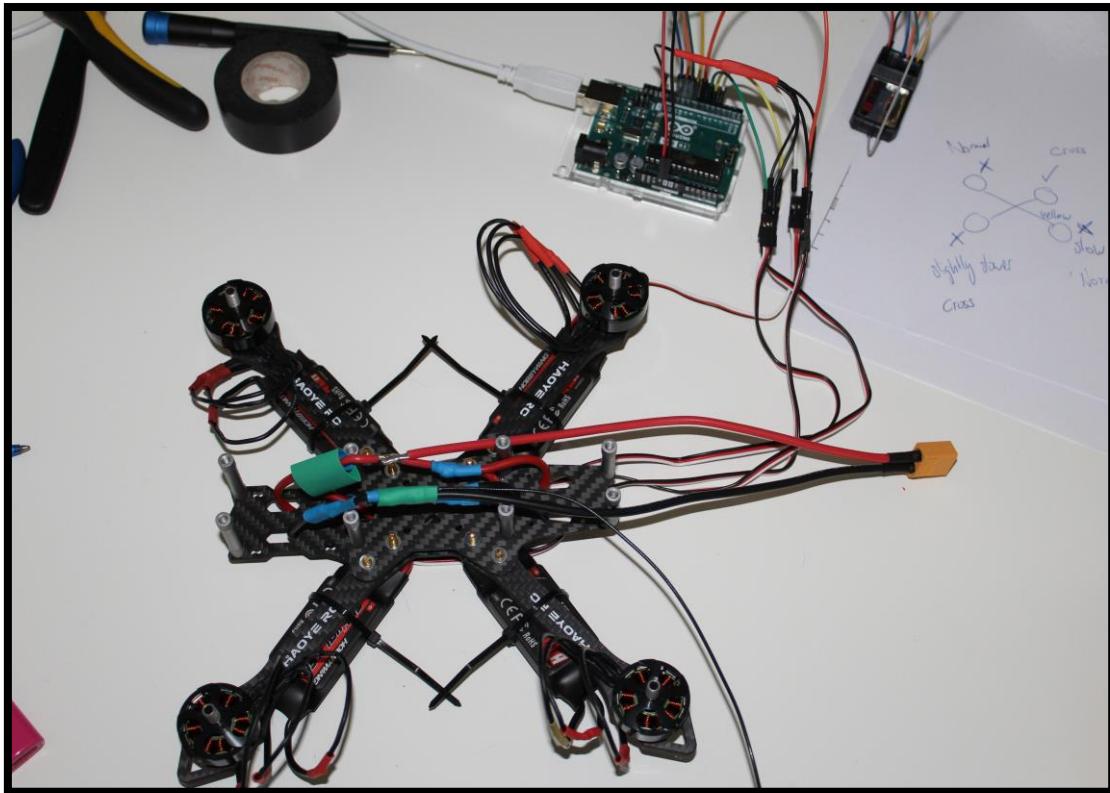


Abbildung 86 LiPo Test (O'Connor)

8.2.3.4 ESC mit Arduino verbinden

Für die definitive Verbindung von den ESCs und dem Arduino der Datenkabel werden alle Erdungskabel zusammengeführt:

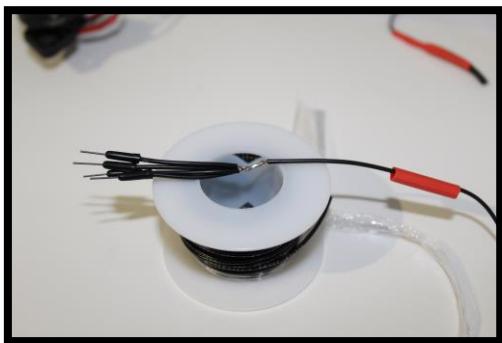


Abbildung 87 Erstellung eines Erdungskabel (O'Connor)

Für eine schönere Kabelführung und um eine bessere Übersicht aller Kabel zu erhalten, werden alle Datenkabel mit einem Schrumpfschlauch gebündelt und das zuvor erstellte Erdungskabel an allen Erdungsschnittstellen angeschlossen.

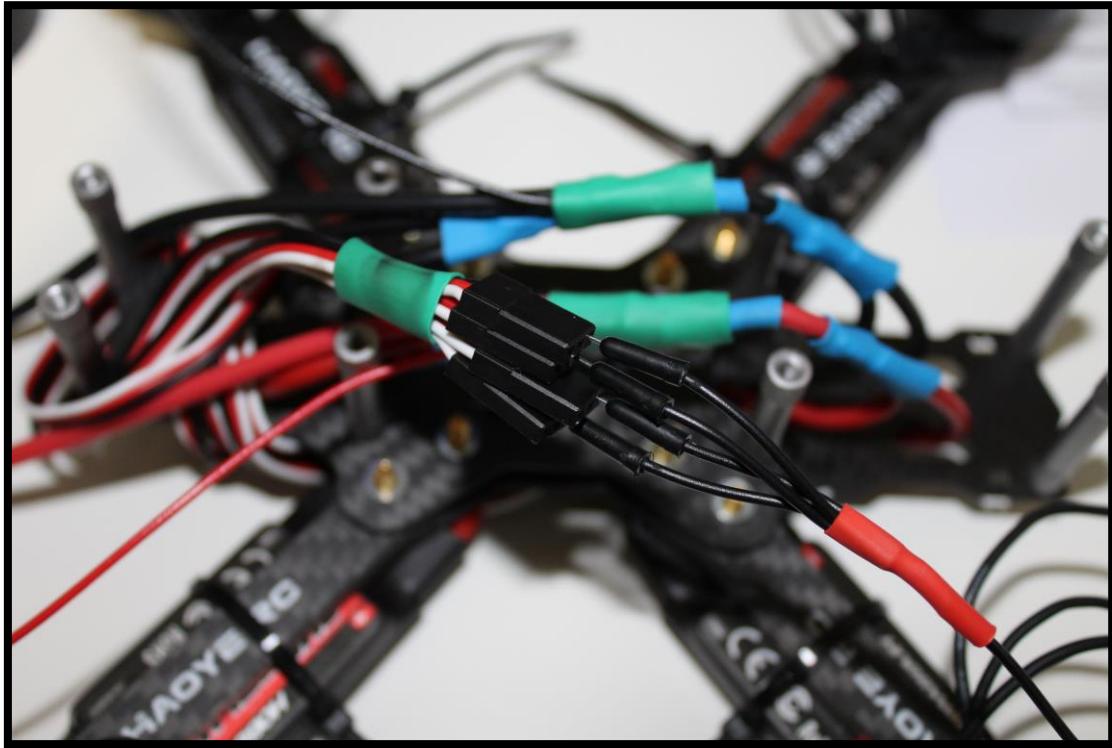


Abbildung 88 Erdungskabel mit ESC-Signalkabel verbinden (O'Connor)

Das PWM Signal wird über das weisse Kabel vom ESC übertragen. Es müssen vier Drähte angeschlossen werden, welche zu den entsprechenden Pins am Arduino führen.

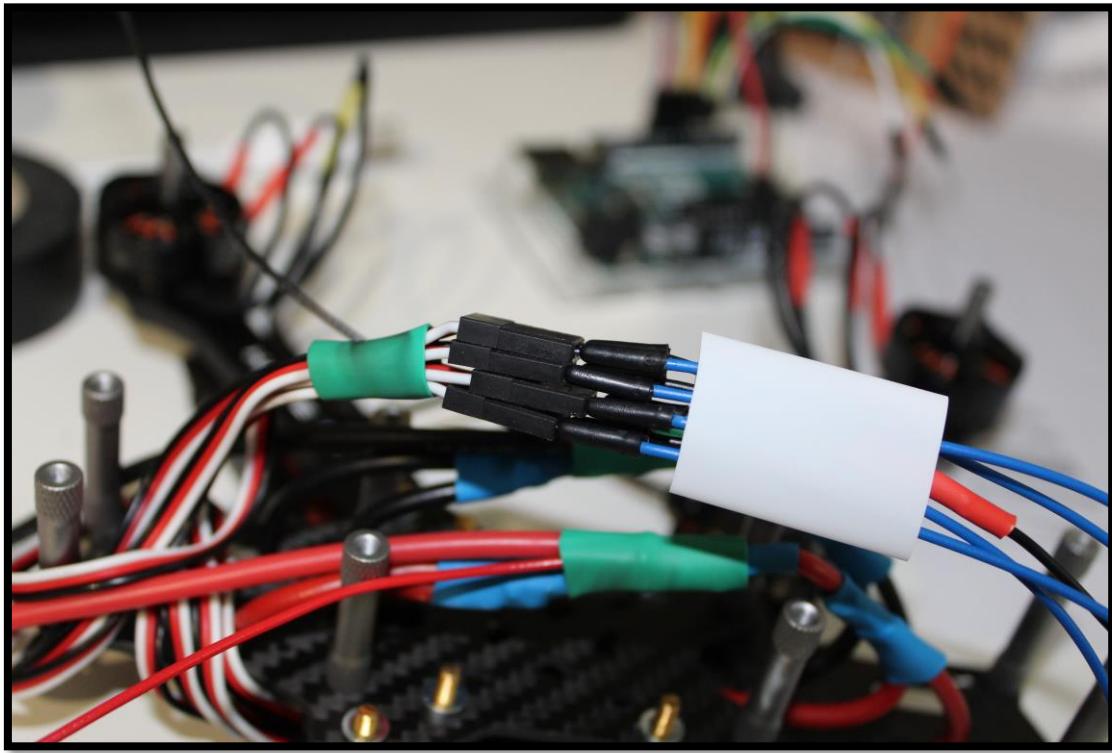


Abbildung 89 Arduino- mit ESC-Signalkabel verbinden (O'Connor)

Damit sich die Kabel nicht während des Fluges loslösen können und weil sie neben den Stromkabeln geführt werden, habe ich das gesamte Paket mit weiteren Schrumpfschläuchen umhüllt für eine gute Isolation auch zum Schutz vor Interferenzstrom.



Abbildung 90 Interferenzstrom Isolierung der Datenkabel (O'Connor)

8.2.4 RC Empfänger

Der RC-Empfänger wird im inneren Kern am Boden der Drohne unterhalb der Stromkabel mit doppelseitigem Klebeband festgeklebt, sodass die Verbindungsports zum Arduino frei bleiben.

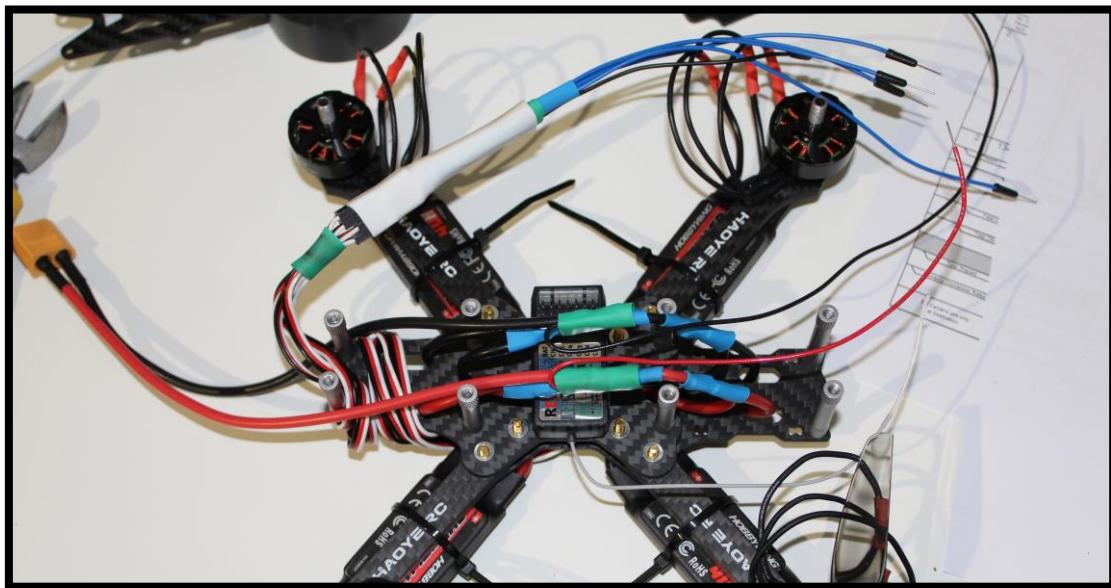


Abbildung 91 RC-Empfänger Montage (O'Connor)

Wenn alle Kabel im Innenraum verstaut sind, kann das Dach des Rahmens montiert werden:

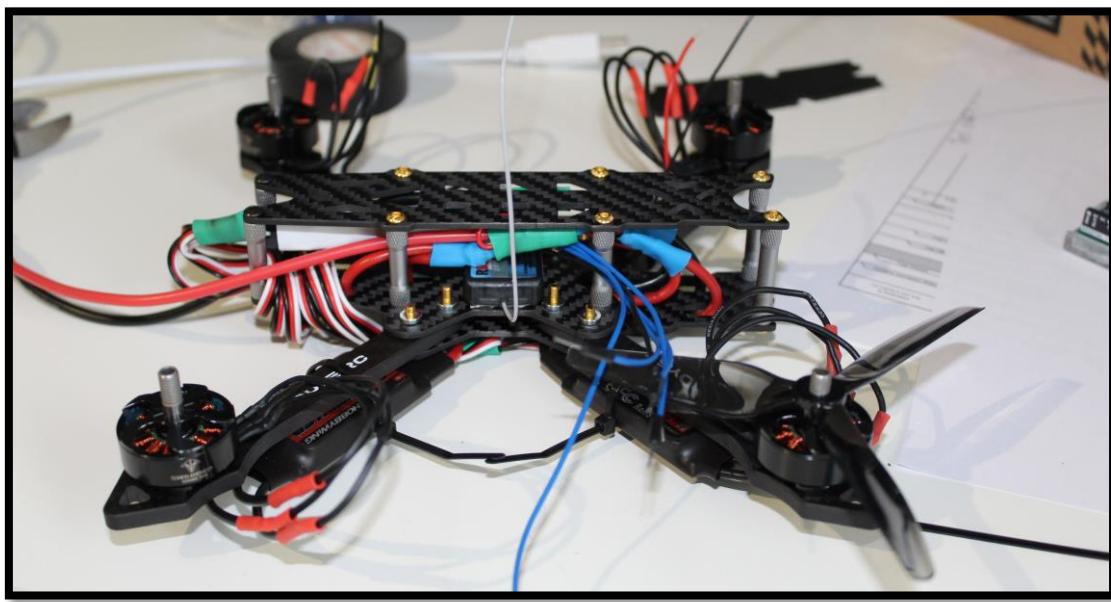


Abbildung 92 Montage des Dachs (O'Connor)

8.2.5 Flugkontroller Arduino

Um die beste Positionierung für den Arduino sowie für die Batterie zu finden, werden beide auf das Dach gesetzt. Mit zwei Fingern kann die Drohne am Mittelpunkt des Rahmens angehoben werden und die Batterie sowie der Arduino verschoben werden, bis die bestmögliche Balance gefunden wurde. Die Stelle des Arduino und der Batterie sollte markiert werden.

