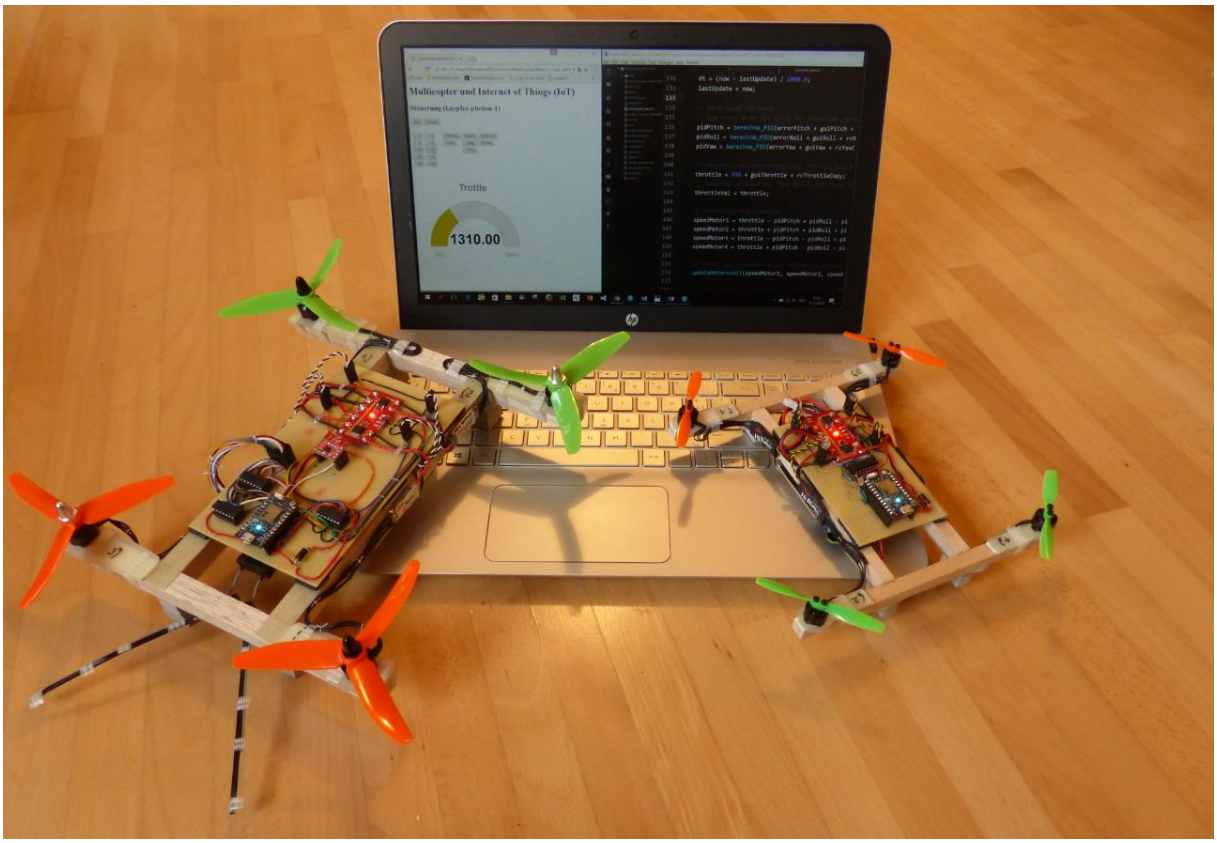
Kantonsschule Wiedikon Zürich, Herbst 2016

Internet Drohne
Bau und Programmierung



Emmanuel Küpfer, 6a

Maturitätsarbeit 2016

Betreuende Lehrperson: Patrick Spengler

Danksagung

Ein sehr grosser Dank gilt Patrick Spengler, der sich kurzfristig bereit erklärte die Betreuung meiner Maturitätsarbeit zu übernehmen.

Ebenfalls bedanke ich mich bei meinen Eltern und Irene Marti, die mich bei meiner Arbeit unterstützten.

Authentizitätserklärung

Ich bezeuge hiermit, dass die nachfolgende Arbeit von mir verfasst wurde und alle Quellen und Hilfsmittel korrekt angegeben wurden.

Inhaltsverzeichnis

1 Einleitung

1.1 Vielfältige Nutzung

Drohen, oder im Fachjargon Multicopter genannt, sind eine Erfindung des letzten Jahrzehnts. Mit einem funktionstüchtigen Microcomputer und einer Rahmenkonstruktion, die mehrere nach unten wirkende Propeller besitzt, können diese semiautomatischen bis vollautomatischen Flugobjekte für viele Zwecke eingesetzt werden. Ein gutes Beispiel dafür ist ein von ETH-Studenten entwickelter Multicopter, der Lawinenopfer aufspüren soll (Baumann, 2016). Solche Multicopter weisen einen simplen Aufbau und ein komplexes Programm auf, jedoch ist vielen Personen die Funktionsweise dieser Fluggeräte nicht bekannt.

Ich persönlich setze mich schon seit geraumer Zeit in meiner Freizeit mit Multicoptern auseinander. Allerdings verwende ich ausschliesslich Komponenten, die zusammengebaut, aber auf keinen Fall programmiert werden müssen. Dennoch interessiere ich mich für ihre Funktionsweise. Aus diesem Grund habe ich mich in meiner Maturaarbeit mit Multicopterbau und Programmierung beschäftigt.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist der vollständige Bau eines Multicopters, die Verwendung eines Microcomputers sowie die Nutzung eines Web-GUIs, mit welchem der Multicopter anschliessend über das Internet gesteuert werden kann. Dabei inbegriffen sind der Entwurf eines eigenen Rahmens wie auch das Zusammenlöten der ausgewählten Elektrobauteile und auch das Schreiben eines funktionierenden Programmcodes, der für einen stabilen Flug und der Steuerung verantwortlich ist. Jeder dieser Arbeitsschritte soll verständlich und kohärent dokumentiert werden.

2 Steuerung eines Multicopter

2.1 Flugachsen und Ansteuerung der Motoren

Der für diese Arbeit gebaute Multicopter weist eine Motorenanordnung in Form eines H auf. Bei dieser Konfiguration sind zwei Motoren parallel am vorderen Ende des Multicopters angebracht und zwei Motoren am hinteren Ende, wie in Abbildung 1 dargestellt. Im Gegensatz zu anderen Fluggeräten, wie beispielsweise dem Flächenflugzeug oder dem Hubschrauber, braucht ein Multicopter keine mechanisch verstellbare Steuerkomponente um seine Flugrichtung oder seine Flughöhe zu ändern. Einzig mit Variieren der Motorendrehzahl können Höhe und Richtung festgelegt werden.

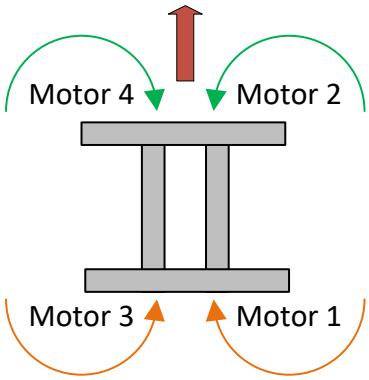


Abbildung 1 - Motorenanordnung

Multicopter mit vier Motoren, wie das der Fall in dieser Arbeit ist, besitzen ein Motorenpaar, das im Uhrzeigersinn dreht und eines, das im Gegenuhrzeigersinn dreht. Mit der gleichen Anzahl Motoren pro Umdrehungsrichtung wird gewährleistet, dass sich die auf den Multicopter übertragenen Drehmomente gegenseitig aufheben. Wie in der Abbildung 1 ersichtlich, ist die Umdrehungsrichtung der Motoren über die Diagonale gleich. Ein Erhöhen der Umdrehungszahl aller Motoren hat ein Steigen zur Folge. Eine Reduktion der Motorendrehzahl führt dazu, dass der Multicopter an Höhe verliert.

Für einen Vorwärtsflug – Drehung um die Querachse – muss die Drehzahl der beiden hinteren Motoren erhöht werden. Ein Analoges Verfahren gilt, wenn der Wunsch besteht, in die entgegengesetzte Richtung zu fliegen. Um eine Seitwärtsbewegung – Drehung um die Längsachse – in eine Richtung einzuleiten, wird wie bei der Drehung in der Querachse die Drehzahl eines der an den Seiten befindlichen Motorenpaars erhöht.
Bei Drehungen um Längs- und Querachse ist zu beachten, dass die Motorendrehzahl beider angesteuerten Motoren um den gleichen Wert erhöht werden muss, damit sich der Multicopter nicht infolge unterschiedlicher Drehmomente nach links oder rechts wegdreht. Ist ein Gieren – Drehung um die Gierachse – jedoch erwünscht, wird für eine Linksdrehung die Drehzahl der im Uhrzeigersinn drehenden Motoren und für eine Rechtskurve die Drehzahl der im Gegenuhrzeigersinn drehenden Motoren erhöht. In beiden Fällen wird das neutrale Drehmoment aufgehoben, damit der Multicopter in die gewünschte Richtung giert (Wikipedia, 2016). In Abbildung 2 werden die drei Flugachsen im Bezug zum Multicopter dargestellt. Der rote Pfeil gibt die Flugrichtung an.