Der Arduino wird anschliessend am Rahmen mit doppelseitigem Klebeband montiert und alle Kabel können wie vom Schaltplan ersichtlich angeschlossen werden.

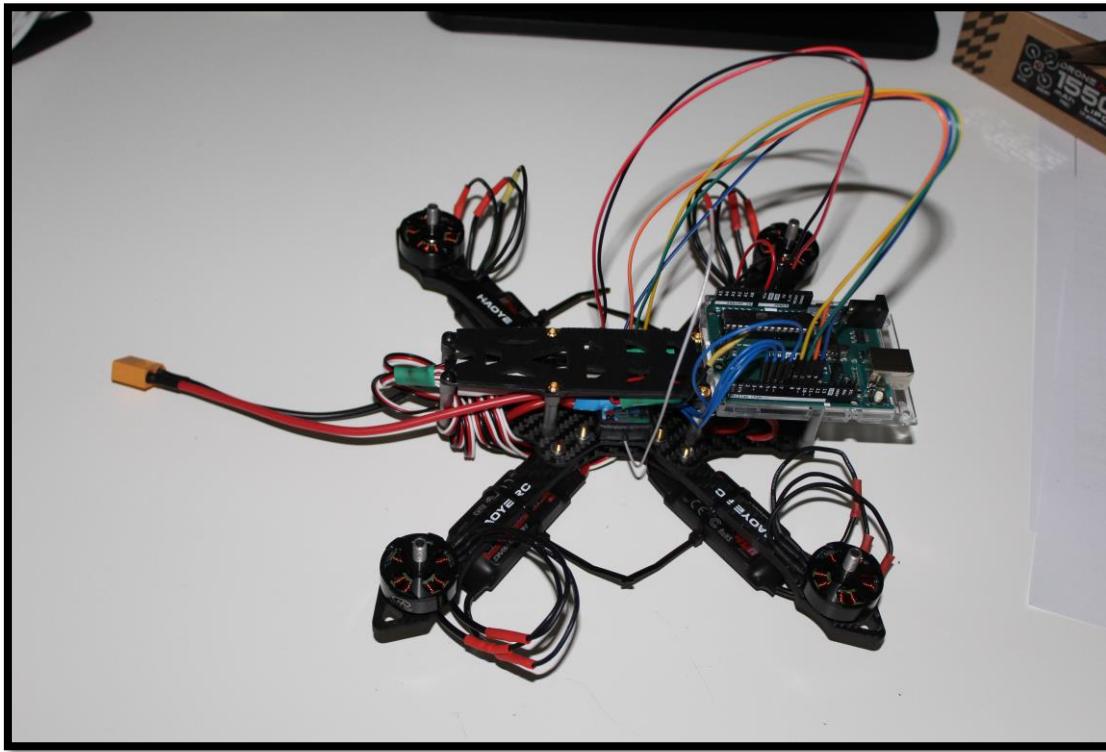


Abbildung 93 Arduino am Rahmen befestigen (O'Connor)

8.2.6 Datenkabelvalidierung

Nach der Implementierung der Businesslogik zu Pitch, Yaw & Roll im Release 2 kann die Drohne zur Überprüfung dessen sowie ob die Datenkabel und Stromkabel richtig angeschlossen wurden, für Testzwecke in Betrieb genommen werden ohne Propeller.

8.2.7 Kabelmanagement

Es ist wichtig, dass es keine losen Kabel gibt, welche durch die Propeller beschädigt werden könnten. Die Verbindungs kabel von den Motoren zu den ESCs werden mit Kabelbinder am Rahmen befestigt. Sämtliche PWM Signal- sowie Stromkabel werden auf eine angebrachte Länge gekürzt und ebenfalls am Rahmen angemacht.

8.2.8 Propeller

Die Propeller werden aus Sicherheitsgründen erst montiert, wenn die Drohne die erste initiale Testphase überstanden hat und sichergestellt werden kann, dass alles einwandfrei funktioniert. Zu beachten ist, dass die Propeller sich in CW und CCW Bauform unterscheiden und eine vordefinierte Drehrichtung haben.

8.2.9 Batterie

Die Batterie wird am zuvor markierten Punkt mit einem Kabelbinder am Rahmen befestigt.

8.2.10 Fahrwerk

Um die Kabel auf der Unterseite gut zu schützen, wird ein Fahrwerk aus Styropor gefertigt. Damit das Styropor reibungsrobust wird, wird dieses zusätzlich mit Klebeband umwickelt. Anschliessend kann es an der Unterseite angebracht werden.



Abbildung 94 Fahrwerk
(O'Connor)

8.2.11 Trägheitsmessungseinheit

Der MPU6050 sollte so zentral wie möglich am Quadrokopter angebracht werden. Doppelseitiges Klebeband mit einer Schaumschicht eignet sich am besten, um die Vibrationen der Motoren etwas abzudämmen. Die Orientierung des MPU6050 spielt eine sehr wichtige Rolle. Anhand der auf der Frontseite markierten Pfeile kann die Orientierung ausfindig gemacht werden:

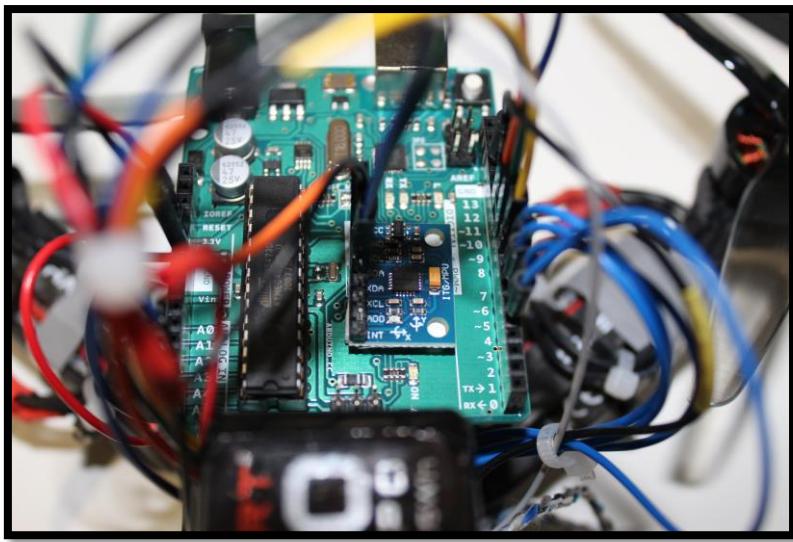


Abbildung 95 MPU-6050 Orientierung (O'Connor)

8.2.12 Endergebnis

Nachfolgend ein Bild von dem Resultat der Montage:

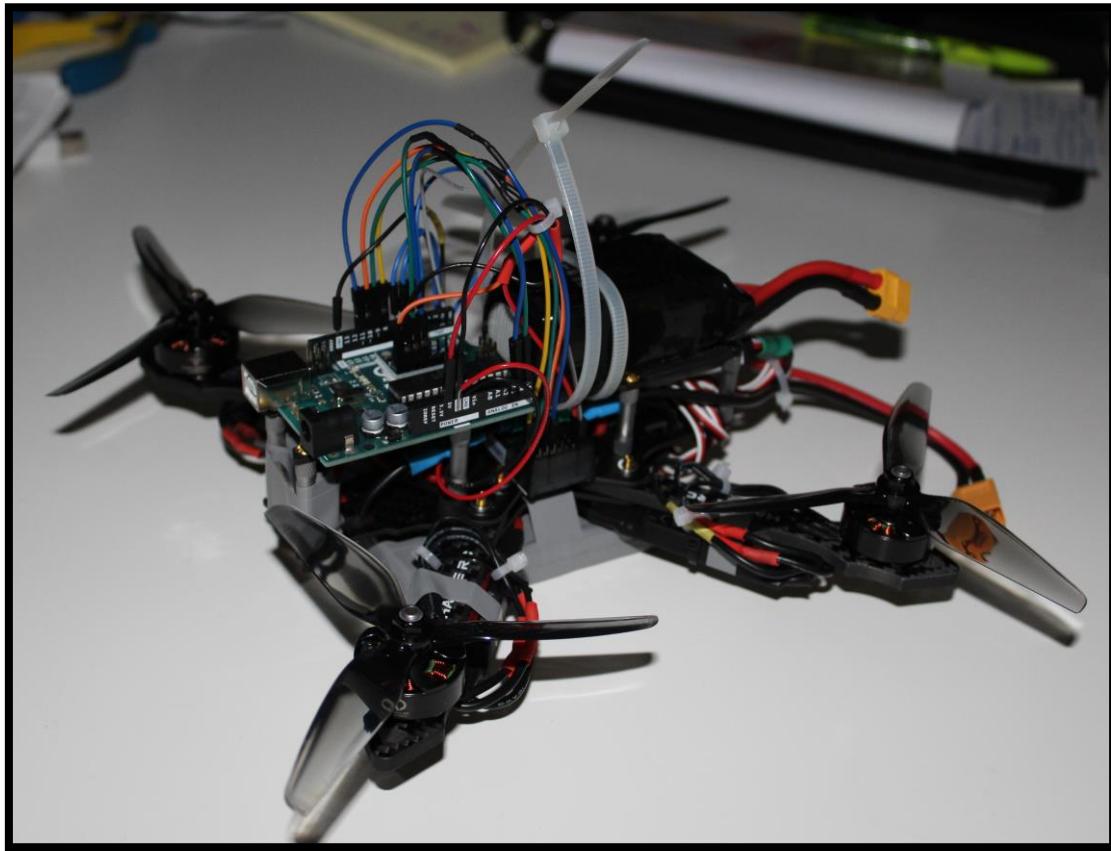


Abbildung 96 Abschluss der Montage (O'Connor)

8.3 Programmieren

Im folgenden Kapitel geht es um die Implementierung des Quellcodes.

8.3.1 Release 1: Throttle (Gas)

Der erste Release wurde zusammen mit den ersten beiden Prototypen entwickelt und ist in den entsprechenden Kapiteln beschrieben.

8.3.2 Release 2: Pitch, Yaw & Roll

Für die Kontrolle über die Bewegung in der Luft können bei den einzelnen Motoren jeweils die Geschwindigkeit manipuliert werden. Dadurch kann eine gewünschte Flugrichtung erzeugt werden.

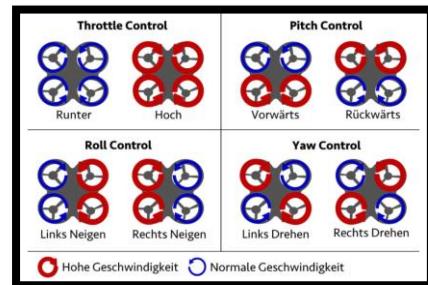


Abbildung 97 Quadrokopter Bewegung durch Rotoren Geschwindigkeitsanpassung (FPVRacing, 2021)

Durch den ersten Prototyp vom RC-Empfänger konnten die einzelnen Channels ausfindig gemacht werden:

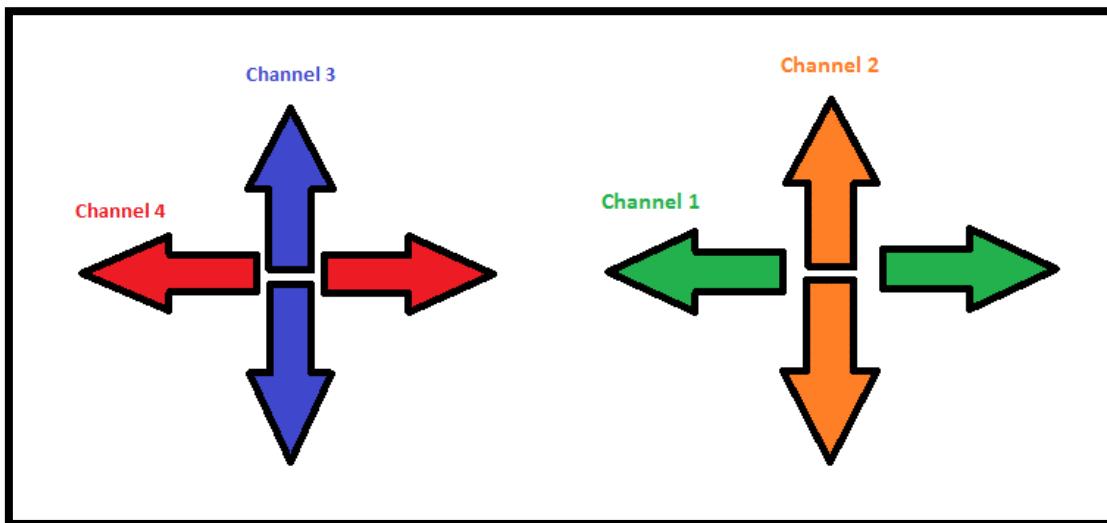


Abbildung 98 RC-Sender Kanalzuweisung (O'Connor)

Daraus können die einzelnen Channels dem jeweils zuständigen Bewegungsbefehl zugewiesen werden:

1. CH => Roll
2. CH => Pitch
3. CH => Throttle
4. CH => Yaw

Arduino Drohne selber bauen

Der Throttle-Wert vom Channel 3 wird von maximal 2000µs auf 1800µs herunterskaliert. Dadurch entsteht ein freier Restwert von 200µs für die Steuerung der einzelnen Motoren. Die Werte der PWM-Signale von den Channels 1, 2 und 4, welche für die Bewegung der Drohne zuständig sind, werden von 1000µs bis 2000µs auf -100µs bis +100µs umgewandelt mit der map() Funktion.