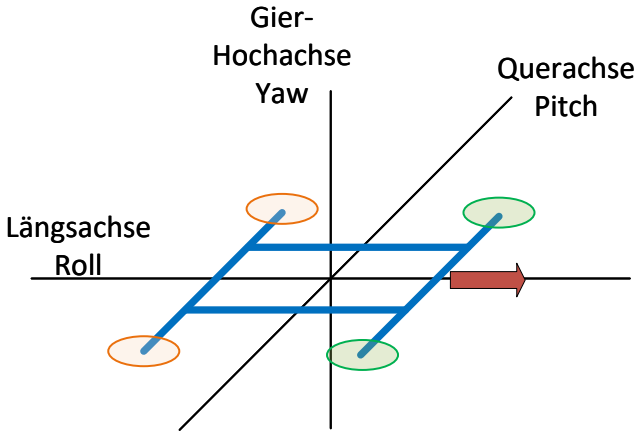


Abbildung 2 - Flugachsen

2.2 Sensor

Um die aktuelle Lage des Multicopters im Schwebeflug messen zu können, wird ein kombinierter Sensorbaustein, der sich aus Beschleunigungssensor, Kreiselsensor, auch Gyrometer genannt, und Kompass zusammensetzt, verwendet. Dieser Sensor ermöglicht es, dass ein Multicopter mit Hilfe von mathematischen Berechnungen autonom in der Luft ausharren kann. Der Beschleunigungssensor liefert Beschleunigungswerte, also Beschleunigung in der Einheit g. Der Kreiselsensor gibt die Winkelgeschwindigkeit, also eine Geschwindigkeit in Grad pro Sekunden, aus. Der Kompass misst die magnetische Flussdichte und liefert eine Ausgabe in milliTesla (mT). Diese Beschleunigungs- und Drehgeschwindigkeiten müssen in Winkelwerte umgerechnet werden, weil mit diesen die Differenz zur Horizontalen berechnet werden soll. Diese Winkelwerte werden danach in den Berechnungen, die im folgenden Kapitel «2.3 PID und Regelung der Motoren» erläutert werden, verwendet.

2.3 PID und Regelung der Motoren

Der PID-Regler ist dazu, da Umwelteinflüsse, die auf den Multicopter einwirken, auszugleichen. Bei der Kalibrierung oder nach einem Steuerinput wird ein neuer Sollwert der Lage für den Multicopter festgelegt. Aufgrund von Umwelteinflüssen wie beispielsweise Wind kann es vorkommen, dass der Multicopter seine festgelegte Lage nicht beibehalten kann und sich somit der Sollwert der Lage von dem aktuellen Istwert unterscheidet. Es entsteht also eine Abweichung, die sich aus der Differenz von Sollwert und Istwert berechnen lässt. Um dem Fehler entgegen zu wirken, wird ein PID Regler in das Programm implementiert (Busch, 2012), (Valvano, 2014). Der PID Regler besteht aus drei Komponenten, einem Proportionalanteil, einem Integralanteil und einem Differenzialanteil. In die Berechnung jeder Komponente fliesst ein konstanter Wert ein: K_p für Proportional, K_i für Integral und K_d für Differenzial. Der konstante Wert wird mit dem Fehler E(t) verrechnet. Das Ergebnis U ist der berechnete Korrekturfaktor zum Zeitpunkt t.

$$U(t) = Kp * E(t) + \int_0^t K(i) * E(t) * d(t) + \mathrm{Kd} * \frac{dE(t)}{d(t)}$$

Formel 1 - Die PID Gleichung

Beim Proportionalanteil wird der Fehler mit der Konstanten K_p multipliziert. Die Konstante K_p bestimmt genau, wie stark dem Fehler entgegengewirkt wird. Je grösser die Konstante K_p ist, desto stärker reagiert die Regelung auf die Differenz von Sollwert und Istwert. Ist die Konstante jedoch zu gross, reagiert der Regler zu stark auf den Fehler und der Multicopter beginnt zu schwingen, weil der Sollwert bei jeder Korrektur überschritten wird. Ein zu kleines K_p führt dazu, dass eine Korrektur zu langsam ausgeführt wird.
Nach einer Proportionalregelung oder auch durch ein schwaches Abdriften des Multicopters, können kleine Abweichungen ums Zentrum entstehen. Der Proportionalregler kann diese nicht vollständig ausgleichen. Es wird eine zweite Regelungseinheit, nämlich die Integralregelung benötigt.

Die Integralregelung berechnet aus der Summe der Fehler den Korrekturwerte. Sie wird so lange ausgeführt bis die Summe aller Fehler wieder den Wert 0 hat. Anders wie bei der Proportionalrechnung wird die laufende Zeit in diese Rechnung einbezogen. Je länger ein Fehler andauert, desto grösser wird der Korrekturwert und desto schneller wird die Sollposition wieder erreicht. Analog zu der Proportionalregelung muss auch bei der Integralregelung eine Konstante K_i festgelegt werden. Ein zu hohes K_i kann wie ein zu hohes K_p zu Schwingungen führen. Ein zu kleines K_i führt dazu, dass nach einer Abweichung ums Zentrum die Sollposition sehr langsam erreicht wird.

Proportional- und Integralregelungen korrigieren Fehler, bis der Sollwert wieder erreicht wird. Trotz gut ermittelten K_P und K_I Konstanten, kann es vorkommen, dass der Multicopter aufgrund seiner Trägheit über den Sollwert hinausschiesst. Es wird also eine dritte Regelung, nämlich die Differenzialregelung benötigt.

Die Differenzialregelung wirkt auf die Änderungsrate des Fehlers. Dazu wird der Unterschied aufeinander folgender Fehler durch die Zeitdifferenz dividiert. Wird der Quotient pro Rechendurchgang kleiner, bedeutet das, dass die Änderungsrate abnimmt. In diesem Fall hat die Differenzialregelung eine dämpfende Wirkung und verhindert ein Überschiessen. Wird der Quotient pro Rechendurchgang grösser, bedeutet das, dass die Änderungsrate des Fehlers zunimmt.
Die Konstante K_d bestimmt die Stärke des Differenzialreglers. Ein zu grosses K_d hat eine überdämpfende Wirkung auf das Proportional- und Integralglied. Infolgedessen nähert sich der Multicopter nur langsam dem Sollwert. Ein zu kleines K_d führt dazu, dass die Wirkung des Differenzialanteiles keinem allfälligen Überschwingen über den Sollwert hinaus entgegenwirken kann. Der Multicopter schwingt also über den Sollwert hinaus.

3 Bau des Multicopters

3.1 Bau des Rahmens

Für den Bau des Gerüsts des Multicopters wurde hartes Balsaholz ausgewählt, da es wenig wiegt, einfach zu bearbeiten ist und eine dämpfende Wirkung auf Vibrationen hat. Die 1cm x 1cm Holzstäbe wurden, wie auf der Abbildung 3 ersichtlich, in Form des Grossbuchstabes «H» verklebt. Inspiration war der Flitetest Knuckle H-Quad (Flitetest, 2016). Als Klebstoff wurde Heisskleber verwendet, weil sich dieser gut mit Holz verbindet und elastisch bleibt und so bei einem Absturz weniger schnell bricht. Aus einer 1mm dickem Glasfaserplatte wurden vier Motorenbefestigungen zugeschnitten und an die Unterseite des Multicopters befestigt. Diese Art der Befestigung erlaubt einen sehr kleinen Abstand der Propeller zur Oberseite der Arme. Befinden sich die Propeller nahe dem Schwerpunkt, verleiht das dem Multicopter ein neutrales Flugverhalten. Zur Verstärkung wurden bei den Verbindungsstellen, dort, wo die Längsstäbe auf die Querstäbe treffen, ebenfalls Glasfaserteile in Form von Rechtecken angebracht. Als Landegestell wurde ein Schaumstoff in vier Stücke zugeschnitten und diese an der Unterseite der Multicopterarme befestigt.

Abbildung 3 - Der Multicopterrahmen

3.2 Auswahl der Komponenten

3.2.1 Particle Photon

Leichtes Gewicht, WLAN-Verbindung und genügende Rechenleistung waren die Bedingungen bei der Auslese für eine geeignete Mikrokontroller Einheit. Der ausgewählte Particle Photon Baustein ist ein typischer Vertreter der «Internet of Things» Familie. Unter «Internet of Things», kurz «IoT», wird die Verwendung von Mikrocomputern bei alltäglichen Gebrauchsgegenständen verstanden. Das Particle Photon verfügt über eine WLAN Schnittstelle und hat deswegen einen kabellosen Zugang zum Internet. Diese Eigenschaft der drahtlosen Datenübertragung ist zwingend notwendig für das Vorhaben, einen Multicopter im Flug ansteuern zu können. Das Particle Photon ist Arduino kompatibel. Particle bietet zur Kontrolle ihrer Bausteine eine IT Infrastruktur an. Dank der Particle Cloud ist es leicht möglich, mit einem Web GUI eine Funktion auf dem Photon Baustein aufzurufen (Particle, Particle Cloud, 2016). In Abbildung 4 wird die Infrastruktur der Particle IoT Internetsteuerung schematisch dargestellt.

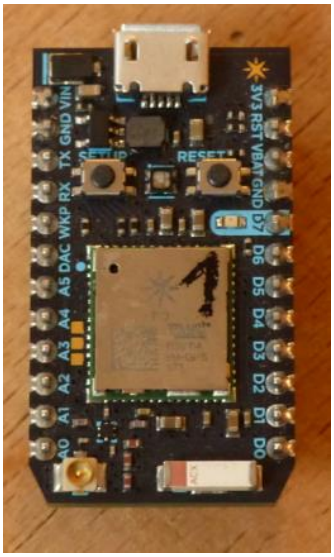


Abbildung 4 - Particle Photon Baustein

3.2.2 Photon IMU Shield

Das SparkFun Photon IMU Shield, Abbildung 5, ist ein Sensoren Brett, das einen LSM9DS1 Sensor mit 3-Achsen-Beschleunigungsmesser, 3-Achsen-Gyroskop, und 3-Achsen-Kompass besitzt. Der Particle Photon kann direkt mit auf dem IMU Shield eingesteckt werden. Zur Integration im Programmcode stellt der Hersteller eine Bibliothek zur Verfügung und ein Beispiel, wie die Bibliothek verwendet werden kann (SparkFun Electronics, 2016).

Abbildung 5 - Photon IMU Shield

3.2.3 Level-Shifter (3V to 5V und 5V to 3V)

Ein Problem, das beim Lesen des Datenblattes vom Particle Photon Bausteins auftauchte, ist seine geringe Output Spannung von 3,3 Volt. Die Regler der Motoren benötigen allerdings eine Spannung von 5 Volt. Also musste noch ein Level-Shifter Modul, das die Spannung von 3.3V auf 5V transferiert, zwischen Particle Photon und Regler eingebaut werden (Adafruit Industries, 2016). Das Level-Up Modul muss mit 5V betrieben werden, um die Signalspannungen des Particle Photons von 3.3V auf 5V anzuheben. Diese Spannung wird vom Stromverteiler Board (PDB) am 5V Ausgang abgezweigt. Das Ground Kabel (GND) des Level-Shifter Moduls und das der Signalkabel wird zusammengeführt und mit der GND des PDBs verbunden. Das verwendete Level-Shifter Modul ist in Abbildung 6 ersichtlich.

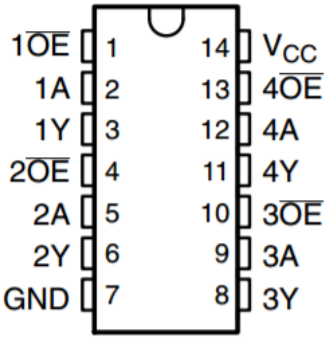


Abbildung 6 - 74AHCT125 - Level-Shifter (3V to 5V)

3.2.4 Motoren und Regler

Für dieses Projekt wurden die vier kleinstmöglichen Motoren ausgesucht, die zusammen das doppelte Gewicht des vollständigen Multicopters tragen können. Das hat den Vorteil, dass die Motoren nicht die volle Leistung erbringen müssen, wenn sich der Multicopter in der Schwebeposition befindet und dass sie so nicht heisslaufen. Ebenfalls bedeuten kleinere Motoren weniger Gewicht, was wiederum heisst, dass die Motoren weniger Leistungspotenzial erbringen müssen. Aufgrund dieser Aspekte wurden vier Motoren des Herstellers DYS ausgewählt (DYS B. , 2016). Die passenden Propeller sind die DYS PROP 3020 (Drone Matters, 2016). Das bedeutet der Propeller hat einen Durchmesser von drei Zoll und eine Steigung von zwei Zoll. Die Regler wurden entsprechend des Stromverbrauchs der Motoren ausgewählt. Bei den Reglern handelt es sich um vier 10 Ampere Regler der XM Serie, die ebenfalls aus dem Hause DYS stammen (DYS X. , 2016). Grund für diesen Entscheid ist die BLHeli Software, die darauf geladen ist (BlHeli, 2016). Diese Software erlaubt zu einem späteren Zeitpunkt ein müheloses Einstellen der Parameter mit Hilfe einer Applikation. In Abbildung 7 ist links der Motor mit montiertem Propeller und rechts der Regler zu sehen.

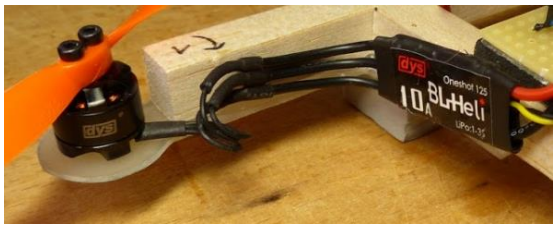


Abbildung 7 - Motor und Regler

3.2.5 Speisung

Um alle vier Regler, bzw. alle vier Motoren sowie die Kontroller-Einheit mit Strom zu versorgen, wurde eine Lantian Stromverteilungsplatine verwendet (Banggood.com, 2016). Solche Platinen sind im Modellbau üblich und heissen dort Power Distribution Board (PDB). Diese erlauben, alle Regler und die Steuereinheit mit Strom zu versorgen, ohne ein kompliziertes Kabel dafür zu benötigen. In Abbildung 8 ist ein vollständig verlötetes PDB abgebildet.

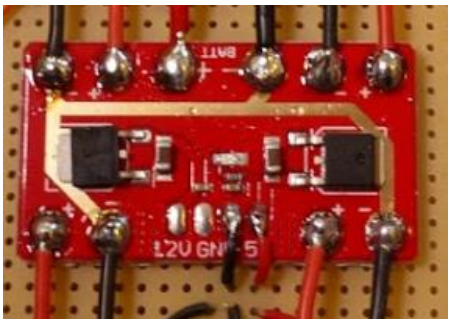


Abbildung 8 - Stromverteilerplatine

3.3 Bau der elektrischen Steuerung

Vor dem Zuschneiden der Lötplatine wurden alle Komponenten darauf platziert um, das Layout zu erfassen. Zuerst wurde das SparkFun Photon IMU Shield auf die Platine angebracht. Es muss darauf geachtet werden, dass die Achsen des Multicopters mit denen des Photon IMU Shield übereinstimmen. Das heisst konkret, dass die Querachse dem Nicken, die Längsachse dem Rollen und die Gierachse dem Gieren zugewiesen wird. Zur Hilfe der korrekten Ausrichtung des Photon IMU Shield befindet sich in seiner unteren linken Ecke eine kleine Legende der Achsenauslegung.

Die Regler werden mit einem Pulsweiten Modulation Signal (PWM) angesteuert. Ein solches Signal kann an den Particle Photon Pins: D2, D3, TX und RX ausgegeben werden. Die vier PWM Pins des Particle Photons werden mit dem Level-Shifter Modul verbunden. Vom Level-Shifter Modul werden die Signalkabel auf die Regler weitergeführt. Dazu werden auf der Platine vier dreifach Pin Servostecker verlötet, wobei der 5V Eingang vernachlässigt werden kann, weil die Regler keinen solchen aufweisen. Die Signalkabel, die vom Level-Shifter wegführen, werden mit den Signalkabeln der Regler verbunden. Die Ground Kabel der Regler werden gleichermassen zusammengeführt und mit dem Ground des PDBs verbunden.

Auf das Power Distribution Board wurden je die beiden Litzen der vier Regler angelötet. Diese werden direkt mit der Batteriespannung, die zwischen 12.6 Volt und 11.1 Volt beträgt, betrieben. Für den Betrieb wird eine im Modellbau übliche Lithium Polymer Batterie verwendet. Diese Batterien zeichnen sich durch eine hohe Leistungsdichte aus.

Das Particle Photon und das SparkFun IMU Shield werden auch mit 5V des PDBs betrieben. Die 5V Speisung muss am Pin VIN und der Ground am Pin GND verlötet werden. Sämtliche Ground Kabel der Regler, des Level-Shifter Moduls und der Kontrollbordeinheit werden zusammengeführt und mit der GND Lötstelle des PDBs verbunden. Wenn das Photon Particle über die USB Schnittstelle betrieben wird und gleichzeitig eine Batterie das System inklusive dem Photon Particle mit Strom versorgt, muss eine Überspannung vermieden wird, weil sonst die einzelnen Komponenten Schaden nehmen können. Das Particle Photon weist schon vorgefertigt einen Schutz auf. Um das Stromversorgungsboard zu schützen, muss zwischen ihm und der Kontrollbordeinheit eine Diode verbaut werden, sodass der Strom nur von der Versorgung zum Kontroller-Board fliessen kann, aber nicht umgekehrt. Es wurde eine Diode des Typs Schottky verwendet, weil das Photon-Board denselben Schutz aufweist (DigiKey, 2016). Der Abbildung 9 ist die Positionierung der Komponenten zu entnehmen.