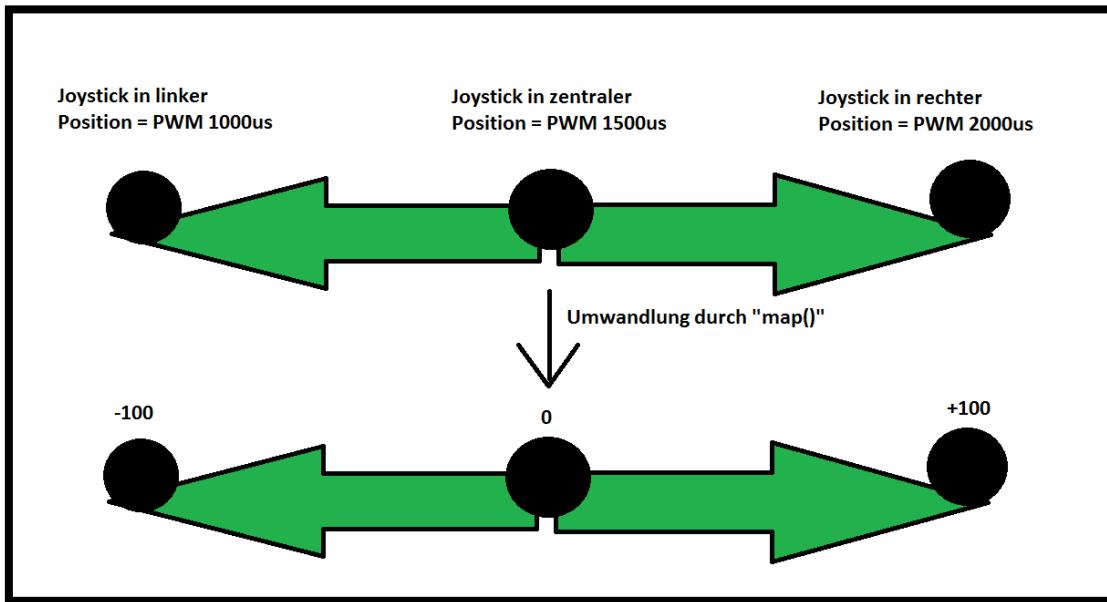


Abbildung 99 PWM Signalumwandlung (O'Connor)

```
roll      = map(PWM_channel_input[0], PWM_MIN, PWM_MAX, -100, 100);
pitch     = map(PWM_channel_input[1], PWM_MIN, PWM_MAX, -100, 100);
throttle  = map(PWM_channel_input[2], PWM_MIN, PWM_MAX, 1000, 1800);
yaw       = map(PWM_channel_input[3], PWM_MIN, PWM_MAX, -100, 100);
```

Abbildung 100 PWM Signalumwandlung Code (O'Connor)

Der Throttle-Wert bildet die Grundlage für alle Motoren:

```
ESC_PWM[0] = throttle;
ESC_PWM[1] = throttle;
ESC_PWM[2] = throttle;
ESC_PWM[3] = throttle;
```

Abbildung 101 ESC Ausgangsgeschwindigkeit (O'Connor)

Danach wird durch logische Operationen aufgrund des Werts aus der Roll, Pitch und Yaw variable den entsprechenden Motoren ein Wert der Position vom Joystick zwischen 0-100 addiert.

Sendet der CH1 ein PWM Signal welches grösser ist als 1500µs, wird dieser in einen Wert von 0 bis 100 umgewandelt und der rechte Joystick neigt nach rechts. Für ein Rollmanöver nach rechts muss bei den linken Motoren 2 & 3 der Wert addiert werden, damit diese schneller drehen wie die rechten.

```
// Roll
if(roll > 0) {
    ESC_PWM[3] += roll;
    ESC_PWM[2] += roll;
} else {
    ESC_PWM[0] -= roll;
    ESC_PWM[1] -= roll;
}
```

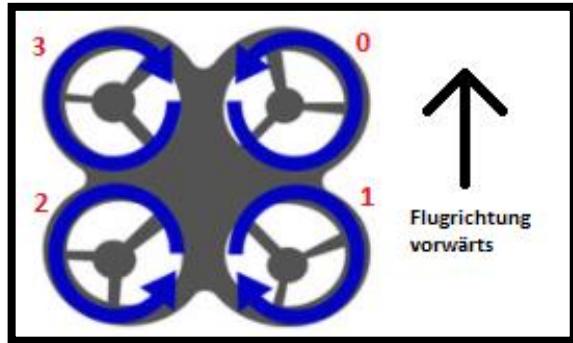


Abbildung 102 Basiswert mit gewünschter Flugrichtungsbefehl addieren (O'Connor)

Hingegen, wenn der CH1 ein PWM Signal sendet, welches kleiner ist als 1500µs, neigt sich der Joystick nach links und es wird ein Wert von -100 bis 0 der variabel zugewiesen. Dadurch wird bei den Motoren 0 & 1 der Wert addiert, denn es gilt:

$$x - (-y) = x + y$$

Nach derselben Logik sind auch Pitch und Yaw implementiert worden:

```
// Pitch
if(pitch > 0) {
    ESC_PWM[1] += pitch;
    ESC_PWM[2] += pitch;
} else {
    ESC_PWM[0] -= pitch;
    ESC_PWM[3] -= pitch;
}
```

```
// Yaw
if(yaw > 0) {
    ESC_PWM[0] += yaw;
    ESC_PWM[2] += yaw;
} else {
    ESC_PWM[3] -= yaw;
    ESC_PWM[1] -= yaw;
}
```

Abbildung 104 Pitch-Kalkulation (O'Connor)

Abbildung 105 Yaw-Kalkulation (O'Connor)

Der Throttle-Wert wurde bewusst um 200µs verkleinert, denn: Sollte z. B. der rechte Joystick sich in einer Ecke befinden, dann wird gleichzeitig Roll und Pitch vom Motor gegenüberliegend zur Joystickrichtung zusammen addiert für maximal +200µs.

Dadurch, dass der Yaw-Befehl zusätzlich nochmals +100µs potenziell dazu addieren könnte, gibt es einen Schutz Mechanismus:

```
// MAX PWM Failsafe
if(ESC_PWM[0] > PWM_MAX) ESC_PWM[0] = PWM_MAX;
if(ESC_PWM[1] > PWM_MAX) ESC_PWM[1] = PWM_MAX;
if(ESC_PWM[2] > PWM_MAX) ESC_PWM[2] = PWM_MAX;
if(ESC_PWM[3] > PWM_MAX) ESC_PWM[3] = PWM_MAX;
```

Abbildung 106 Maximaler PWM-Signal Schutz (O'Connor)

Die resultierenden Werte aus dem Array ESC_PWM können nun an die entsprechenden ESCs wieder übertragen werden wie im Prototyp 2.

8.3.3 Release 3: Autonomes Schweben – MPU6050 PID Regler

Im nachfolgenden Kapitel werden alle Schritte für das autonome Schweben implementiert.

8.3.3.1 Aktuelle Position bestimmen

Damit die aktuelle Orientierung im Dreidimensionalen Raum gemessen werden kann, wird ein Startpunkt für alle drei Achsen von 0 definiert. Anschliessend können die Messungen des Gyroskops kontinuierlich addiert respektive subtrahiert werden. Somit weiss der Flugkontroller stetig, auf welcher Achse welche Abweichungen vom Nullpunkt vorhanden sind und kann entsprechende Gegenmassnahmen treffen, um einen Ausgleich zu schaffen und dadurch konstant in der Luft ausbalanciert sein.

Die Messungen vom Gyroskop müssen dafür zuerst umgerechnet werden. Im Datenblatt wurde definiert, dass ein gemessener Wert von $65.5 = 1^\circ/\text{s}$ entspricht, wenn die maximal zu messende Gradänderung auf $500^\circ/\text{s}$ eingestellt wurde. Dieser errechnete Grad/s muss nochmals umgerechnet werden auf die Dauer der Main loop(). Diese durchläuft eine Wiederholung von 250Hz (4ms).

$$\frac{65.5}{250\text{Hz}} = \text{Gradänderung pro Durchlauf der Main loop()}$$

Um die Rechenleistung des Mikrocontrollers zu reduzieren, wird die Rechnung als Multiplikation umgeformt:

$$\text{Gyroskopmessung} * 0.0000611 = \text{Gradänderung}$$

```
// 0.0000611 = 1 / (250Hz / 65.5)
pitch_offset += gyro_axis[0] * 0.0000611;
roll_offset   += gyro_axis[1] * 0.0000611;
yaw_offset    = gyro_axis[2] * 0.0000611;
```

Abbildung 107 Gyroskop Winkelberechnung (O'Connor)

Mit dieser einfachen Umrechnung kann bereits die aktuelle Position ausgegeben werden, wenn keine Drehung auf der Z-Achse stattfindet.

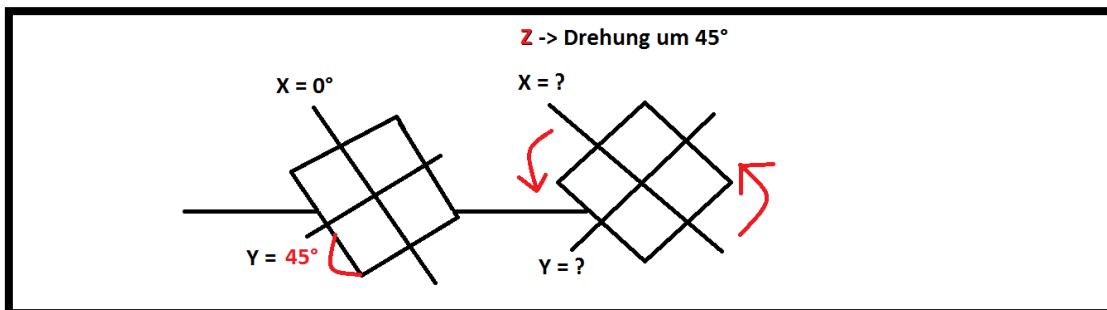


Abbildung 108 Visualisierung des Winkelberechnungsproblems bei Drehung auf der Z-Achse (O'Connor)

Durch die Drehung der Z-Achse muss die Steigung der X- und Y-Achse angepasst werden:

$$Y \text{ Steigung} -= X \text{ Steigung} * \sin(Z \text{ Gradänderung} * \frac{\pi}{180})$$

$$X \text{ Steigung} += Y \text{ Steigung} * \sin(Z \text{ Gradänderung} * \frac{\pi}{180})$$

(Fisher, 2015)

Die Arduino sin()-Funktion erwartet als Parameter die Einheit Radian, deswegen muss die Gradmessung multipliziert werden mit pi / 180. (Arduino, 2021)

Übersetzt in Arduino-Code:

```
// 0.01745555 = 1 / (3.142(PI) / 180°) ° to Radian
pitch_offset -= roll_offset * sin(yaw_offset * 0.01745555);
roll_offset += pitch_offset * sin(yaw_offset * 0.01745555);
```

Abbildung 109 Gyroskop Winkelberechnung mit Z-Achse (O'Connor)

8.3.3.2 Abweichungen

Die errechnete Position vom Gyrosensor hat die Eigenschaft, im Laufe der Zeit abzuweichen von der tatsächlichen Position. Dies ist auf einen Rundungsfehler zurückzuführen durch die kontinuierliche Addition mit einer pi Multiplikation. Um dem entgegenzuwirken, muss zusätzlich auch die Position aus den Werten des Beschleunigungsmessers errechnet werden.

$$Y \text{ Steigung} = \sin^{-1}\left(\frac{Y}{\sqrt{X^2 + Y^2 + Z^2}}\right)$$

$$X \text{ Steigung} = \sin^{-1}\left(\frac{X}{\sqrt{X^2 + Y^2 + Z^2}}\right)$$

(Fisher, 2015)

Im Arduino Code muss das Ergebnis von Radianen zu Grad noch umgerechnet werden:

```
// 1 rad = 57.296°
gravity_vector = sqrt((acc_axis[0]*acc_axis[0])+(acc_axis[1]*acc_axis[1])+(acc_axis[2]*acc_axis[2]));
if(abs(acc_axis[1]) < gravity_vector) acc_pitch_angle = asin((float) acc_axis[1] / gravity_vector) * 57.296;
if(abs(acc_axis[0]) < gravity_vector) acc_roll_angle = asin((float) acc_axis[0] / gravity_vector) * -57.296;
```

Abbildung 110 Beschleunigungsmesser Winkelberechnung (O'Connor)

8.3.3.3 Komplementärfilter

Da die Daten des Gyroskops einen kumulativen Fehler enthalten und die Sensordaten des Beschleunigungsmessers sehr sensibel auf Vibrationen von den Motoren reagieren, werden beide Daten mittels eines einfachen Komplementärfilters zusammengeführt. Dadurch

werden die kumulativen Fehler sowie Vibrationsstörungen gegenseitig aufgehoben: (Albaghdadi & Ali, 2019)

```
pitch_offset = pitch_offset * 0.9996 + acc_pitch_angle * 0.0004;
roll_offset  = roll_offset  * 0.9996 + acc_roll_angle  * 0.0004;
```

Abbildung 111 Komplementärfilter (O'Connor)

8.3.3.4 PID-Regler

Ein Proportional-Integral-Differenzial-Regler ist ein Regelkreis-Rückkopplungsmechanismus, welcher in Regelsystemen verwendet wird. Er berechnet kontinuierlich den Fehlerwert $e(t)$ als Differenz zwischen einem gewünschten Sollwert und den zuvor kalkulierten Neigungswinkel. Dies ist notwendig, damit die einzelnen Motoren wissen, wie viel schneller sie relativ zueinander Drehen müssen, um die Balance aufrecht erhalten zu können. (studyflix, 2019)

Ein PID Regler setzt sich aus einem P-, I- und D-Glied zusammen:

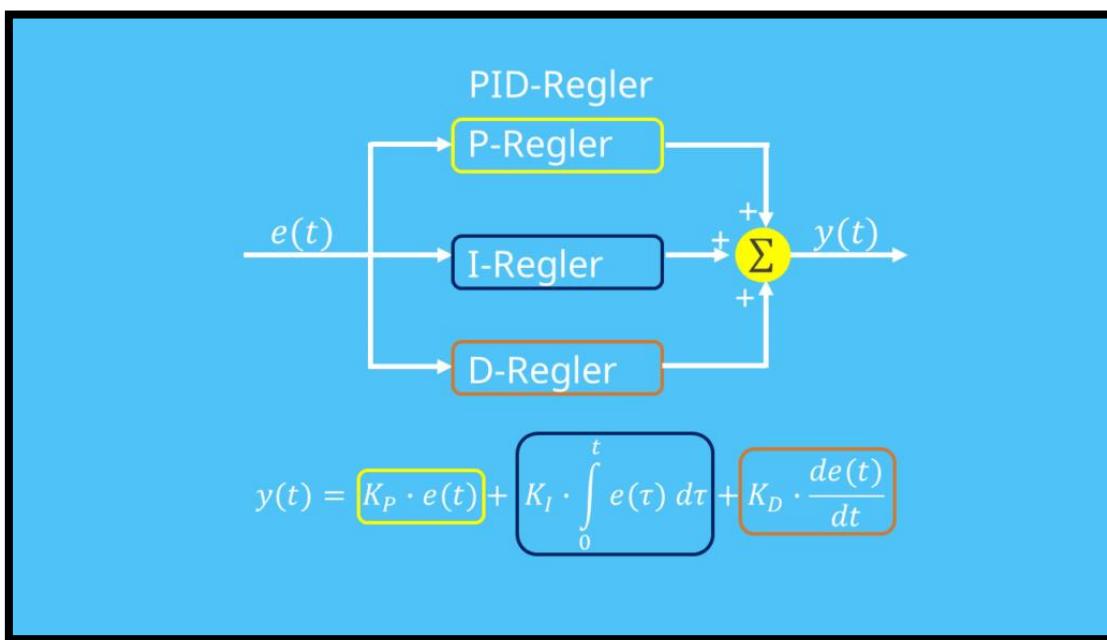


Abbildung 112 PID-Regler (studyflix, 2019)

Die Werte K_P , K_I und K_D sind konstante, welche gesetzt werden. Sie werden beim PID-Tuning nachträglich justiert.