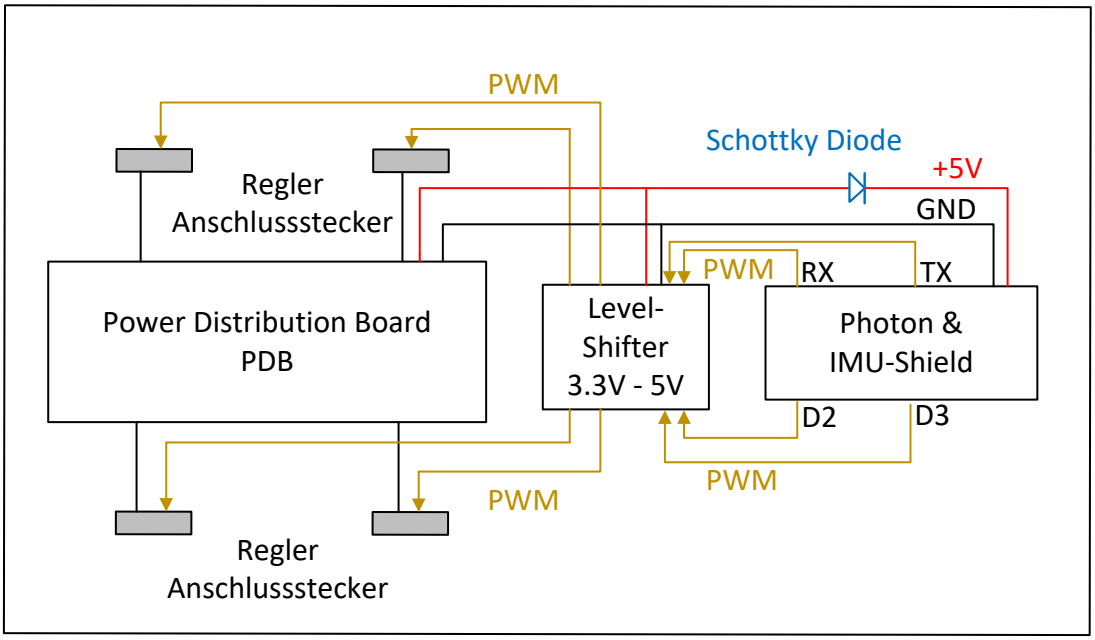


Abbildung 9 - Layout der Komponenten auf der Lötplatine

Die Motoren wurden mit Schrauben an den in Kapitel «3.1 Bau des Rahmens» beschriebenen Glasfaserteilen auf den Armen befestigt. Die Regler wurden mit Heisskleber an die Längsverstrebungen des Multicopters verklebt. Die jeweils drei Kabel der Regler und Motoren wurden miteinander verlötet. Die Drehrichtung der Motoren wird durch die Art der Verbindung miteinander bestimmt. Um die Drehrichtung eines Motors zu ändern, muss bloss die Verbindung von zwei der drei Kabeln vertauscht werden. Wenn jedoch, wie in diesem Projekt, Regler mit der Software BlHeli (BlHeli, 2016) verwendet werden, kann die Drehrichtung mit dieser Software verändert werden. Nachdem die Komponenten fertig verlötet worden war, wurde die Lötplatine auf den Rahmen des Multicopter geklebt. In Abbildung 10 und Abbildung 11 ist der flugfertige Multicopter dargestellt.

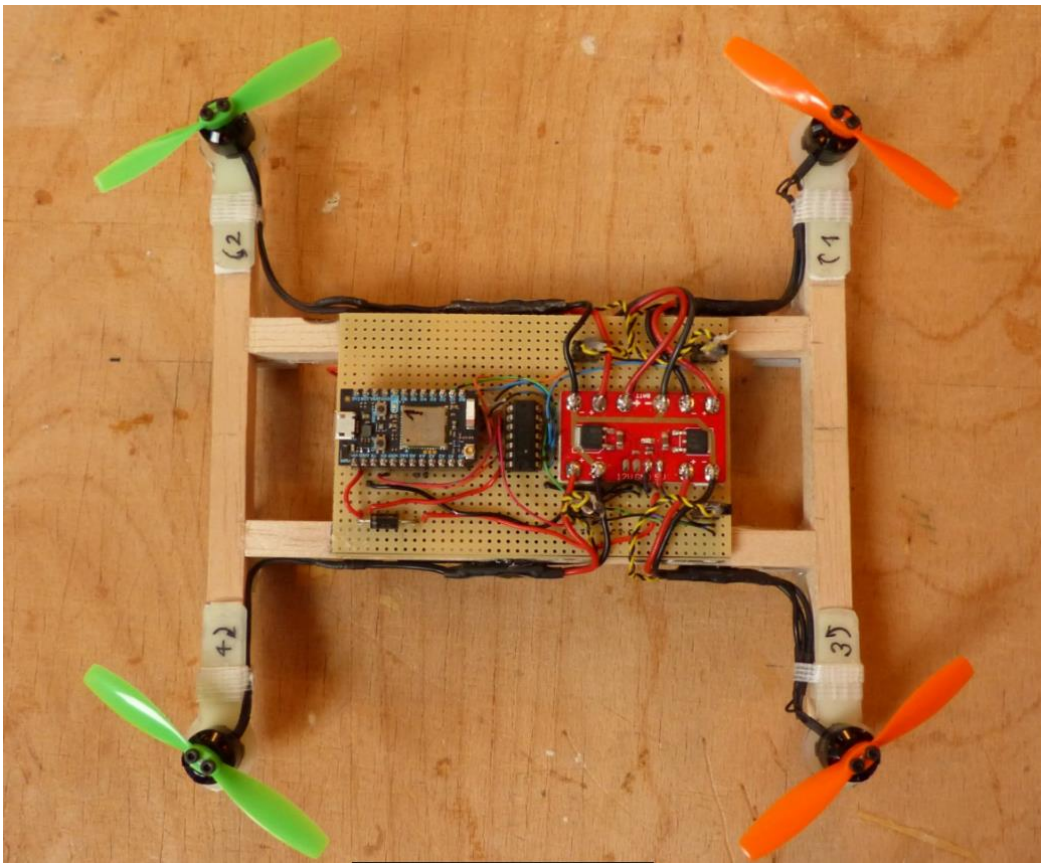


Abbildung 10 - Der fertiggestellte Multicopter

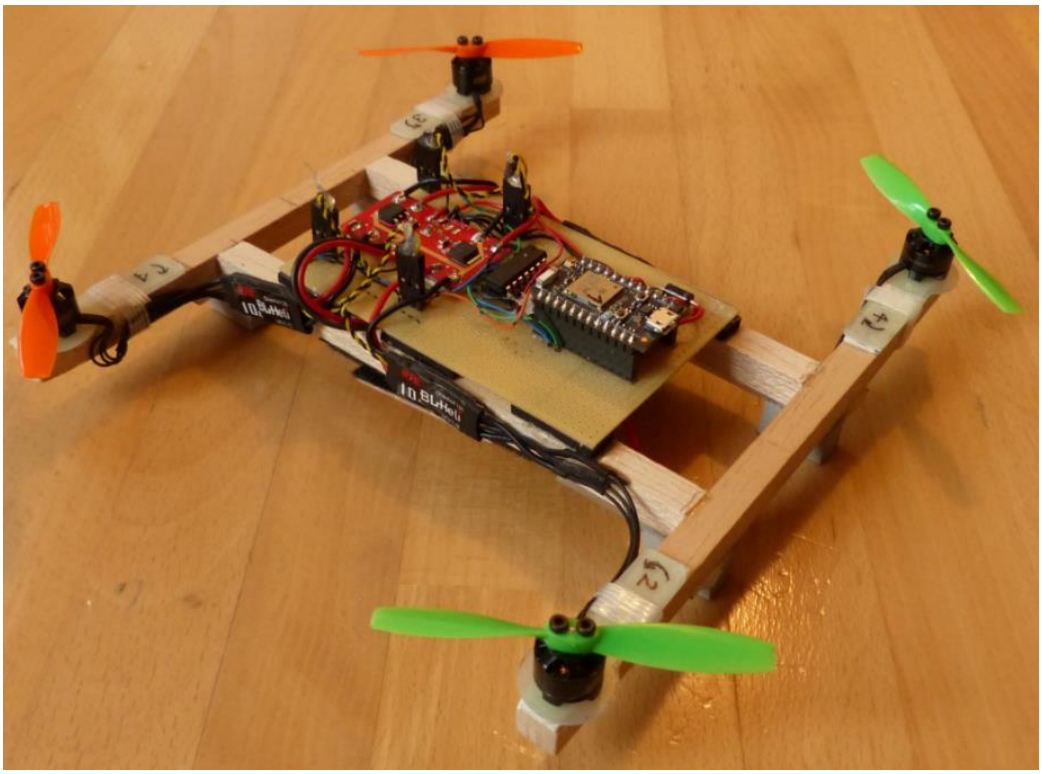


Abbildung 11 - Der fertiggestellte Multicopter

4 Schreiben des Programms

Beispiele für Multicopter Flugcontroller Steuerung sind einige auf dem Internet zu finden. Die Software des beliebten Naze32 Flugcontroller ist OpenSource, aber umfangreich und kompliziert (Diverse, Naze32, 2016). Weitere Quellen sind Quadrocopter Source Code From Scratch von Ben Ripley (benripley, Source Code, 2016) und LaunchPad Flight Controller von Kristian Sloth Lauszus (Lauszus, LaunchPad, 2016). Der Multicopter von Ben Ripley scheint noch nicht so gut zu fliegen, Kristian Sloth Lauszus hat für seinen Multicopter nicht Arduino verwendet. Aufgrund meiner Erfahrungen und meiner Auswahl der Bausteine, habe ich versucht, meine eigene Steuerung zu entwickeln.

4.1 Particle Dev Entwicklungsumgebung

Der Hersteller Particle bietet für seine Bausteine eine eigene Entwicklungsumgebung, die Particle Dev, an (Particle, Particle Dev, 2016). Mit der Particle Dev Entwicklungsumgebung werden sämtliche Programme geschrieben und via Internet auf die Particle Cloud übermittelt. Hier wird der Programmcode kompiliert und als Binärdatei auf den Mikrocontroller des Photon Particle Bausteins gesendet. In Abbildung 12 ist dieser Vorgang dargestellt.

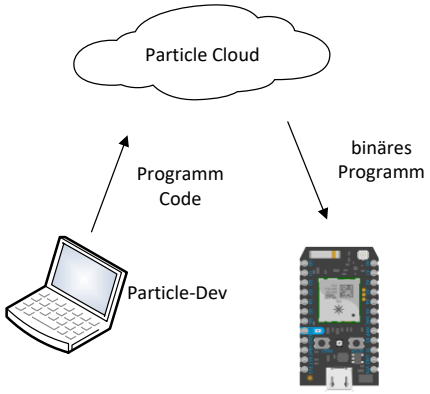


Abbildung 12 - Particle-Dev

4.2 Hauptprogramm

Die Struktur eines Arduino Programms besteht aus einem einmalig durchgeführten Setupblock und einem bis zum Programmende wiederholt aufgerufenen Loopblock.

4.2.1 Setup Funktion

In Abbildung 13 ist das Flussdiagramm des Setupblocks ersichtlich. Das Setup dient dazu, die Regler und die Sensoreinheit zu initialisieren. Für die Ansteuerung der Sensoren wurde die Bibliothek von SparkFun verwendet (SparkFun Electronics, 2016). Die Initialisierung erfolgt gemäss dem Beispiel dieser Bibliothek. Die beiden Schritte werden in den folgenden Kapiteln genau erklärt.

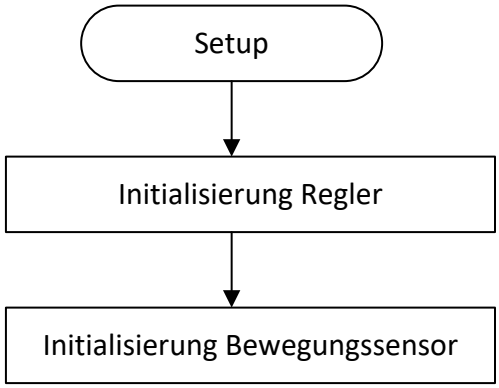


Abbildung 13 - Setup Ablauf

4.2.2 Loop Funktion

Das Particle Photon ruft die Loop Funktion wiederholt bis zum Programmende auf. Zuerst wird geprüft, ob die Sensoren bereit sind. Falls sie es nicht sind, wird der Loop beendet und die Funktion wird wieder vom Particle Photon aufgerufen. Wenn die Sensoren bereit sind, werden in einem ersten Schritt die aktuellem Sensorwerte ausgelesen. In der Fehlerberechnung werden diese Werte vom Sollwert subtrahiert. Das ergibt den Fehler. Im PID Block wird aufgrund des Fehlers ein neuer Korrekturwert berechnet. Diese Berechnung erfolgt für die drei Achsen Pitch, Roll und Yaw. In diesem Programmteil werden ebenfalls die externen Steuerinputwerte des Web-GUIs verarbeitet. Für die Berechnung der Motorendrehzahl werden alle ermittelten Korrekturwerte der Horizontalsteuerung und Schubwerte mit dem korrekten Vorzeichen zusammengezählt. In einem letzten Schritt erfolgt die Ansteuerung der Regler und somit der Motoren. Alle diese Schritte sind als Flussdiagramm in Abbildung 14 aufgeführt und werden in den folgenden Kapiteln genau erklärt.

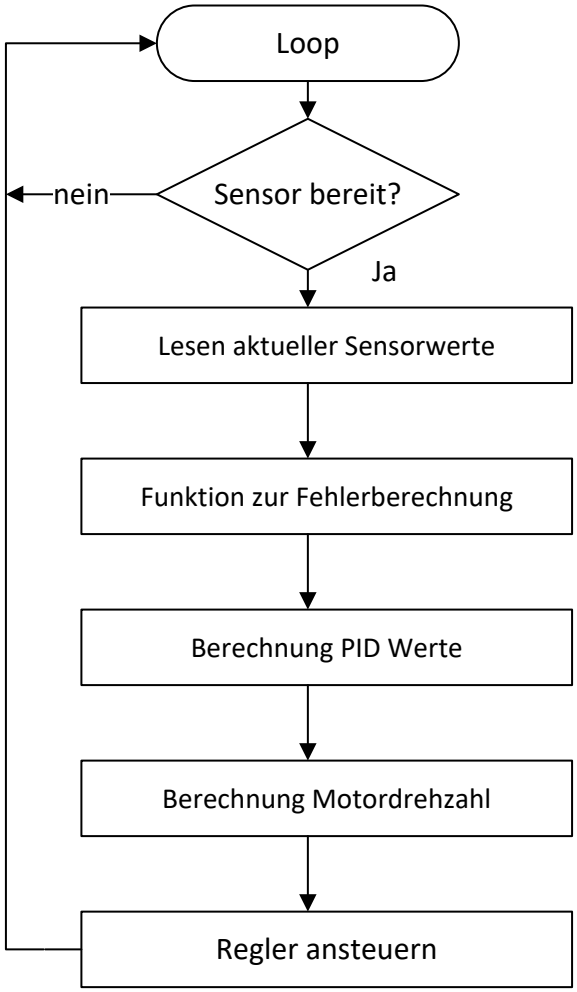


Abbildung 14 - Multicopter Steuerung Ablauf

4.3 Aufteilung in Nebendateien

Um dem Programmcode eine klare Übersicht zu verleihen, wurden die einzelnen Funktionen in Nebendateien ausgelagert. Im Hauptprogramm werden diese Funktionen aufgerufen. Dabei werden ihnen die zu verarbeitenden Variablen übergeben. Nach Ausführung einer Funktion wird dem Hauptprogramm die verarbeiteten Werte zurückgegeben. In der Tabelle 1 sind alle Nebendateien des Hauptprogramms aufgeführt.

Datei	**Funktion**
startup_calibration.cpp, / .h	Bestimmung der Startwerte für Pitch, Roll und Yaw
SparkFunLSM9DS1.cpp / .h	Bibliothek von SparkFun zum Lesen der Sensorwerte
quaternionFilter.cpp, / .h	Bibliothek von SparkFun zur Berechnung von Drehungen
Sensor.cpp / .h	Berechnung der Winkelwerte mit Hilfe von SparkFunLSM9DS1.cpp / .h und quaternionFilter.cpp, / .h
Error.cpp, / .h	Berechnung der Abweichung

Pid.cpp / .h	Berechnung der PID Werte für alle Achsen
Throttle.cpp / .h	Einlesen der Internetsteuerung sowie Arm und Disarm Funktion

Tabelle 1 - Hauptprogramm und Nebendateien

4.4 Initialisierung der Motoren

Regler müssen beim Einschalten mit einem Leerlaufsignal aktiviert werden. Mit der Funktion `initMotors()`, Abbildung 15, werden dafür auf die Motoren Pins Pulsweiten Modulationssignale (PWM) von 400 Hz mit einer Pulsbreite von einer Millisekunde gesendet. PWM wird im Kapitel «4.9.1 PWM zu Ansteuerung der Motoren» erklärt.

```
// Initialisierung Motoren PWM Pins
void initMotors()
{
  analogWrite(D2, 100, 400);
  analogWrite(TX, 100, 400);
  analogWrite(D3, 100, 400);
  analogWrite(RX, 100, 400);
}
```

Abbildung 15 - Initialisierung der Motoren

4.5 Initialisierung vom Startwert

Um einen Startwert der Sensoren für den Multicopter zu erhalten, wird bei der Aktivierung ein Ausgangswert für den Schwebeflug berechnet. In der Funktion `kalibrierung()` werden die aktuellen Sensorwerte der Achsen Pitch, Roll und Yaw in einer `for` Schleife zehnmal ausgelesen und aufsummiert. Diese Summe wird durch zehn dividiert, um daraus einen Mittelwert zu errechnen. Das Resultat wird an das Hauptprogramm zurückgegeben. Mit der Durchschnittsberechnung sollen allfällige Ausreisser der Sensoreinheit ausgeglichen werden.

4.6 Sensor Daten lesen und verarbeiten

Im Kapitel «2.2 Sensor» wird beschrieben weshalb ein Sensor für einen Schwebeflug verwendet wird. Da das Auslesen von Sensoren sehr kompliziert ist, war bald klar, dass ein Sensor mit einer passenden Bibliothek eingesetzt werden muss. Zum IMU Shield von SparkFun (SparkFun Electronics, 2016) ist eine solche Bibliothek erhältlich. Die Bibliothek enthält auch ein Beispiel, wie anhand der Beschleunigungswerte, Kreiselsensorwerte und Kompasswerte Winkel berechnet werden können. Aus dem Programmierbeispiel wurden die passenden Codeblöcke kopiert und in die Nebendatei Sensor.cpp eingefügt.