8.3.3.4.1 Proportional

Der Fehlerwert $e(t)$ wird proportional vergrössert. Dadurch kann die Reaktionszeit beeinflusst werden. Ist die Vergrösserung zu Gross, kommt es zu einer Übersteuerung. (Charing & Granath, 2020)

8.3.3.4.2 Integral

Die Ausgabe der Integralrechnung ist relativ zur Fehlerhistorie. Bei kleinen Abweichungen, welche nicht mehr vom Proportionalteil aufgefangen werden können, werden über eine Zeitperiode kumuliert. (Charing & Granath, 2020)

8.3.3.4.3 Derivative

Die Ausgabe der Derivative Rechnung ist relativ zur Zukunft des Fehlers. Wenn der Istwert plötzlich stark verändert wird, hilft dieser, die Werte vom Proportional- und Integralanteil zu glätten, damit es nicht zu einer Übersteuerung kommt. (Charing & Granath, 2020)

Nehmen wir an, die Drohne ist um 45° gekippt. Um dies initial auszugleichen, wird der P-Anteil die Korrektur vornehmen:

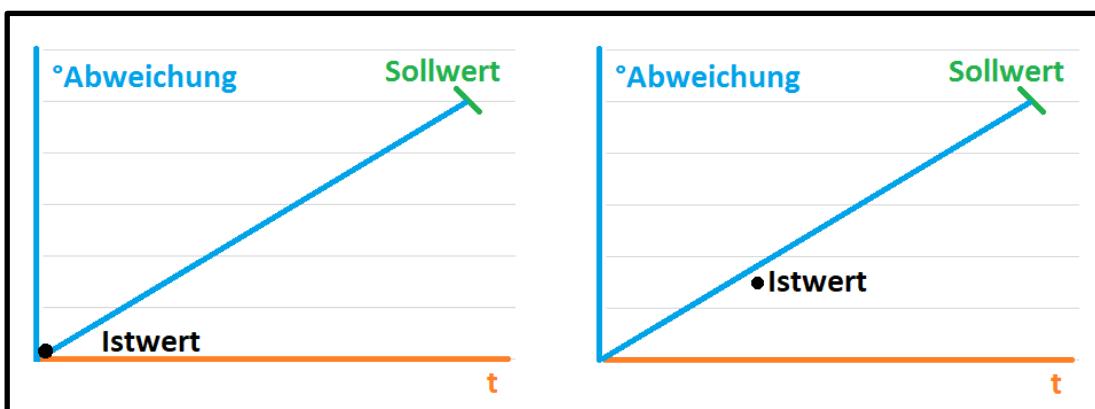


Abbildung 113 PID-Regler verlauf (O'Connor)

Der Istwert hat sich in die korrekte Richtung verschoben, ist aber noch nicht perfekt auf der Abweichung. Bleibt der Istwert konstant unter der Kurve, wird die kleine Abweichung vom Integral aufgefasst und nachkorrigiert.

8.3.3.5 Berechnung

Als Erstes muss der Error-Wert $e(t)$ berechnet werden. Dadurch, dass die Main loop() konstant mit 250Hz durchläuft, kann die Zeit (t) in den Berechnungen weggelassen werden. Es muss lediglich der Soll- von der Ist-Steigung subtrahiert werden: (Brokking, 2021)

```
gyro_roll_angle = (gyro_roll_angle * 0.7) + ((gyro_axis[1] / 65.5) * 0.3);
pid_error      = gyro_roll_angle - pid_desired_roll_angle;
```

Abbildung 114 Error-Wert $e(t)$ berechnen (O'Connor)

8.3.3.5.1 Proportional

Der Error-Wert wird mit dem vordefinierten Wert von K_p multipliziert: (Brokking, 2021)

```
pid_p_roll = pid_kp * pid_error;
```

Abbildung 115 P-Rechnung (O'Connor)

8.3.3.5.2 Integral

Der Error-Wert wird mit dem vordefinierten Wert von K_i multipliziert und der entstandene Wert wird kontinuierlich bei jedem Durchlauf aufsummiert: (Brokking, 2021)

```
pid_i_roll += pid_ki * pid_error;
```

Abbildung 116 I-Rechnung (O'Connor)

8.3.3.5.3 Derivative

Im dritten Schritt wird der Error-Wert vom letzten Durchlauf vom aktuellen Error-Wert subtrahiert und mit dem vordefinierten Wert von K_d multipliziert: (Brokking, 2021)

```
pid_d_roll = pid_kd * (pid_error - pid_d_roll_last_error);
```

Abbildung 117 D-Rechnung (O'Connor)

Anschliessend wird der neue Error-Wert für den nächsten Durchlauf gespeichert: (Brokking, 2021)

```
pid_d_roll_last_error = pid_error;
```

Abbildung 118 $e(t)$ speichern (O'Connor)

Im letzten Schritt werden die drei errechneten Werte zusammenaddiert: (Brokking, 2021)

```
pid_roll = pid_p_roll + pid_i_roll + pid_d_roll;
```

Abbildung 119 PID Σ (O'Connor)

Diese Berechnung muss für jede Achse wiederholt werden. Daraus entstehen die drei Werte pid_roll, pid_pitch und pid_yaw. Die Werte K_p , K_i und K_d müssen für die Roll- und Pitch-Berechnung gleich sein. Die Werte für Yaw können abweichen. (Brokking, 2021)

```
gyro_roll_angle = (gyro_roll_angle * 0.7) + ((gyro_axis[1] / 65.5) * 0.3);
pid_error       = gyro_roll_angle - pid_desired_roll_angle;

pid_p_roll     = PID_KP * pid_error;
pid_i_roll    += PID_KI * pid_error;
pid_d_roll    = PID_KD * (pid_error - pid_d_roll_last_error);

pid_d_roll_last_error = pid_error;
pid_roll        = pid_p_roll + pid_i_roll + pid_d_roll;
```

Abbildung 120 PID-Rechnung der Roll-Achse (O'Connor)

8.3.3.6 PID mit Motoren verbinden

Die kalkulierten PID-Werte werden anschliessend den jeweiligen Motoren nach Zuständigkeits addiert respektive subtrahiert: (electrongoobs, 2021)

```
ESC_PWM[0] = PWM_channel_input[2] - pid_pitch + pid_roll - pid_yaw;  
ESC_PWM[1] = PWM_channel_input[2] + pid_pitch + pid_roll + pid_yaw;  
ESC_PWM[2] = PWM_channel_input[2] + pid_pitch - pid_roll - pid_yaw;  
ESC_PWM[3] = PWM_channel_input[2] - pid_pitch - pid_roll + pid_yaw;
```

Abbildung 121 PID Motorenkalkulation (O'Connor)

Alle Achsen sind um den Nullpunkt ausbalanciert. Ausschlussgebend für eine Addition oder Subtraktion ist, ob der Wert auf der jeweiligen Achse eine Positive oder Negative Zahl ist. Beispielsweise ist der repräsentative Wert der Pitch-Achse Negativ sollte die Drohne nach vorne fliegen. Um dies zu erreichen, müssen die Heckmotoren (M-1 und M-2) schneller Drehen und eine Addition wäre folglich. In der nachfolgenden Zeichnung können die Zuständigkeiten abgelesen werden:

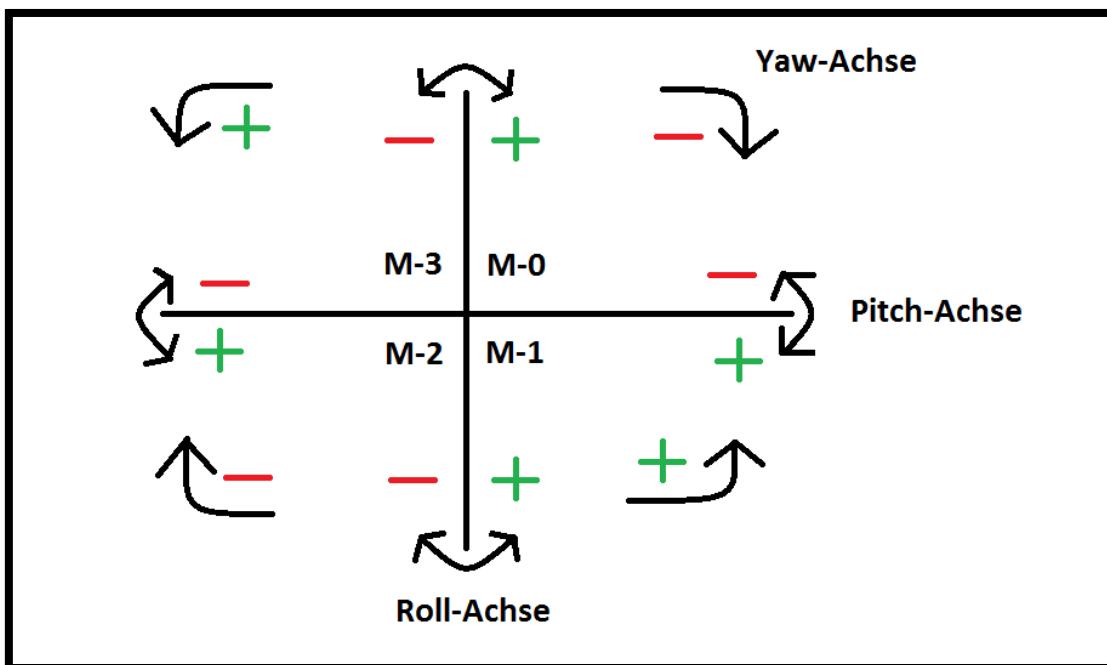


Abbildung 122 Motorenzuweisung (O'Connor)

8.3.3.7 Steuerung

Dadurch, dass die Drohne sich kontinuierlich selbst ausbalanciert mit der Abweichung vom Ist- und Sollwert, kann der Sollwert manipuliert werden, um die Drohne in der Luft steuern zu können. Das PWM Signal der einzelnen Kanäle muss umgewandelt werden von 1000us - 2000us zu Grad/s. Die IMU unterstützt mit den getätigten Einstellungen bis zu 500°/s. Ein Wert von -100°/s bis 100°/s reicht für die Steuerung. Die aktuelle Neigung (Offset) wird an dieser Stelle vom Sollwert subtrahiert. (Brokking, 2021)

```
pid_desired_roll_angle = map(PWM_channel_input[0], PWM_OFF, PWM_MAX, -100, 100) - roll_offset;
pid_desired_roll_angle = 0 - roll_offset;

pid_desired_pitch_angle = map(PWM_channel_input[1], PWM_OFF, PWM_MAX, -100, 100) - pitch_offset;
pid_desired_pitch_angle = 0 - pitch_offset;

pid_desired_yaw_angle = map(PWM_channel_input[3], PWM_OFF, PWM_MAX, -100, 100);
```

Abbildung 123 Manipulation des Sollwerts (O'Connor)

8.3.3.8 PID-Tuning

Im aller letzten Schritt bis zum autonomen Fliegen müssen die vordefinierten Werte des PID-Kontrollers K_p , K_i und K_d justiert werden. Als Ausgangslage werden folgende Werte definiert: (Brokking, 2021)

```
const float PID_KP = 0;
const float PID_KI = 0;
const float PID_KD = 3;
```

Abbildung 124 PID-Tuning Ausgangslage (O'Connor)

8.3.3.8.1 KD

Um den Wert von K_d zu definieren, wird die Drohne eingeschalten und in der Hand geführt. Der Wert wird immer um eins erhöht, bis die Drohne restlos überkompensiert. Anschliessend wird dieser bis zu einem stabilen Zustand dekrementiert. Der Wert des stabilen Zustands wird wiederum um 25% reduziert. (Brokking, 2021)

8.3.3.8.2 KP

Der Wert von K_p wird jeweils um 0.2 erhöht. Für das Testen kann bereits mit Vorsicht die Drohne abheben. Der Wert wird so lange inkrementiert, bis die Drohne überkompensiert. Anschliessend wird dieser um 50% reduziert. (Brokking, 2021)

8.3.3.8.3 KI

Der Wert von K_i wird jeweils um 0.01 erhöht. Ab dem Zeitpunkt der Überkompensation wird dieser wieder um 50% reduziert. (Brokking, 2021)

8.3.4 Release 3.1: Überarbeitung

Durch ein ausgiebiges Testen wurden kleinere Probleme festgestellt. Einige kleine Anpassungen müssen diesbezüglich getätigt werden für einen optimaleren Flug.

8.3.4.1 MPU-6050 Z-Achse invertieren

Ich bin davon ausgegangen, dass die Z-Achse des MPU-6050 im Uhrzeigersinn positiv wird und negativ für eine links Drehung. Im Datenblatt ist ersichtlich, dass dies nicht der Fall ist. Diesbezüglich muss die Z-Achse beim Einlesen invertiert werden:

```
gyro_axis[2] *= -1;
acc_axis[2] *= -1;
```

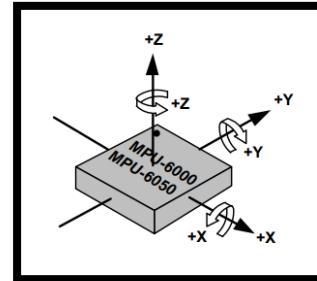


Abbildung 125 MPU-6050 Achsenbeschriftung (InvenSense.inc, 2013)

Abbildung 126 Z-Achse invertieren (O'Connor)

8.3.4.2 Neustart

Nachdem ein Flug stattgefunden hat und die Drohne eine Zwischenlandung gemacht hat, sollten die PID-Werte zurückgesetzt werden für ein erneutes Abheben ohne Vergangenheitsbewältigung.

```
void pid_reset() {
    roll_offset = 0;
    pitch_offset = 0;

    pid_i_roll = 0;
    pid_i_pitch = 0;
    pid_i_yaw = 0;

    pid_d_roll_last_error = 0;
    pid_d_pitch_last_error = 0;
    pid_d_yaw_last_error = 0;
}
```

Abbildung 127 PID-Werte zurücksetzen (O'Connor)

Um den aktuellen Flugstatus festzustellen für den Neustart, wurde eine kleine Methode implementiert:

```
void flight_state_update() {
    // Turn Off Motors
    if(flight_state == 2 && PWM_channel_input[2] < 1050 && PWM_channel_input[3] > 1950)
        flight_state = 0;

    // Start Motors
    if(PWM_channel_input[2] < 1050 && PWM_channel_input[3] < 1050)
        flight_state = 1;

    // Reset PID
    if(flight_state == 1 && PWM_channel_input[2] < 1050 && PWM_channel_input[3] > 1480) {
        flight_state = 2;
        pid_reset();
    }
}
```

Abbildung 128 Flugzustand (O'Connor)

Damit die Motoren starten, muss der linke Joystick sich in der unteren linken Ecke befinden und zurück zur Mitte geführt werden. Für das Ausschalten muss er sich in der unteren rechten Ecke befinden. Dadurch kann festgestellt werden, wann ein Neustart ausgelöst wird.