4.7 Fehlerberechnung

Damit die PID Rechnung ausgeführt werden kann, muss für die Achsen Pitch Roll und Yaw der Fehler berechnet werden. Dazu wird in der Implementation, Abbildung 16, der Istwerten vom Sollwert subtrahiert. Diese Differenz wird als Errorwerte (E_t_) an das Hauptprogramm zurückgegeben.

```
//Fehlerberechung
float berechne_E_t_(float ist_wert, float soll_wert)
{
  float E_t_;
  E_t_ = soll_wert - ist_wert;

  return E_t_;
```

```
}
```

Abbildung 16 - Implementation der Fehlerberechnung

4.8 PID Berechnung

Die Funktion `berechne_PID()` wird nach der Fehlerberechnung im Hauptprogramm für jede Steuerachse aufgerufen. Der Implementation, Abbildung 17, werden die zuvor berechneten Fehler und Zeitdifferenzen übergeben. Die PID Berechnungen werden gemäss der Beschreibung im Kapitel «2.3 PID und Regelung der Motoren» ausgeführt. Die Summen der PID-Berechungen (U_T_) werden dem Hauptprogramm zurückgegeben und den Variablen pidPitch, pidRoll und pidYaw zugewiesen.

```
float berechne_PID(float E_t_, float dt, int achse)
{
  //Definition PID Variabeln
  float P_t_ = 0;
  float I_t_ = 0;
  float D_t_ = 0;
  float U_t_ = 0;

  //Proportionalanteil
  P_t_ = Kp * E_t_;

  //Integralanteil
  //Multiplikaton mit konstantem Faktor dt wegen konstanter Looptime weggelassen
  I_t_ = I_t_1[achse] + Ki * E_t_; // * dt;
  I_t_1[achse] = I_t_;

  //begrenze Integralanteil
  if (I_t_1[achse] > 10) {
    I_t_1[achse] = 10;
    I_t_ = 10;
  }
  if (I_t_1[achse] < -10) {
    I_t_1[achse] = -10;
    I_t_ = -10;
  }

  //Differenzialanteil
  //Division durch konstanten Faktor dt wegen konstanter Looptime weggelassen
  D_t_ = Kd * (E_t_ - E_t_1[achse]); // / dt;
  E_t_1[achse] = E_t_;

  //PID-Summe
  U_t_ = P_t_ + I_t_ + D_t_;

  //Rückgabewert
  return U_t_;
}
```

Abbildung 17 - Implementation des PID Reglers

4.9 Regulierung der Motoren

Im Setup des Hauptprogramms werden den Motoren mit der Funktion `initMotors()` vier verschiedene Ports (D2,D3,TX und RX) zugewiesen und die Pulsbreite sowie Frequenz des Signals bestimmt. Die eigentliche Ansteuerung der Motoren erfolgt in dem externen Web GUI mittels Druckknöpfe. Durch das Web GUI wird auf dem Particle Photon die Funktion `speedMulticopter(String speedCommand)` aufgerufen und der übergebene Wert mit der Funktion `atoi(speedCommand)` der Variable throttle zugewiesen.

Wie im Kapitel «2.1 Flugachsen und Ansteuerung der Motoren» beschrieben, werden bei der Korrektur der Fluglage die Leistungen zweier Motoren gesteigert und die Drehzahl der verbleibenden Motoren proportional zu den gesteigerten gesenkt. Eine Möglichkeit um zu bestimmen, welche Leistung ein Motor bei einer Korrektur annehmen muss, ist die Analyse der Winkelwerte (Pitch, Roll und Yaw) in der Particle Dev Konsole. Der Multicopter wird um alle drei Flugachsen geneigt. Pitch, Roll und Yaw nehmen entweder positive oder negative Werte an. Die Vorzeichen der errechneten PID Werte (pidPitch, pidRoll und pidYaw) müssen so angepasst werden, dass bei einem Fehler die Leistung der richtigen Motoren gesteigert oder gesenkt wird. Das auf die vier Regler gesendete Signal setzt sich aus der Summe des manuell eingegebenen throttle Wertes und der errechneten PID Werte pidPitch, pidRoll und pidYaw zusammen. Wie in Abbildung 18 ersichtlich, werden für die vier Motoren berechneten Summen den Variablen speedMotor1-4 zugeordnet. Diese Variablen sind Parameter der Funktion `updateMotoresAll()`. In jener Funktion werden mit dem Aufruf `analogWrite()` die Werte der Variablen speedMotor1-4 auf die Regler geschrieben.

```
//Berechne Motordrehzahl
speedMotor1 = throttle - pidPitch + pidRoll - pidYaw;
speedMotor2 = throttle + pidPitch + pidRoll + pidYaw;
speedMotor3 = throttle - pidPitch - pidRoll + pidYaw;
speedMotor4 = throttle + pidPitch - pidRoll - pidYaw;

// Aufruf der Funktion um auf Motoren zu schreiben
updateMotoresAll(speedMotor1, speedMotor2, speedMotor3, speedMotor4);
```

Abbildung 18 - Berechnung der Motorenansteuerung

4.9.1 PWM zu Ansteuerung der Motoren

Um die Drehzahl der Motoren zu bestimmen, werden die Regler mit einem Pulsweitenmodulationssignal angesteuert (Arduino, PWM, 2016). Eine Pulsbreite von einer Millisekunde bedeutet Stillstand. Eine

Pulsbreite von zwei Millisekunden bedeutet Vollgas. Die Zeit zwischen zwei Pulsen wird Pulspause genannt. Die Pulsbreite und die Pulspause ergeben zusammen die Periodendauer. In der Abbildung 19 ist ein standardisiertes PWM Signal für eine Servo- oder Motorenansteuerung ersichtlich. Bei einer Periodendauer von 20ms wird mit einer Frequenz von 50Hz auf die Regler geschrieben. Da die Loop Time des Hauptprogramms 400Hz beträgt, könnte nur bei jedem achten Umlauf ein neues Signal auf die Reger übermittelt werden. Aus diesem Grund wurde bei der Initialisierung der Motoren mit der Funktion `analogWrite()` das PWM Signal neu definiert. Wie in Abbildung 20 ersichtlich, wurde die Periodendauer auf 2.5 ms verringert, was eine Schreibfrequenz von 400 Hz erlaubt. Das ist die maximale Schreibgeschwindigkeit der Regler.

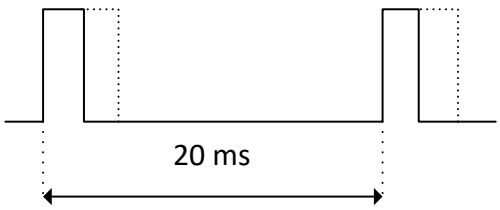


Abbildung 19 - Standard PWM Motorensignal

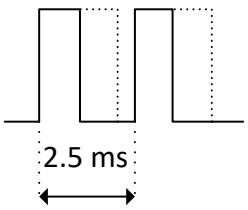


Abbildung 20 - schnelles PWM Motorensignal

4.10 Steuerung des Multicopters

4.10.1 Internetsteuerung

Die Erstellung eines Web-GUIs zur Bedienung eines Particle Photons ist Im Buch «Make: Getting Started with the Photon» detailliert beschrieben (Simon Monk, 2015). Im Web-GUI, Abbildung 21, wurden Druckknöpfe für die Schubregelung zur Höhensteuerung und für die horizontale Navigation erstellt. Diese werden mit der Maus gedrückt. Der aktuelle Schubwert wird in einem Tacho angezeigt. Aufgrund fehlender HTML und JavaScript Kenntnisse, wurden für das Web-GUI Beispiele aus den Kapiteln «Project 7. Control Relays from a Web Page» und «Project 9. Measuring Light over the Internet» übernommen und angepasst. In Abbildung 22 ist ein Auszug aus dem HTML Code zur Anordnung der Druckknöpfe enthalten. Abbildung 23 beschreibt Funktionen zum Aufruf der Photon Particle Cloud.

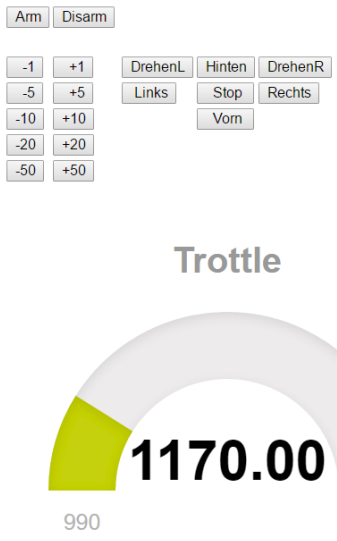


Abbildung 21 - Internet-Steuerung

```
<table>
// Zeilen weggelassen
```

```
    <tr>
            <td><input type="button" onClick="throttle(-5)" value="  -5"/></td>
            <td><input type="button" onClick="throttle(+5)" value="  +5"/></td>
            <td></td>
            <td><input type="button" onClick="move('left')" value=" Links"/></td>
            <td><input type="button" onClick="move('stop')" value="  Stop "/></td>
            <td><input type="button" onClick="move('right')" value="Rechts"/></td>
        </tr>
 </table>
```

Abbildung 22 - HTML Implementation zur Erstellung der Druckknöpfe

```
// Photon kuepfer-1
var accessToken = "97c6bc……..1c36b";
var deviceID = "4300…….3633"

var throttleURL = "https://api.spark.io/v1/devices/" + deviceID + "/throttle";
var moveURL = "https://api.spark.io/v1/devices/" + deviceID + "/move";

function throttle(value)
{
   $.post(throttleURL, {params: value, access_token: accessToken });
}

function move(value)
{
   $.post(moveURL, {params: value, access_token: accessToken });
}
```

Abbildung 23 - JavaScript Funktionen zur Photons Steuerung

5 Inbetriebnahme

5.1 Erste Flugversuche

Für die ersten Flugversuche wurde der Multicopter leicht hängend mit vier Schnüren in Kreuzform an mehreren Stühlen befestigt. Es sollte getestet werden, ob der Multicopter bei Schub in der horizontalen Position ausharren würde oder bei einem Verlassen dieser Position eine richtige Korrektur ausführt, um wieder die gewünschte Position zu erreichen. Dies stellte sich als schlechte Idee heraus, denn bei zunehmender Drehzahl der Motoren begannen sich die Schnüre ungleich zu spannen und der Multicopter wurde regelrecht von den Schnüren in alle Richtungen geschleudert. Um dem entgegenzuwirken, wurde die Versuchsanordnung so umgebaut, dass der Multicopter nur noch an zwei Schnüren gespannt in der Längsrichtung befestigt wurde. Durch die gespannten Schnüre tanzte der Multicopter zwar nicht mehr hin und her, doch er begann nach kurzer Zeit in der Querachse zu schaukeln. Dieses Phänomen konnte auch nicht mit langem Ermitteln von PID Konstanten eliminiert werden.

In einem weiteren Versuch, als der Multicopter mit etwa Viertelschubstellung von Hand auf alle vier Seiten bewegte wurde, stellte ich fest, dass die Lagekorrekturen erst durchgeführt wurden, nachdem sich der Multicopter schon eine gewisse Zeit in einer Schräglage befunden hatte. Somit erklärte sich auch das Schaukeln des Multicopters an den Schnüren, denn bei jedem Schaukeln traf die Korrektur zu spät ein. Durch die ausbleibende Korrektur konnte der Multicopter immer in eine Richtung pendeln, ohne dass sofort gegengesteuert wurde. Da die Schnüre über dem Schwerpunkt des Multicopters befestigt wurden, pendelte dieser wieder in Richtung Horizontalstelle. Die Korrektur setzte auf Grund ihrer Verspätung erst hier ein und beförderte den Multicopter mit verstärkter Wirkung in die entgegengesetzte Richtung.

Die Verzögerung bei den Korrekturen konnte zwar nicht mit einer Analyse der Konsole ausgemacht werden, doch stellte sich heraus, dass sich die Werte des Gierens mit der Zeit immer stärker von ihrem Anfangswert unterschieden, ohne dass überhaupt eine Korrektur eintrat. Ebenfalls begannen die Gierwerte auszuschlagen, wenn der Multicopter in der Längs- und Querachse bewegt wurde. Dies lag daran, dass der Kompass sehr ungenaue Werte lieferte. Im nächsten Kapitel wird eine Alternative für die Pitch, Roll und Yaw Berechnung beschrieben.

5.2 Verwendung Verschiedener Bibliotheken und Sensoren

Nach Internetrecherche stellte sich heraus, dass die Bibliothek von SparkFun für die Berechnung der aktuellen Lage nur den Beschleunigungssensor und den Kompass zu Rate zog und deshalb die Gierkorrektur falsch berechnete. Aus diesem Grund wurde in den folgenden Flugversuchen die Arduino Bibliothek MadgwickAHRS, welche die Abweichung ausschliesslich aus einer Verknüpfung von Beschleunigungs- und Kreiselsensor berechnet, verwendet (Libraries, 2016). Obwohl beim Vergleich der Konsole und den tatsächlich gemessenen Winkelwerten eine Übereinstimmung herrschte, begann der Multicopter dennoch zu schaukeln. Auch wenn das Schaukeln nicht so stark wie mit der SparkFun Bibliothek war, reichte die Arduino Bibliothek MadgwickAHRS jedoch nicht aus, um einen stabilen Schwebeflug zu gewährleisten.

Nach erneuter Internetrecherche entschied ich mich für einen anderen Sensor mit einer anderen Bibliothek zu verwenden. Der SparkFun IMU Breakout mit einer Motion Prozess Unit (MPU)-9250 ist der Nachfolger des im Modellflugsport oft verwendeten MPU-6500. Die gleichnamige Bibliothek MPU-9250 (Electronics, 2016) arbeitet zum einen mit Verknüpfungen der einzelnen Sensoren, zum anderen verwendet sie einen Tiefpassfilter, um Störungen zu eliminieren. Mit der neuen Sensoreinheit und der neuen Bibliothek konnte ein Schaukeln erfolgreich unterbunden werden.

6 Multicopter V2

6.1 Gründe für Multicopter V2

Die beiden grössten Probleme bestanden darin, dass der erste Multicopter zum einen zu unruhig fliegt und zum anderen, dass die Signale der Internetsteuerung mit einer zu grossen zeitlichen Verzögerung den Multicopter erreicht. Deshalb wurde ein zweiter Multicopter gebaut. Dieser besitzt die gleichen Proportionen wie der erste Multicopter, aber er ist etwas grösser. In Abbildung 27 sind beide Multicopter abgebildet, um den Grössenunterschied zu zeigen. Wie bei der ersten Version werden das Particle Photon als Mikroprozessor, das Lantian als Stromverteilerboard und Level-shifter eingesetzt. Als Sensor wird der in Kapitel «5.2 Verwendung Verschiedener Bibliotheken und Sensoren» beschriebene MPU-9250 verwendet. Der MPU wurde, wie im Schema auf Abbildung 24 ersichtlich, in der Mitte der Lötplatine befestigt. Somit wirken beim Drehen um alle drei Achsen weniger Zentripetalkräfte auf den Sensor.

Der RC Empfänger operiert mit einer Eingangsspannung von 5V und gibt ebenfalls Signale in dieser Spannung aus. Das Particle Photon arbeitet jedoch mit einer Spannung von 3.3 Volt. Wie bei den Reglern und dem Particle Photon muss zwischen dem Empfänger und dem Particle Photon ein Level-Shifter Modul eingebaut werden. Dieses wird jedoch mit 3.3V betrieben. Da ein Level-Shifter Modul nur vier Kanäle hat, mussten zwei Stück verbaut werden für den fünf Kanal RC Empfänger.s

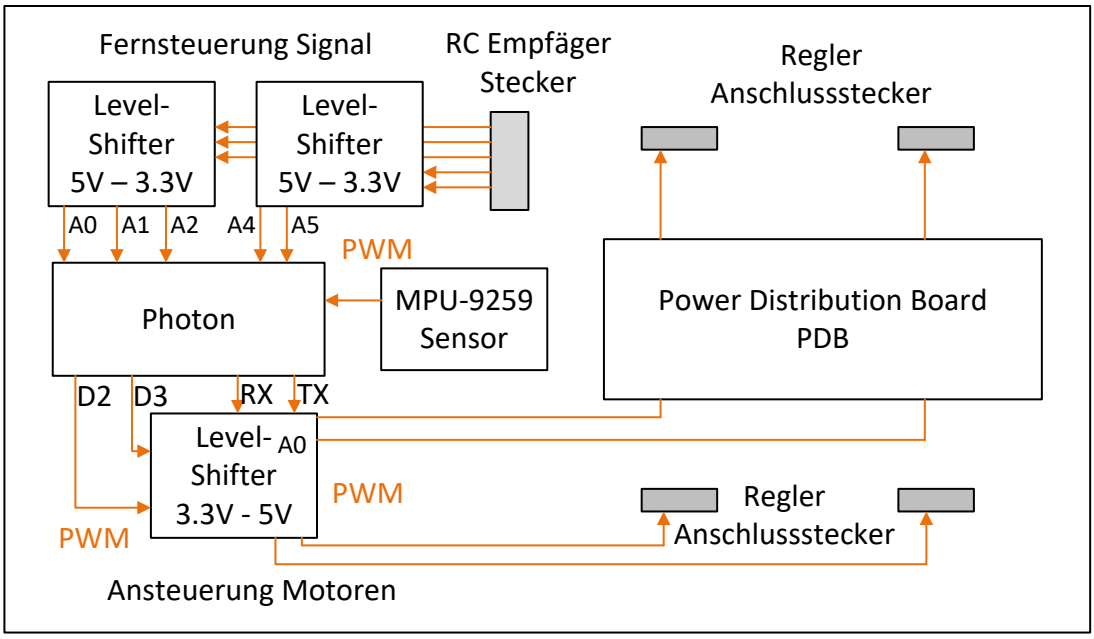


Abbildung 24 - Layout der Komponenten des Multicopters V2

6.2 RC Steuerung

Nach einigen Testflügen mit der im Kapitel «4.10.1 Internetsteuerung» beschriebenen Steuerung mit Hilfe des Web GUI, entstand das Bedürfnis den Multicopter zusätzlich mit einer Fernsteuerung bedienen zu können, weil die Signalübermittlung des Web-GUIs eine zu grosse Verzögerung aufweist. Eine ausführliche Beschreibung, wie das mit Arduino programmiert werde kann, erhält der Artikel «How To Read Multiple RC Channels» (Can_I_Trade?, 2016). Ein RC Empfänger verwendet ebenfalls PWM Signale als Output. Im Setup wird die Funktion `rcReceiverSetup()` aufgerufen. Diese Funktion ruft in der Nebendatei receiver.cpp fünfmal die Funktion `attachInterrupt()` für die Steuerachsen Pitch, Roll, Yaw, Gas und Arm (Aktivierung) auf. Die Interrupt Service Routine (ISR) wird zu jeder Zeit parallel

zum normalen Programm (Loop) aufgerufen. Aus diesem Grund wird in der Particle Photon Doku beschrieben, dass mehrere Aspekte beachtet werden müssen:

- Variablen Volatile definieren (Particle, Interrupt, 2016)
- Verwendung Atomic_Block (Particle, ATOMIC_BLOCK, 2016) und lokale Kopie von den Variablen der Interrupt Methode (Particle, Interrupt, 2016)
- Kurze Interruptmethode