8.3.4.3 PID-Wert begrenzen

Eine Begrenzung des PID-Werts schützt vor Über- und Untersteuerung bei einem plötzlich starken Richtungswechsel durch z. B. Windböen in der PID-Kalkulation:

```
if (pid_i_roll > PID_MAX) pid_i_roll = PID_MAX;
else if(pid_i_roll < PID_MAX * -1) pid_i_roll = PID_MAX * -1;
```

Abbildung 129 PID-Begrenzung (O'Connor)

8.3.4.4 PWM-Signal für ESCs einschränken

Die Errechnung des PID-Werts, kombiniert mit der Motorenzuständigkeit, kann den ESC-Wert unter 1000us bringen, diesbezüglich muss die Einschränkung des PWM Signals auch vom Minimum überprüft werden:

```
// MIN PWM Failsafe
if(ESC_PWM[0] < PWM_MIN) ESC_PWM[0] = PWM_MIN;
if(ESC_PWM[1] < PWM_MIN) ESC_PWM[1] = PWM_MIN;
if(ESC_PWM[2] < PWM_MIN) ESC_PWM[2] = PWM_MIN;
if(ESC_PWM[3] < PWM_MIN) ESC_PWM[3] = PWM_MIN;

// MAX PWM Failsafe
if(ESC_PWM[0] > PWM_MAX) ESC_PWM[0] = PWM_MAX;
if(ESC_PWM[1] > PWM_MAX) ESC_PWM[1] = PWM_MAX;
if(ESC_PWM[2] > PWM_MAX) ESC_PWM[2] = PWM_MAX;
if(ESC_PWM[3] > PWM_MAX) ESC_PWM[3] = PWM_MAX;
```

Abbildung 130 ESC-PWM-Signal-Einschränkung (O'Connor)

8.3.4.5 Pufferzone

Das PWM-Signal des RC-Empfängers fluktuiert ständig, vor allem in der neutralen Position. Damit es nicht zu ungewollten Abschweifungen kommt, wird eine Pufferzone um die neutrale Position der jeweiligen Achsen erstellt.

```
if(PWM_channel_input[1] < 1492 || PWM_channel_input[1] > 1508) {
    pid_desired_pitch_angle = map(PWM_channel_input[1], PWM_OFF, PWM_MAX, -100, 100) - pitch_offset;
} else {
    pid_desired_pitch_angle = 0 - pitch_offset;
}
```

Abbildung 131 Pufferzone (O'Connor)

9 ENTSTEHUNGSZEIT

Die Entstehungszeit dieser Arbeit kann aus der nachfolgenden Tabelle entnommen werden:

Teilaufgabe	Entstehungszeit
Komponenten (Recherche, Aussuchen)	25h
Montage (inkl. Fehlschläge & Prototypen)	22h
Entwicklung (Quellcode, Testen, Tuning & Recherche)	54h
Dokumentation	70h
Total	171h

10 SCHLUSSWORT

Beim Eigenbau einer Drohne gibt es sehr viel zu beachten. Es genügt nicht, auf „Gut Glück“ zufällige Komponenten zu verwenden. Ausschlaggebend für die Entscheidung der Komponenten ist der Verwendungszweck. Renndrohnen sind kleiner und legen einen grossen Wert auf Geschwindigkeit und Agilität. Filmdrohnen hingegen favorisieren Stabilität und müssen schwere Kameras herum transportieren können.

Alle Motoren zusammen benötigen mindestens ein 2:1 Verhältnis von Auftrieb und Gewicht, um abheben zu können. Die entsprechenden Geschwindigkeitsregler müssen die Motoren mit ausreichend Strom beliefern können ansonsten kann es zu einer Unterversorgung führen. Die LiPo Batterie muss auch auf die Drohne abgestimmt sein. Ein Kompromiss zwischen Gewicht und maximale Flugzeit sowie ausreichende Belastbarkeit durch den hohen Stromkonsum der Motoren ist essenziell. Die Auswahl der Propeller verändert zusätzlich den Auftrieb und den Stromkonsum der Motoren und die Berechnungen müssen angepasst werden. Der Rahmen muss genügend Platz bieten für alle Komponenten und das richtige Material muss ausgesucht werden. Es sollte steif, aber gleichzeitig auch flexibel genug sein, um Vibrationen der Motoren kompensieren zu können. Die Vibrationen verfälschen die Messungen der Trägheitsmessungseinheit. Diese ist das essenzielle Element für das autonome Fliegen und bestimmt die dreidimensionale Orientierung. Der RC-Empfänger und Sender müssen aufeinander abgestimmt sein und im gleichen Frequenzbereich kommunizieren. Für die Weiterverwendung der übertragenen Daten mit dem Flugkontroller muss der Empfänger ein PWM Signal übermitteln können. Das Gehirn des Gesamten, der Arduino, ist für die gesamte Datenaufnahme, Berechnung und Ansteuerung der Geschwindigkeitsregler zuständig. Alle einzelnen Komponenten müssen nicht nur am Rahmen montiert werden, sondern auch durch Löten und Steckverbindungen in einen Schaltkreis integriert werden für die Stromaufnahme sowie den Austausch von Daten untereinander.

Die Steuerung kann grundsätzlich auf zwei Arten implementiert werden:

Im Release 2 wird das Radiosignal des RC Senders via RC Empfänger als PWM Signal dem Arduino übermittelt via den entsprechenden Channels. Der Arduino verwendet als Basisgeschwindigkeit der Motoren den Throttle-Wert. Für die Bewegung wird anschliessend das Signal für Pitch-, Roll- und Yaw-Manöver herunterskaliert und zu den entsprechenden Motoren addiert. Dadurch drehen die Motoren mit einer unterschiedlichen Geschwindigkeit und die Drohne führt die gewünschte Bewegung aus.

Im Release 3 wird das autonome Schweben implementiert. Zusätzlich zum Release 2 wird die dreidimensionale Orientierung von einer Trägheitsmessungseinheit errechnet. Mittels eines PID-Kontrollers wird der Fehlerwert vom Ist- und Sollzustand aller Achsen separat

Arduino Drohne selber bauen

kalkuliert und bei den Motoren die Geschwindigkeit angepasst für das autonome Stabilisieren. Durch die Eigenschaft, dass die Drohne sich kontinuierlich selbst balanciert, kann für die Bewegung der Sollwert manipuliert werden.

Drohnen werden in vier Hobbykategorien unterteilt. Die einzelnen Kategorien beinhalten Flugzeuge und Gleiter, Helikopter, Multikopter und VTOL. In diesem Projekt habe ich mich mit einem Quadrokopter (kat. Multikopter) tiefgründig auseinandergesetzt.

Es hat mir unglaublich viel Spass bereitet, das Ziel zu verfolgen und unzählige Hürden zu bewältigen. Es war mir nicht bewusst, wie gut die einzelnen Komponenten aufeinander abgestimmt sein müssen. Ich finde die Komplexität des autonomen Fliegens unglaublich spannend und ich war sehr verwundert beim Auseinandernehmen der PID-Logik, wie einfach und wenig Codezeilen dafür tatsächlich benötigt werden.

Der Inhalt meines Titelblatts soll das Endprodukt in vollen Zügen zur Geltung bringen. Zudem sollte es den Leser motivieren, nicht nur die Arbeit zu lesen, sondern auch selbst umzusetzen.



Abbildung 132 Fliegende Drohne (O'Connor)

11 LITERATURVERZEICHNIS

- Al-baghdadi, A., & Ali, A. A. (2019). *researchgate*. Abgerufen am 01. Oktober 2021 von https://www.researchgate.net/publication/340113111_An_Optimized_Complementary_Filter_For_An_Inertial_Measurement_Unit_Contain_MP6050_Sensor
- Arduino. (2021). *arduino*. Abgerufen am 28. August 2021 von <https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/>
- Arduino. (2021). *arduino*. Abgerufen am 24. September 2021 von <https://www.arduino.cc/en/reference/wire>
- Arduino. (2021). *playground arduino*. Abgerufen am 10. September 2021 von <https://playground.arduino.cc/Main/PinChangeInterrupt/>
- Brokking, J. (2021). *brokking*. Abgerufen am 01. Oktober 2021 von <http://www.brokking.net/imu.html>
- Brown, J. (06. 12 2019). *My DroneLab*. Abgerufen am 18. August 2021 von <https://www.mydronelab.com/blog/arduino-quadcopter.html>
- Brown, J. (12. August 2019). *My DroneLab*. Abgerufen am 18. August 2021 von <https://www.mydronelab.com/accessories/quadcopter-frames.html>
- Brown, J. (15. August 2019). *My DroneLab*. Abgerufen am 18. August 2021 von <https://www.mydronelab.com/blog/drone-uses.html>
- Brown, N. (02. September 2021). *kompulsa*. Abgerufen am 10. September 2021 von <https://www.kompulsa.com/introduction-pwm-pulse-width-modulation-works/>
- Bundesamt für Kommunikation. (04. Oktober 2021). *BAKOM*. Von <https://www.bakom.admin.ch/bakom/de/home/geraete-anlagen/besondere-geraete/flugmodelle-im-35-mhz-und-40-mhz-band.html> abgerufen
- Buschbaum, U. (18. Januar 2015). *ulrichbuschbaum*. Abgerufen am 30. September 2021 von <https://ulrichbuschbaum.wordpress.com/2015/01/18/using-the-mpu6050s-dlpf/>
- Charing, N., & Granath, E. (01. April 2020). *power-and-beyond*. Abgerufen am 25. September 2021 von <https://www.power-and-beyond.com/pid-controller--definition-and-explanations-a-915227/>
- electronoobs. (2021). *electronoobs*. Abgerufen am 24. September 2021 von https://electronoobs.com/eng_robotica_tut9_2.php

- electronoobs. (2021). *electronoobs*. Abgerufen am 24. September 2021 von https://electronoobs.com/eng_robotica_tut9_2_2.php
- Fisher, C. J. (14. Februar 2015). *analog*. Abgerufen am 25. September 2021 von <https://www.analog.com/en/app-notes/an-1057.html>
- FPVRacing. (2021). *FPVRacing*. Abgerufen am 17. September 2021 von <https://fpvracing.ch/de/content/7-grundsatzliche-funktion-quadrocopter-multicopter>
- IntoFPV. (18. September 2018). *IntoFPV*. Abgerufen am 28. August 2021 von <https://intofpv.com/t-kv-is-not-what-you-think-it-is>
- InvenSense.inc. (19. August 2013). *invensense*. Abgerufen am 10. September 2021 von <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>
- Liang, O. (17. Juni 2020). *OscarLiang*. Abgerufen am 10. September 2021 von <https://oscarliang.com/choose-propellers-mini-quad/>
- Liang, O. (19. Februar 2021). *OscarLiang*. Abgerufen am 10. September 2021 von <https://oscarliang.com/lipo-battery-guide/>
- Liang, O. (12. Juni 2021). *OscarLiang*. Abgerufen am 09. September 2021 von <https://oscarliang.com/quadcopter-motor-propeller/>
- Microchip Technology Inc. (2018). *Microchip*. Abgerufen am 02. September 2021 von <http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf>
- Millis, S. (2019). *The Dawn of the Drone: From the Back-Room Boys of World Ware One*. Oxford: Casemate Publishers. Abgerufen am 31. August 2021
- Nedelkovski, D. (04. Februar 2019). *HowToMechatronics*. Abgerufen am 17. September 2021 von <https://howtomechatronics.com/how-it-works/how-brushless-motor-and-esc-work/>
- Poole, N. (22. Mai 2012). *SparkFun*. Abgerufen am 28. August 2021 von <https://www.sparkfun.com/tutorials/348>
- Prof. Dr. Bendel, O. (13. Juli 2021). *Wirtschaftslexikon Gabler*. Abgerufen am 23. August 2021 von <https://wirtschaftslexikon.gabler.de/definition/drohne-54115/version-384618>
- Sparkfun. (2021). *learn sparkfun*. Abgerufen am 28. August 2021 von <https://learn.sparkfun.com/tutorials/what-is-an-arduino/all>

studyflix. (23. August 2019). *studyflix*. Abgerufen am 01. Oktober 2021 von <https://studyflix.de/informatik/pid-regler-1450>

Swiss Post. (23. März 2021). *Drones*. Abgerufen am 17. September 2021 von A vision has become reality: <https://www.post.ch/en/about-us/innovation/innovations-in-development/drones>

12 ABBILDUNGSVERZEICHNIS

Abbildung 1 Erste Flugdrohne (https://www.iwm.org.uk/history/a-brief-history-of-drones)	7
Abbildung 2 De Havilland Queen Bee seaplane L5894 (https://www.iwm.org.uk/history/a-brief-history-of-drones)	8
Abbildung 3 Flugzeug (https://pixabay.com/vectors/aircraft-drone-usa-airplane-plane-161415/)	9
Abbildung 4 Helikopter (https://pixabay.com/photos/helicopter-model-propeller-aircraft-1934758/)	9
Abbildung 5 Multikopter (https://pixabay.com/photos/drone-hexacopter-uav-rpas-aircraft-2168938/)	9
Abbildung 6 VTOL (https://wingtra.com/mapping-drone-wingtraone/vtol-drone/)	9
Abbildung 7 Quadrokopter Rotordrehrichtung (FPVRacing, 2021)	11
Abbildung 8 Quadrokopter Bewegungsachsen (FPVRacing, 2021)	11
Abbildung 9 Quadrokopter Bewegung durch Rotoren Geschwindigkeitsanpassung (FPVRacing, 2021)	12
Abbildung 10 BLDC Motor Konzept (https://electricalbaba.com/brushless-dc-bldc-motor/)	14
Abbildung 11 Magnetfeld einer Spule (https://quizlet.com/306997277/magnetic-field-around-a-solenoid-diagram/)	15
Abbildung 12 Kraftwechselwirkung (Nedelkovski, 2019)	15
Abbildung 13 Visualisierung der Magnetfelder (https://www.researchgate.net/figure/FE-model-of-a-12-pole-BLDC-prototype-motor-with-a-distributed-fully-pitched-stator_fig5_4099599)	15
Abbildung 14 Innen- und Aussenläufer BLDC Motor (https://www.tytorobotics.com/blogs/articles/how-brushless-motors-work)	16
Abbildung 15 Datenblatt Motor-Pull-Gewicht (https://www.amazon.com/iFlight-2750KV-Brushless-Racing-Quadcopter/dp/B07Y9JK2MW)	17
Abbildung 16 BLDC Motorgrösse (https://www.getfpv.com/motors/what-is-a-drone-motor.html)	18
Abbildung 17 CW und CCW Verbindung (O'Connor)	19

Abbildung 18 ESC Schaltplan und 360° Intervall (https://e-surfer.com/de/selbstbau-eines-elektrio-longboards/).....	20
Abbildung 19 LiPo Batterie (https://www.conrad.ch/de/p/droneart-modellbau-akkupack-lipo-11-1-v-1550-mah-zellen-zahl-3-75-c-softcase-xt60-1784861.html).....	21
Abbildung 20 LiPo-Zellen (Liang, OscarLiang, 2021).....	21
Abbildung 21 BLDC Motor Datenblatt Bedarfsermittlung (https://www.amazon.com/iFlight-2750KV-Brushless-Racing-Quadcopter/dp/B07Y9JK2MW)	22
Abbildung 22 Propeller Drehrichtung (https://www.dronefactory.ch/produkt/azure-vanover-limited-edition-prop-iron-grey/).....	24
Abbildung 23 RC-Empfänger (https://www.conrad.ch/de/p/reely-ht-4-handfernsteuerung-2-4-ghz-anzahl-kanaele-4-inkl-empfaenger-1310036.html)	25
Abbildung 24 MPU-6050 Achsenbeschriftung (InvenSense.inc, 2013)	26
Abbildung 25 MPU-6050 Datenblatt Beschleunigungsmesser Sensibilität (InvenSense.inc, 2013)	26
Abbildung 26 MPU-6050 Datenblatt Gyroskop Sensibilität (InvenSense.inc, 2013)	26
Abbildung 27 Arduino Uno (https://store.arduino.cc/products/arduino-uno-rev3/)	27
Abbildung 28 Arduino Uno Pinbelegung (https://www.circuito.io/blog/arduino-uno-pinout/).....	27
Abbildung 29 PWM Signal (O'Connor)	29
Abbildung 30 Schaltplan (O'Connor).....	30
Abbildung 31 Schaltplan Zoom (O'Connor).....	31
Abbildung 32 RC-Empfänger (Reely HR-4 Manual)	32
Abbildung 33 Prototyp 1 (O'Connor)	33
Abbildung 34 Interrupt (O'Connor)	34
Abbildung 35 Arduino Uno Pinbelegung (https://create.arduino.cc/projecthub/Alojz/arduino-pocket-game-console-a-maze-maze-game-63c225).....	35
Abbildung 36 PCMSK0 Register (Microchip Technology Inc, 2018)	35
Abbildung 37 PCIE0 Pin-Change-Interrupt (O'Connor)	35
Abbildung 38 PCINT0 Service Routine (O'Connor).....	36

Abbildung 39 PINB Adressen (Microchip Technology Inc, 2018).....	36
Abbildung 40 Pin 8 HIGH? (O'Connor)	36
Abbildung 41 Pin 9 HIGH? (O'Connor)	36
Abbildung 42 Aktive Input-Pins ermitteln (O'Connor).....	37
Abbildung 43 Notwendige Interrupt-Deklarationen (O'Connor).....	37
Abbildung 44 PWM-Signalmessung (O'Connor).....	38
Abbildung 45 Serienmonitor Initialisierung (O'Connor)	38
Abbildung 46 Ausgabe vom PWM Signal des Empfängers auf dem Serienmonitor (O'Connor)	38
Abbildung 47 RC-Kanalzuweisung (O'Connor).....	39
Abbildung 48 PORTD Register (Microchip Technology Inc, 2018).....	40
Abbildung 49 Arduino Pinbelegung (https://create.arduino.cc/projecthub/Alojz/arduino-pocket-game-console-a-maze-maze-game-63c225)	40
Abbildung 50 Pins 4:7 als Ausgabepins definieren (O'Connor).....	41
Abbildung 51 Zielzeitermittlung (O'Connor)	41
Abbildung 52 PORTD HIGH-Signal für alle Motoren (O'Connor).....	41
Abbildung 53 Beendung des PWM Signals (O'Connor)	42
Abbildung 54 250Hz Timer (O'Connor)	42
Abbildung 55 ESC Kalibrierung (O'Connor)	42
Abbildung 56 Wire.h Initialisierung (O'Connor)	43
Abbildung 57 PWR_MGMT_1 Register (InvenSense.inc, 2013).....	43
Abbildung 58 PWM_MGMT_1 Register verändern (O'Connor)	43
Abbildung 59 Gyroskop Register (InvenSense.inc, 2013)	44
Abbildung 60 Gyroskop Gradbereich (InvenSense.inc, 2013)	44
Abbildung 61 Gyroskop Messbereich definieren (O'Connor)	44
Abbildung 62 ACCEL Konfigurationsregister (InvenSense.inc, 2013)	44
Abbildung 63 Messbereich Beschleunigungsmesser (O'Connor).....	44
Abbildung 64 DLPF Konfigurationsregister (InvenSense.inc, 2013)	45
Abbildung 65 DLPF Bandbreite (InvenSense.inc, 2013)	45

Abbildung 66 DLPF definieren (O'Connor).....	45
Abbildung 67 Speicheradressen der Messwerte (InvenSense.inc, 2013)	46
Abbildung 68 MPU-6050 Daten Einlesen (O'Connor).....	46
Abbildung 69 MPU-6050 Daten speichern (O'Connor)	47
Abbildung 70 Einlesen MPU6050 Bit-Schiftung (O'Connor)	47
Abbildung 71 Ausgabe von Gyroskop-Daten (O'Connor)	47
Abbildung 72 Sensordatenkalibrierung ausrechnen (O'Connor)	48
Abbildung 73 Sensordatenkalibrierung anwenden (O'Connor)	48
Abbildung 74 Direktvergleich mit und ohne Kalibrierung (O'Connor)	48
Abbildung 75 Rahmenkomponenten (https://optimum-racing.ch/de/frame/1361-haoye-rc-x1-5-inch-freestyle-fpv-frame-kit.html)	49
Abbildung 76 Rahmen (O'Connor).....	50
Abbildung 77 Befestigung der Motoren am Rahmen (O'Connor)	50
Abbildung 78 ESC & Motor Kabelverbindung (O'Connor)	51
Abbildung 79 ESC & Motor Kabelverbindung Isolation (O'Connor)	51
Abbildung 80 Motorenkennzeichnung (FPVRacing, 2021).....	52
Abbildung 81 ESC Stromanschlussverbindung (O'Connor).....	52
Abbildung 82 Erstes Testen (O'Connor)	53
Abbildung 83 Zweites Testen (O'Connor)	53
Abbildung 84 Stromkreis von allen ESCs zusammenführen (O'Connor)	54
Abbildung 85 Stromkreis Batterie- und Arduino-Anschluss (O'Connor).....	54
Abbildung 86 LiPo Test (O'Connor).....	55
Abbildung 87 Erstellung eines Erdungskabel (O'Connor).....	55
Abbildung 88 Erdungskabel mit ESC-Signalkabel verbinden (O'Connor)	56
Abbildung 89 Arduino- mit ESC-Signalkabel verbinden (O'Connor)	57
Abbildung 90 Interferenzstrom Isolierung der Datenkabel (O'Connor)	57
Abbildung 91 RC-Empfänger Montage (O'Connor)	58
Abbildung 92 Montage des Dachs (O'Connor).....	58

Abbildung 93 Arduino am Rahmen befestigen (O'Connor)	59
Abbildung 94 Fahrwerk (O'Connor).....	60
Abbildung 95 MPU-6050 Orientierung (O'Connor)	60
Abbildung 96 Abschluss der Montage (O'Connor)	61
Abbildung 97 Quadrokopter Bewegung durch Rotoren Geschwindigkeitsanpassung (FPVRacing, 2021)	62
Abbildung 98 RC-Sender Kanalzuweisung (O'Connor)	62
Abbildung 99 PWM Signalumwandlung (O'Connor)	63
Abbildung 100 PWM Signalumwandlung Code (O'Connor)	63
Abbildung 101 ESC Ausgangsgeschwindigkeit (O'Connor)	63
Abbildung 102 Basiswert mit gewünschter Flugrichtungsbefehl addieren (O'Connor)	64
Abbildung 103 Motornummerierung (FPVRacing, 2021)	64
Abbildung 104 Pitch-Kalkulation (O'Connor).....	64
Abbildung 105 Yaw-Kalkulation (O'Connor)	64
Abbildung 106 Maximaler PWM-Signal Schutz (O'Connor)	65
Abbildung 107 Gyroskop Winkelberechnung (O'Connor)	66
Abbildung 108 Visualisierung des Winkelberechnungsproblem bei Drehung auf der Z-Achse (O'Connor)	66
Abbildung 109 Gyroskop Winkelberechnung mit Z-Achse (O'Connor).....	67
Abbildung 110 Beschleunigungsmesser Winkelberechnung (O'Connor)	67
Abbildung 111 Komplementärfilter (O'Connor)	68
Abbildung 112 PID-Regler (studyflix, 2019).....	68
Abbildung 113 PID-Regler verlauf (O'Connor)	69
Abbildung 114 Error-Wert $e(t)$ berechnen (O'Connor)	69
Abbildung 115 P-Rechnung (O'Connor).....	70
Abbildung 116 I-Rechnung (O'Connor).....	70
Abbildung 117 D-Rechnung (O'Connor)	70
Abbildung 118 $e(t)$ speichern (O'Connor)	70
Abbildung 119 PID Σ (O'Connor)	70

Abbildung 120 PID-Rechnung der Roll-Achse (O'Connor).....	70
Abbildung 121 PID Motorenkalkulation (O'Connor).....	71
Abbildung 122 Motorenzuweisung (O'Connor).....	71
Abbildung 123 Manipulation des Sollwerts (O'Connor)	72
Abbildung 124 PID-Tuning Ausgangslage (O'Connor).....	72
Abbildung 125 MPU-6050 Achsenbeschriftung (InvenSense.inc, 2013)	73
Abbildung 126 Z-Achse invertieren (O'Connor).....	73
Abbildung 127 PID-Werte zurücksetzen (O'Connor).....	73
Abbildung 128 Flugzustand (O'Connor)	73
Abbildung 129 PID-Begrenzung (O'Connor).....	74
Abbildung 130 ESC-PWM-Signal-Einschränkung (O'Connor)	74
Abbildung 131 Pufferzone (O'Connor).....	74
Abbildung 132 Fliegende Drohne (O'Connor).....	77
Abbildung 133 Interviewbestätigung (Wilks, OConnor).....	91

13 WEITERVERWENDUNG

Hiermit bestätige ich, dass die vorliegende Arbeit als Demonstrationsbeispiel verwendet werden darf.

.....
Ort, Datum

.....
Christopher O'Connor

14 EIGENSTÄNDIGKEITSERKLÄRUNG

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Die Stellen der Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Ich stimme zu, dass diese Arbeit mit einer Plagiatserkennungssoftware geprüft wird. Das Datenschutzrecht wird hierbei beachtet.

Ich versichere, dass ich diese Arbeit noch an keinen anderen Ort abgegeben habe.

.....

Ort, Datum

.....

Christopher O'Connor

15 REFLEXION

Nachfolgend meine Reflexion der Arbeit.

15.1 Beurteilung des Arbeitsprozesses

Ich bin sehr zufrieden mit dem Arbeitsprozess. Durch die frühzeitige Auseinandersetzung der Planung im Konzept konnte ich mir den chronologischen Ablauf ausmalen. Auch die Unterteilung des Gesamtziels hat mir sehr geholfen. Dadurch wusste ich immer, welche Teilaufgabe als Nächstes ansteht und welche voneinander abhängig sind.

15.2 Beurteilung der Arbeit als Ganzes

Meiner Meinung nach ist mein Endergebnis in vieler Hinsicht gut gelungen. Ich bin noch nicht komplett zufrieden mit dem Endresultat der Dokumentation. Ich habe mir so viel Wissen angeeignet und eine richtige Passion entwickelt gegenüber diesem Thema, dass es mir sehr schwierig gefallen ist, wo ich die Grenze setzen sollte. Hätte ich noch länger Zeit gehabt, wäre ich gerne noch detaillierter auf die einzelnen Komponenten eingegangen sowie auf verschiedene Kommunikationsprotokolle. Trotzdem bin ich sehr zufrieden mit meiner Drohne und würde am liebsten nochmals eine bauen mit zusätzlichen Sensoren und stetig die Fähigkeiten weiterentwickeln. Das gewonnene Wissen aus dieser Arbeit hat den Grundstein dazu gebildet.

15.3 Beurteilung des Lernerfolgs

Mit der Zielsetzung einer flugfähigen Drohne musste ich mir alles aneignen, was es nur so zu wissen gibt. Als kleines Beispiel: Ich musste sogar die Steuerung nachschlagen, welcher Joystick ist für welchen Befehl normalerweise zuständig. Ich kenne nun alle notwendigen Komponenten und ihre Funktionsweisen in- und auswendig. Auch wie sie untereinander kommunizieren können und wie dies manipuliert werden kann mittels eines Flugkontrollers. Ich bin sogar absolut am Ziel vorbeigeschossen. Ich war ein zu schlechter Pilot, um meine eigene Drohne in der Luft selber zu stabilisieren. Es ist mir gelungen, auch diese Hürde zu bewältigen und diesen Teil auf den Mikrocontroller auszulagern. Für die Zukunft nehme ich mir definitiv vor, ein Thema nicht am Stichtag auszuwählen ohne Vorwissen.

15.4 Umgang mit Schwierigkeiten

Es gab unzählig viele Schwierigkeiten. Ich wollte alles selbst entdecken und habe diesbezüglich auch die Prototypen entwickelt. Der erste Fehlgriff geschah beim ersten Prototyp. Durch einen Fabrikationsfehler kam es zu einem Kurzschluss im ESC und ich habe nicht genügend schnell gehandelt. Die geräucherte Batterie musste ersetzt werden. Nach dieser Lektion war ich extrem sorgfältig und hatte viele Sicherheitsmassnahmen eingeplant wie beispielsweise nur jeweils das Erdungskabel zusammen zu löten und das Stromkabel Erstmal nur provisorisch durch das blosse Drauflegen zu verbinden. Bei der Entstehung des Programmes war mein logisches Denken besonders stark gefordert, insbesondere bei der Implementation der Drehbewegungen: Pitch, Roll und Yaw im Release 2 und praktisch der gesamte Release 3. Bei aufgetretenen Schwierigkeiten musste ich oft meine Gedanken zur Ursache auf ein Blatt Papier bringen. Diesbezüglich befinden sich auch einige meiner Paint-Kreationen in dieser Dokumentation. Für mich sind die Schwierigkeiten das, was mich an der Programmierung so fesselt und motiviert. Die Stetige Suche nach Verbesserung und kreative Lösungssuche, ist für mich immer ein Highlight der Retrospektive.

15.5 Zeitmanagement

Ich hatte in diesem Projekt grosse Schwierigkeiten mit dem Zeitmanagement. Beginnend mit keinen Vorerfahrungen im Modellfliegen sowie Lieferengpässen der Komponenten und der langen Lieferungszeiten. Ich hatte anfänglich nur bis zum Release 2 geplant. Das Fliegen war zu diesem Zeitpunkt einerseits durch den schlechten Piloten und andererseits durch die Schwierigkeiten der Balance im dreidimensionalem Raum nicht möglich über einer Höhe von 10 cm. Das Experimentieren mit der Gewichtsverlagerung von der Batterie nach unten hat auch nicht genügend Verbesserungen gebracht. Der Zeitpunkt dieser Realisierung war auch nicht gerade am Anfang und ich wollte unbedingt eine funktionierende Drohne. Daraus ist der grösste und komplexeste Teil dieser Arbeit im Release 3 entstanden mit ein wenig Überstunden.

16 ANHANG

Der Anhang beinhaltet die Interviewbestätigung, das Interview und den Quellcode.

16.1 Interviewbestätigungen

Name	Wilks
Vorname	Simon
Tätigkeit	Co-Founder UAVenture AG
Tel.	062 544 93 12
E-Mail	simon@uaventure.com
Datum	29.09.2021



Abbildung 133 Interviewbestätigung
(Wilks, O'Connor)

Unterschrift zur Bestätigung für die Durchführung des Interviews:

A handwritten signature in blue ink, which appears to be "Simon Wilks".

16.2 Interview

Ich möchte mit dem Interview meine Internetquellen, welche viele verschiedene Meinungen aufweisen, von einer Fachperson verifizieren lassen, damit ich sicher sein kann, dass die Informationen, welche ich mir im Verlaufe dieser Arbeit bereits angeeignet habe, auch tatsächlich stimmen.

16.2.1 Einleitung

In diesem Interview werde ich Simon Wilks befragen. Er hat langjährige Programmierkenntnisse und arbeitete unter anderem bei Google als Engineering Manager. Seit acht Jahren hat er sich sein Hobby zum Beruf gemacht und die Firma UAVenture mitgegründet. Das Start-up entwickelt Autopiloten und Missionsplanungstools für landwirtschaftliche Drohnen.

16.2.2 Fragen & Antworten

How did you get started with drones?

I have been working with drones for the past 12 years. I was working at Google at the time as an Engineering Manager and I wanted to combine my two passions which were software development and flying. Prior to that, I had always been flying around with RC model planes, hand- and paragliders. Flight was always my passion. I wanted to take this further. Flying RC planes was kind of cool, but how could I bring these two worlds together? I then started looking around to see what kind of technology was around that I could start learning about this. I came across an open-source project called paparazziUAV and bought my first autopilot kit from them and basically what turned up in the post was a couple of PCBs, some pieces of electronics and a bunch of wires. That was it, I mean there were no instructions, nothing and the challenge started from there really. What do I do with this? How do I put this thing together? That was the kick-off point. I got then involved in that project a little bit. That was really the beginning of all of this.

From what I can see around here, you have come a long way!

Yeah, we have. I tell you what! A lot of sweat, tears and blood as they say.

Do you still work with the drones themselves, like still do the coding and assembling or are you busy with other work since you are the co-founder?

I am not sure if I should say fortunately or unfortunately. I tend to be very focused. Either I am doing 100% one thing or 100% another. If I have to split myself between the two, it really is a torture for me and that is actually a problem here because we are only four people. M. does all the flight-testing for us and assembling the drones and so on and so forth. Then we have D. and my co-founder A. doing the flight control. I do then what is left, which is the ground control station. The ground control station is an android-based app for a tablet and something that runs on Raspberry Pi's or NVidias, which we call a co-pilot or mission computer and there is an insane amount of development work to be done there. Basically, that is often like a 16h day or something all through the week and the weekends quit a lot. To get the work done, we need to do it all by ourselves. Which means, I do a lot of software development and then we just have bursts where we must do things for the company. That is paperwork or raising money, and that is sort of the point where we are now. I now shifted away the last 4 weeks to about 90% writing pitch decks and business plans and really doing an absolute minimum on software development to keep those parts I'm responsible for going. If we would have a bigger team, we could split it up more and I would probably even

be doing nothing in terms of coding. However, when you are a small company like this and still in start-up mode, which after 7 years we still are, you have to do everything. So, both. A lot of coding and, when it is needed, a lot of business stuff.

Your drone comes with a ground station. Is the calculation outsourced or does it happen directly on the drone itself?

The actual flight controller algorithms happen on the drone. To be precise on the autopilot. What you want is all the algorithms, so everything about stabilizing or setting its rate of motion or its position in pitch and roll and so on, to happen as fast as possible. Which means it has to be right on the autopilot, which controls then the servos or control services or whatever.

Then something like the co-pilot tends to take care of high-level things. It takes care of pulling the flight logs off the computer at the end of the flight and sending it up over the internet to our servers. It also helps to extend the AO capabilities. We have sometimes two or even three radars connected if you are flying over terrain to get distance measurements because our autopilot does not have enough ports. We do it there, bundle the results, do some processing, and feed it to the autopilot. Additional communication equipment like satellite or LTE communications can also happen over the co-pilot.

The ground control station is only responsible for creating a flight mission plan, which it then uploads, to the autopilot or doing configurations before a flight and doing sensor calibrations and then monitoring the flight in 3D. You can also interact with the drone during flight, so for example stop a current mission and tell the drone to head home with the android app or move around with the onscreen joysticks.

I have asked this question, because I have seen a ted talk which was quite old (8y) and they had a drone in a room and the room was fitted with a lot of cameras to determine the position of the drone and all algorithm calculations where outsourced on a computer.

That works in a room very nicely because you have a controlled environment. You have a lot of processing power on desktop computers or even more right. With our drone, which can fly beyond line of sight. For example, our customer in Germany flies its drones down in Malawi. They have 4G communication and that sometimes falls out and all they then got is satellite communication and then you're only sending 250bits every 10 seconds which only allows an update of position or a simple command for example to fly home. That is why

everything has to happen internally. These wonderful drones you have seen flying in the video are sort of the ideal or the future even. This can change, as we are getting closer to 5G networks, where if you are in range and have the very low latency and high bandwidths, you would be able to start shifting more things off the drone and to compute resources on the network. But it will all come down to latency. If it is not fast enough, the drone will have already travelled 10m or so by the time your response comes.

Your drones are fully autonomous after initial setup?

Yes exactly, either you give it a bunch of waypoints which you can define freely its latitude and long locations or even its altitude. You can just drag it with the app in 3D and pull the path around. For crop spraying or survey missions, when drones are doing some sort of scanning you create an outline, and the app will fill it with a survey pattern. You then just upload it and press start and off it goes.

I have noticed you also use PWM signals for intern communication from autopilot to the servos etc. PWM obviously has a delay time for the value it sends. Is that a problem considering one transmission can take up to 2us and you need to control each motor separately, which adds up pretty fast?

It depends on what you are looking at. In terms of commercial drones like we have here, it's not a problem because usually what you are dealing with by just looking at a simple multirotor: you have big props, so these here are around 25-30 inches in size with a low KV motor that are efficient but turn relatively slowly. Which means, every time you give it a change, a command to speed up or slow down, you got to deal with inertia. The time it needs to respond is much longer than the actual control time. Therefore, getting an even faster control will not even do you much.

So, where it does play a role, is when you are flying on tiny FPV drones. The little ones with 2k – 3k RPM motors, they are extremely agile and there, you notice the difference. There it counts. On big drones, it does not really make a big difference at all. Moreover, if you are looking at a fixed wing with a servo, it is like who cares. There the servos are running at like 50Hz, and it is more than enough. The entire aircraft is big and slow. Being able to move quickly does not do anything for you.

What have your biggest challenges been with the auto hovering?

I did not do too much on the low level of coding with this kind of problem. We were originally involved in the open source PX4, which is the fly stack for the Pixhawk and branched out from that and a lot of it, especially the multirotor code, already existed. I have tried similar projects like you are doing now but just with a dual rotor setup, just to experiment and I coded it on my phone! How far can you get with a mobile phone to control a multirotor?

The biggest problems really are, dealing with PID loops and tuning and the timing, so how fast are your control loops, do you have delays? Interrupts? Because if something happens there, it will throw everything else out. That is probably the hardest bit when you are doing everything from scratch if you want to make it work. Other things we still find today is, that sensors maybe stop giving you results for 100ms or 0.5s or just stop and sometimes this just happens because the way you communicate over I2C. I spent so much time in the past on tuning. These days I am happy because I do not have to do it anymore, the others are taking care of all that.

Another big challenge is with the IMU and the magnetic GPS sensor. What a great idea to have a sensor which interacts with magnetic fields on a drone potentially with magnetic parts and all the current from the motors and batteries which also generate small magnetic fields. Also, even flying by a building which is constructed of a lot of metal can give you interferences resulting in wrong data!

Where do you see drones in the Future?

I see drones everywhere. When we look at things right now, especially crop spraying, which means spraying fertiliser, pesticides and seeding. I see them taking over and replacing all the manual labour. It is happening now with our drones. Our partner in Taiwan who is doing a deal with an Indonesian company, they have rice fields down there and have these massive fields and they are walking around with hand sprayers. This will all be replaced by autonomous vehicles or drones in this case. I think what a big turning point of those types of drones will be, but also all and the biggest barrier at the moment is power source. Our biggest limitation is batteries. If you could have 100 times the power density, then you could fly a lot longer and carry a lot more weight. And this will have a huge impact on the industry. Similar to cars as well, they do also a better job than a person can. The sooner you get a human out of the loop the better. Autopilots will do a much more exacter job. What we see with drones as well in terms of collecting data or spraying is that it's much more precise. When you have precise positioning, you can fly exactly the lane that you need to and have

exactly the amount of overlap and do not go over the border, handle wind drift better and things like this. You save on chemicals applied. This also goes for drones collecting information. Building up a global photographic map of the entire world is a very intensive and expensive thing to do with humans with full sized airplanes and there is always the delay, I mean go and look on google maps now and look maybe where you are living right and it is probably two years old at least. If it were safe and cheap you would have drones fly over like once a week, then you could refresh it all the time. I think drones will get more efficient and obviously smarter and probably into some areas, you will not be as happy about. A Better opportunity to monitor people and all that kind of thing. But the limits will extend, I mean where do drones stop and aircraft begin? When you look at another main area where we are involved in is delivery. We have a company where we co-founded in South America which does drone deliveries through several countries there. You can start delivering Pizzas or things that weigh 3kg and then let's do 20kg and then it will be 50, 100, 200 and then you have full sized aircrafts. I think that barrier is really sort of fuzzy where drones stop and maned aircrafts end. Eventually, at some point in the future, you will not have no more pilots. Same as cars, right? You have self-driving cars because they are proven to be much safer than having a human behind the steering wheel and the same will happen with maned aircraft. You have these two worlds approach each other and merge in the middle.

If you have a drone that is big enough to fit a person inside, but the person does not interact with the drone itself, so the drone still flies autonomous, would you still consider it a drone?

For me it would be because if my drone here carries 11kg of pesticide or 60kg of pesticide or happens to be a person, it is still a payload. Therefore, it is still a drone.

16.2.3 Reflexion

Mein Besuch bei der UAVenture war unglaublich spannend. Ich durfte nicht nur das Interview halten, sondern habe auch einen Rundgang bekommen. Aus dem Gespräch während meiner Anwesenheit konnte ich die Leidenschaft von allen Mitarbeitern spüren. Der Umgang war durchaus sehr freundlich und ich fühlte mich sehr willkommen. Das Interview hat im Bastelraum stattgefunden. Wir waren umgeben von Experimenten sowie einsatzfähigen Drohnen. Meine Fragen konnten diesbezüglich nicht nur theoretisch, sondern auch praxisbezogen beantwortet werden. Ich konnte bereits während der ersten Frage feststellen, dass Herr Wilks sehr viele Erfahrungen mit Interviews hat. Dies hat vieles erleichtert. Meine Fragen wurden sehr ausführlich und ehrlich beantwortet. Das Interview hat auf Englisch stattgefunden. Für mich war das super, ich bin mit der englischen Sprache aufgewachsen und meine ganze Recherche war ebenfalls in Englisch. Dadurch kam es nicht zu Übersetzungsverwirrungen bei Nennungen von Komponenten oder anderen Fachbegriffen. Ich konnte während meines Aufenthalts sehr viel Neues lernen und mein Vorwissen evaluieren. Besonders beeindruckt hat mich die Wichtigkeit der niedrigen Latenz und seine Zukunftsvision. Seine Leidenschaft hat mich gepackt.

16.3 Quellcode

```
/*
 * Author: O'Connor Chris
 * Date: 27.10.2021
 * Version: 3.1
 */

#include <Wire.h>

volatile bool PCI_channel_state[4] = {false, false, false, false};
volatile int PWM_channel_input[4];
volatile long PCI_current_time, PCI_channel_timer[4];

unsigned long ESC_loop_time, current_time, ESC_PWM_target_time[4];

int ESC_PWM[4], gyro_axis_cal[3], gyro_axis[3], acc_axis[3], temp, flight_state;

float gyro_pitch_angle, pitch_offset, acc_pitch_angle,
      gyro_roll_angle, roll_offset, acc_roll_angle,
      gyro_yaw_angle, yaw_offset, gravity_vector;

float pid_p_pitch, pid_i_pitch, pid_d_pitch, pid_d_pitch_last_error, pid_desired_pitch_angle, pid_pitch, pid_error;
float pid_p_roll, pid_i_roll, pid_d_roll, pid_d_roll_last_error, pid_desired_roll_angle, pid_roll;
float pid_p_yaw, pid_i_yaw, pid_d_yaw, pid_d_yaw_last_error, pid_desired_yaw_angle, pid_yaw;

bool setup_complete = false;

const int PWM_MAX = 2000;
const int PWM_MIN = 1100;
const int PWM_OFF = 1000;

/***
 * PID Settings
 */
const float PID_KP = 0.0;
const float PID_KI = 0.0;
const float PID_KD = 3.0;
const int PID_MAX = 200;

void setup() {
    Wire.begin();
    TWBR = 12;

    DDRB |= B00100000; // Output Digital port 13
    DDRD |= B11110000; // Output Digital port 4, 5, 6 & 7

    digitalWrite(12, HIGH);

    init_mpu_6050();

    gyro_cal();

    /**
     * Setup PinChangeInterrupt
     */
    PCICR |= 0b00000001; // PCIE0 for PCMSK0
    PCMSK0 |= 0b00001111; // PCINT 0:3 => Input 8:11

    digitalWrite(13, HIGH);

    /**
     * Wait to exit Setup until Receiver Channel 3 (Throttle) is in lowest position
     */
    while(PWM_channel_input[2] < 1000 || PWM_channel_input[2] > 1020) {
        PORTD |= B11110000; // PORTD HIGH
        delayMicroseconds(1000);
        PORTD &= B00001111; // PORTD LOW
        delayMicroseconds(3000);
    }

    setup_complete = true;
    flight_state = 0;
    current_time = micros();

    digitalWrite(13, LOW);
}

void loop() {
    read_mpu_6050_data();
    offset_calc();
    flight_state_update();
    gyro_calc();
    pid_calc();
    esc_pwm_calc();
}
```

Arduino Drohne selber bauen

```
/**  
 * Set looptime to 250Hz (4ms)  
 */  
while(micros() < current_time + 4000);  
current_time = micros();  
  
PORTD |= B11110000; // SET HIGH on all ESCs  
  
// Calculate target time of each ESC  
ESC_PWM_target_time[0] = ESC_PWM[0] + current_time;  
ESC_PWM_target_time[1] = ESC_PWM[1] + current_time;  
ESC_PWM_target_time[2] = ESC_PWM[2] + current_time;  
ESC_PWM_target_time[3] = ESC_PWM[3] + current_time;  
  
while(PORTD >= 16) {  
    ESC_loop_time = micros();  
    if(ESC_loop_time >= ESC_PWM_target_time[0]) PORTD &= B11101111;  
    if(ESC_loop_time >= ESC_PWM_target_time[1]) PORTD &= B11011111;  
    if(ESC_loop_time >= ESC_PWM_target_time[2]) PORTD &= B10111111;  
    if(ESC_loop_time >= ESC_PWM_target_time[3]) PORTD &= B01111111;  
}  
  
/*  
 * Calculate Motorspeeds  
 */  
void esc_pwm_calc() {  
  
    if(flight_state == 2) {  
  
        ESC_PWM[0] = PWM_channel_input[2] - pid_pitch + pid_roll - pid_yaw;  
        ESC_PWM[1] = PWM_channel_input[2] + pid_pitch + pid_roll + pid_yaw;  
        ESC_PWM[2] = PWM_channel_input[2] + pid_pitch - pid_roll - pid_yaw;  
        ESC_PWM[3] = PWM_channel_input[2] - pid_pitch - pid_roll + pid_yaw;  
  
        /*  
         * PWM MIN Failsafe  
         */  
        if(ESC_PWM[0] < PWM_MIN) ESC_PWM[0] = PWM_MIN;  
        if(ESC_PWM[1] < PWM_MIN) ESC_PWM[1] = PWM_MIN;  
        if(ESC_PWM[2] < PWM_MIN) ESC_PWM[2] = PWM_MIN;  
        if(ESC_PWM[3] < PWM_MIN) ESC_PWM[3] = PWM_MIN;  
  
        /*  
         * PWM MAX Failsafe  
         */  
        if(ESC_PWM[0] > PWM_MAX) ESC_PWM[0] = PWM_MAX;  
        if(ESC_PWM[1] > PWM_MAX) ESC_PWM[1] = PWM_MAX;  
        if(ESC_PWM[2] > PWM_MAX) ESC_PWM[2] = PWM_MAX;  
        if(ESC_PWM[3] > PWM_MAX) ESC_PWM[3] = PWM_MAX;  
  
    } else {  
        /*  
         * Turn Motors off  
         */  
        ESC_PWM[0] = 1000;  
        ESC_PWM[1] = 1000;  
        ESC_PWM[2] = 1000;  
        ESC_PWM[3] = 1000;  
    }  
}  
  
void pid_calc() {  
  
    /*  
     * Roll  
     */  
    gyro_roll_angle = (gyro_roll_angle * 0.7) + ((gyro_axis[1] / 65.5) * 0.3);  
    pid_error = gyro_roll_angle - pid_desired_roll_angle;  
  
    pid_p_roll = PID_KP * pid_error;  
    pid_i_roll += PID_KI * pid_error;  
    pid_d_roll = PID_KD * (pid_error - pid_d_roll_last_error);  
  
    if(pid_i_roll > PID_MAX) pid_i_roll = PID_MAX;  
    else if(pid_i_roll < PID_MAX * -1) pid_i_roll = PID_MAX * -1;  
  
    pid_d_roll_last_error = pid_error;  
  
    pid_roll = pid_p_roll + pid_i_roll + pid_d_roll;  
  
    if(pid_roll > PID_MAX) pid_roll = PID_MAX;  
    else if(pid_roll < PID_MAX * -1) pid_roll = PID_MAX * -1;  
  
    /*  
     * Pitch  
     */  
    gyro_pitch_angle = (gyro_pitch_angle * 0.7) + ((gyro_axis[0] / 65.5) * 0.3);  
    pid_error = gyro_pitch_angle - pid_desired_pitch_angle;  
  
    pid_p_pitch = PID_KP * pid_error;  
    pid_i_pitch += PID_KI * pid_error;  
    pid_d_pitch = PID_KD * (pid_error - pid_d_pitch_last_error);  
  
    if(pid_i_pitch > PID_MAX) pid_i_pitch = PID_MAX;  
    else if(pid_i_pitch < PID_MAX * -1) pid_i_pitch = PID_MAX * -1;  
  
    pid_d_pitch_last_error = pid_error;  
  
    pid_pitch = pid_p_pitch + pid_i_pitch + pid_d_pitch;  
  
    if(pid_pitch > PID_MAX) pid_pitch = PID_MAX;  
    else if(pid_pitch < PID_MAX * -1) pid_pitch = PID_MAX * -1;
```

Arduino Drohne selber bauen

```
/*
 * Yaw
 */
gyro_yaw_angle = (gyro_yaw_angle * 0.7) + ((gyro_axis[2] / 65.5) * 0.3);
pid_error = gyro_yaw_angle - pid_desired_yaw_angle;

pid_p_yaw = 3 * pid_error;
pid_i_yaw += 0.02 * pid_error;
pid_d_yaw = 0 * (pid_error - pid_d_yaw_last_error);

if (pid_i_yaw > PID_MAX) pid_i_yaw = PID_MAX;
else if(pid_i_yaw < PID_MAX * -1) pid_i_yaw = PID_MAX * -1;

pid_d_yaw_last_error = pid_error;

pid_yaw = pid_p_yaw + pid_i_yaw + pid_d_yaw;

if (pid_yaw > PID_MAX) pid_yaw = PID_MAX;
else if(pid_yaw < PID_MAX * -1) pid_yaw = PID_MAX * -1;

}

/**
 * Reset PID values
 */
void pid_reset() {

roll_offset = 0;
pitch_offset = 0;

pid_i_roll = 0;
pid_i_pitch = 0;
pid_i_yaw = 0;

pid_d_roll_last_error = 0;
pid_d_pitch_last_error = 0;
pid_d_yaw_last_error = 0;

}

void offset_calc() {

/**
 * Calculate Basic Offsets
 * 0.0000611 = 1 / (250Hz / 65.5)
 *
 */
pitch_offset += gyro_axis[0] * 0.0000611;
roll_offset += gyro_axis[1] * 0.0000611;
yaw_offset = gyro_axis[2] * 0.0000611;

/**
 * Calculate Offsets with Yaw Offset
 * 0.01745555 = 1 / (3.142(PI) / 180°) ° to Radian
 */
pitch_offset -= roll_offset * sin(yaw_offset * 0.01745555);
roll_offset += pitch_offset * sin(yaw_offset * 0.01745555);

/**
 * Accelerometer calculations
 * 1 rad = 57.296°
 */
gravity_vector = sqrt((acc_axis[0]*acc_axis[0])+(acc_axis[1]*acc_axis[1])+(acc_axis[2]*acc_axis[2]));
if(abs(acc_axis[1]) < gravity_vector) acc_pitch_angle = asin((float) acc_axis[1] / gravity_vector) * 57.296;
if(abs(acc_axis[0]) < gravity_vector) acc_roll_angle = asin((float) acc_axis[0] / gravity_vector) * -57.296;

/**
 * Complementary Filter Gyrodrift
 */
pitch_offset = pitch_offset * 0.9996 + acc_pitch_angle * 0.0004;
roll_offset = roll_offset * 0.9996 + acc_roll_angle * 0.0004;

}

/**
 * Manipulate Offsets for Movement
 */
void pry_calc() {

if(PWM_channel_input[0] < 1492 || PWM_channel_input[0] > 1508) {
pid_desired_roll_angle = map(PWM_channel_input[0], PWM_OFF, PWM_MAX, -100, 100) - roll_offset;
} else {
pid_desired_roll_angle = 0 - roll_offset;
}

if(PWM_channel_input[1] < 1492 || PWM_channel_input[1] > 1508) {
pid_desired_pitch_angle = map(PWM_channel_input[1], PWM_OFF, PWM_MAX, -100, 100) - pitch_offset;
} else {
pid_desired_pitch_angle = 0 - pitch_offset;
}

pid_desired_yaw_angle = 0;
if(PWM_channel_input[2] > 1050) {
pid_desired_yaw_angle = map(PWM_channel_input[3], PWM_OFF, PWM_MAX, -100, 100);
}
}
```

Arduino Drohne selber bauen

```
/**  
 * Gyro Calibration  
 */  
void gyro_cal() {  
  
    for (int i = 0; i < 2000 ; i++) {  
        if(i % 200 == 0) digitalWrite(13, !digitalRead(13));  
  
        read_mpu_6050_data();  
  
        gyro_axis_cal[0] += gyro_axis[0];  
        gyro_axis_cal[1] += gyro_axis[1];  
        gyro_axis_cal[2] += gyro_axis[2];  
  
        PORTD |= B11110000; // Port 4, 5, 6 and 7 HIGH  
        delayMicroseconds(1000);  
        PORTD &= B00001111; // Port 4, 5, 6 and 7 LOW  
        delayMicroseconds(3000);  
    }  
  
    /**  
     * Calculate Gyro Offset  
     */  
    gyro_axis_cal[0] /= 2000;  
    gyro_axis_cal[1] /= 2000;  
    gyro_axis_cal[2] /= 2000;  
}  
  
void flight_state_update() {  
    /**  
     * Turn Off Motors  
     */  
    if(flight_state == 2 && PWM_channel_input[2] < 1050 && PWM_channel_input[3] > 1950)  
        flight_state = 0;  
  
    /**  
     * Start Motors  
     */  
    if(PWM_channel_input[2] < 1050 && PWM_channel_input[3] < 1050)  
        flight_state = 1;  
  
    /**  
     * Reset PID  
     */  
    if(flight_state == 1 && PWM_channel_input[2] < 1050 && PWM_channel_input[3] > 1480) {  
        flight_state = 2;  
        pid_reset();  
    }  
}  
  
void read_mpu_6050_data() {  
  
    Wire.beginTransmission(0x68);  
    Wire.write(0x3B);  
    Wire.endTransmission();  
    Wire.requestFrom(0x68, 14);  
  
    while(Wire.available() < 14);  
  
    acc_axis[0] = Wire.read()<<8|Wire.read();  
    acc_axis[1] = Wire.read()<<8|Wire.read();  
    acc_axis[2] = Wire.read()<<8|Wire.read();  
    temp = Wire.read()<<8|Wire.read();  
    gyro_axis[0] = Wire.read()<<8|Wire.read();  
    gyro_axis[1] = Wire.read()<<8|Wire.read();  
    gyro_axis[2] = Wire.read()<<8|Wire.read();  
  
    if(setup_complete) {  
        gyro_axis[0] -= gyro_axis_cal[0];  
        gyro_axis[1] -= gyro_axis_cal[1];  
        gyro_axis[2] -= gyro_axis_cal[2];  
    }  
  
    gyro_axis[2] *= -1;  
    acc_axis[2] *= -1;  
}  
  
void init_mpu_6050() {  
  
    // WakeUp  
    Wire.beginTransmission(0x68);  
    Wire.write(0x6B);  
    Wire.write(0x00);  
    Wire.endTransmission();  
    // Gyro Config  
    Wire.beginTransmission(0x68);  
    Wire.write(0x1B);  
    Wire.write(0x08);  
    Wire.endTransmission();  
    // Accel Config  
    Wire.beginTransmission(0x68);  
    Wire.write(0x1C);  
    Wire.write(0x10);  
    Wire.endTransmission();  
    // DLPF  
    Wire.beginTransmission(0x68);  
    Wire.write(0x1A);  
    Wire.write(0x03);  
    Wire.endTransmission();  
}
```

Arduino Drohne selber bauen

```
/**  
 * 4s delay to connect Battery properly without ESC BEEP  
 */  
for (int i = 0; i < 1000; i++) {  
    PORTD |= B1111000; // PORTD HIGH  
    delayMicroseconds(1000);  
    PORTD &= B0000111; // PORTD LOW  
    delayMicroseconds(3000);  
}  
  
}  
  
/**  
 * PinChangeInterrupt  
 */  
ISR(PCINT0_vect) {  
    PCI_current_time = micros();  
  
    // CH 1  
    if(PINB & B00000001) {  
        if(!PCI_channel_state[0]) {  
            PCI_channel_state[0] = 1; // Input 8 HIGH?  
            PCI_channel_timer[0] = PCI_current_time; // Remember state  
            PCI_channel_timer[0] = PCI_current_time; // Start timer  
        }  
        else if(PCI_channel_state[0]) {  
            PCI_channel_state[0] = 0; // Input 8 !HIGH? && changed from 1 to 0.  
            PWM_channel_input[0] = PCI_current_time - PCI_channel_timer[0]; // Remember state  
            PWM_channel_input[0] = PCI_current_time - PCI_channel_timer[0]; // Calculate Time (PWM)  
        }  
    }  
    // CH 2  
    if(PINB & B00000010) {  
        if(!PCI_channel_state[1]) {  
            PCI_channel_state[1] = 1;  
            PCI_channel_timer[1] = PCI_current_time;  
        }  
        else if(PCI_channel_state[1]) {  
            PCI_channel_state[1] = 0;  
            PWM_channel_input[1] = PCI_current_time - PCI_channel_timer[1];  
            PWM_channel_input[1] = 1500 - PWM_channel_input[1] + 1500;  
        }  
    }  
    // CH 3  
    if(PINB & B00000100) {  
        if(!PCI_channel_state[2]) {  
            PCI_channel_state[2] = 1;  
            PCI_channel_timer[2] = PCI_current_time;  
        }  
        else if(PCI_channel_state[2]) {  
            PCI_channel_state[2] = 0;  
            PWM_channel_input[2] = PCI_current_time - PCI_channel_timer[2];  
            if(PWM_channel_input[2] > 1800) PWM_channel_input[2] = 1800;  
        }  
    }  
    // CH 4  
    if(PINB & B00001000) {  
        if(!PCI_channel_state[3]) {  
            PCI_channel_state[3] = 1;  
            PCI_channel_timer[3] = PCI_current_time;  
        }  
        else if(PCI_channel_state[3]) {  
            PCI_channel_state[3] = 0;  
            PWM_channel_input[3] = PCI_current_time - PCI_channel_timer[3];  
        }  
    }  
}
```