Das Einlesen der Fernsteuerungssignale ist für alle Achsen, auch der Funktion Arm, gleich. In Abbildung 25 ist die Implementation, um ein Fernsteuerungssignal zu lesen, am Beispiel der Achse Roll ersichtlich. Abbildung 26 beschreibt einen PWM Puls eines Fernsteuerungsempfängers. Der Kanal Roll des Fernsteuerungsempfängers ist mit dem Particle Photon Pin A0 verbunden. Mit der Interrupt Service Routine `interruptPinA0()` wird die Pulsbreite des Signals an Pin A0 gemessen. Dafür wird bei einem aufsteigendem Signal mit der Funktion `micros()` die Startzeit gemessen und beim abfallenden Signal die Endzeit. Die Differenz von Startzeit und Endzeit ergibt die Pulsbreite. Da die Abweichung von der Pulsmitte interessiert, wird 1500 von der Differenz abgezogen.

```
// Kanal CH1 Pin A0, roll
// Abweichung vom Mittelwert 1500
void interruptPinA0()
{
  if (digitalRead(A0) == HIGH)
  {
    // aufsteigendes Signal       (1.)
    startRoll = micros();
  }
  else
  {
    // abfallendes Signal         (2.)
    rcRoll = micros() - startRoll - 1500;
  }
}
```

Abbildung 25 - Interrupt Service Routine

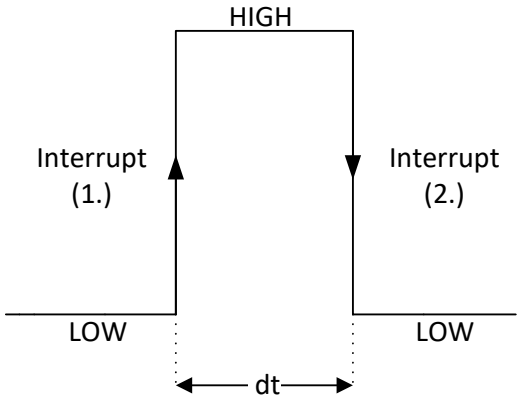


Abbildung 26 - RC PWM Signal

6.3 Grössere Komponenten – Weniger Vibrationen

Weil der neu gebaute Multicopter V2 grösser als der vorherige ist, konnten auch stärkere Motoren und grössere Propeller verwendet werden. Diese rotieren im Flug langsamer und generieren deswegen weniger Vibrationen. Das bedeutet, dass die empfindlichen Sensoren weniger durch Vibration bedingt falsche Werte liefern als beim ersten Multicopter und somit ein ruhigeres Flugverhalten erreicht werden kann. In der Abbildung 27 sind beide Multicopter nebeneinander platziert, um den Grössenunterschied zu zeigen wobei sich rechts der zuerst gebaute Multicopter und links der Multicopter V2 befindet.

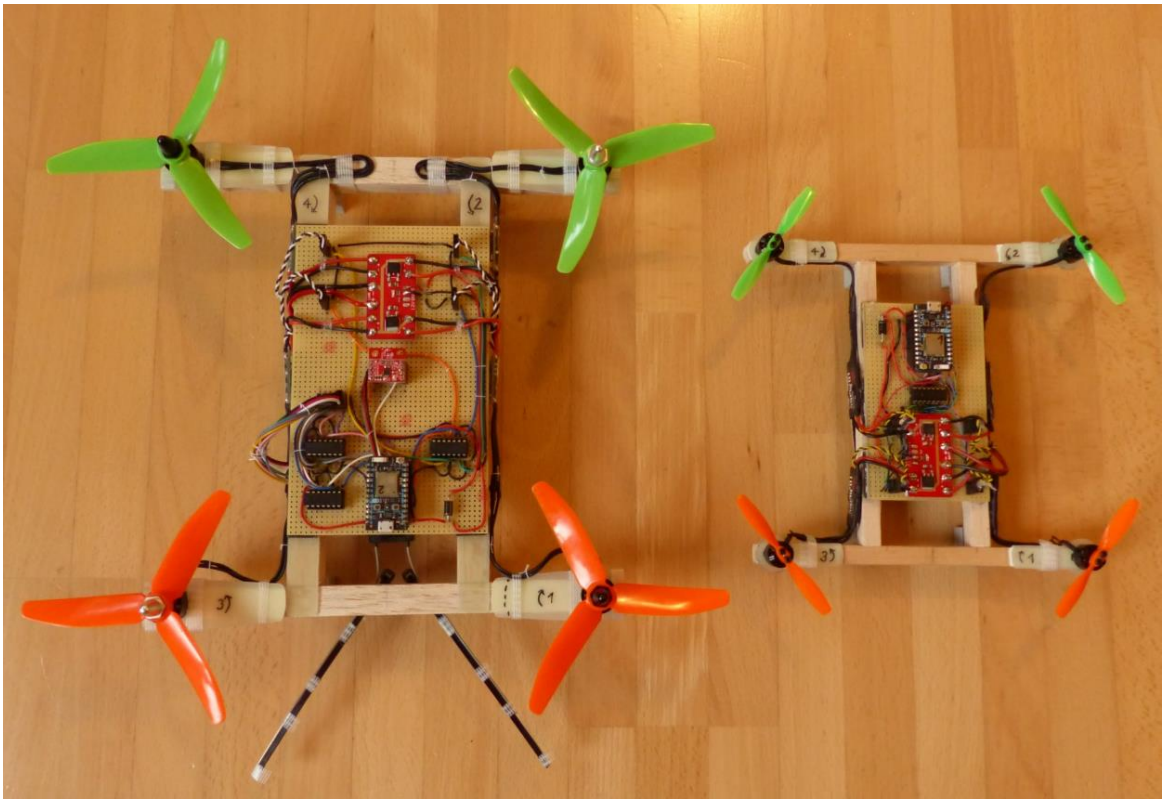


Abbildung 27 - Grössenvergleich der beiden Multicopter

7 Schlussfolgerung

7.1 Erkenntnisse

Der Bau der beiden Multicopter verlief aufgrund eigener Erfahrung problemlos, einzig die vielen Kabel stifteten manchmal Verwirrung.

Das Erlernen der Programmiersprache nahm einige Zeit und Übung in Anspruch. Dank konkreter Programmierung des Multicopters konnte das Erlernte gut angewendet und vertieft werden.

Das Ziel, einen Multicopter mit eigenem Programm via Internetsteuerung mit Hilfe eines Web-GUIs zu steuern, konnte erreicht werden. Die Steuerung über das Internet funktionierte zwar einwandfrei, allerdings war der Multicopter auch nach vollendeter PID Justierung nicht in der Lage, seine Position auf Dauer zu halten. Ein Abdriften in verschiedene Richtung bewirkte das Ausgleichen per Mausklicks zum einen unmöglich, weil das Signal über das WLAN mit einer zeitlichen Verzögerung von etwa einer Sekunde beim Multicopter eintraf und so meistens ein Absturz nicht mehr vermieden werde konnte. Zum andern kann pro Mausklick nur eine einzige Achse angesteuert werden, aber in den meisten Fällen ist es nötig, mehrere Achsen gleichzeitig anzusteuern.

Um zu zeigen, dass der Multicopter dennoch in der Lage ist zu fliegen, wurde, wie in Kapitel «6.2 RC Steuerung» beschrieben, eine RC Anlage verbaut. Überraschenderweise erwies sich dies als ein simples Unterfangen und konnte somit schnell ausgeführt werden. Mit der RC Anlage ist es ohne Probleme möglich, den Multicopter sicher in der Luft zu bewahren.

Grosse Problem stellten die verschiedenen Bibliotheken zu den einzelnen Sensoren dar. Aufgrund von fehlendem Fachwissen konnte nie ausgeschlossen werden, dass die Misserfolge nicht auch auf eine inkorrekt funktionierende Bibliothek zurückzuführen war. Aus diesem Grund wurden, wie in Kapitel «5.2 Verwendung verschiedener Bibliotheken und Sensoren» beschrieben, mehrere Bibliotheken und Sensoren getestet.

7.2 Zahlreiche Optimierungen

Einige Verbesserungen wurden schon im Laufe des Projekts durchgeführt. So wurden laufend Bibliotheken und Sensoren ausgetauscht, um die bestmögliche Kombination zu ermitteln. Ebenfalls wurde auch die Konstruktion des Multicopters umgebaut, um die Chancen auf einen erfolgreichen Flug zu optimieren. Dazu wurden die in Kapitel «3.1 Bau des Rahmens» beschriebenen Glasfaserplatten zur Motorenbefestigung von der Unterseite entfernt und auf der Oberseite befestigt, denn die Motoren waren so angebracht, dass die Propeller nur mit einem Abstand von etwa einem Millimeter über die Arme hinweg rotierten. Ausserdem wurde unterhalb der Längsverstrebung des Rahmens ein Holzbrett befestigt, sodass die Batterie am tiefstmöglichen Punkt montiert werden kann. Mit beiden Umbauten wird gewährleistet, dass sich der Schwerpunkt des Multicopters deutlich unterhalb der Schublinie der Propeller befindet. Bei einer Abweichung der Schwebeposition hilft nun neben den elektronischen Steuerbefehlen auch die Gravitationskraft, um den Multicopter wieder in die waagrechte Position zurück zu befördern. Die Trägheit des Multicopters wird so ebenfalls durch den tieferliegenden Schwerpunkt vergrössert. Nach dem Umbau wurde erheblich weniger Vibration am Multicopter wahrgenommen. Um weitere Vibrationen zu minimieren, wurde zwischen den Befestigungspunkten von Lötplatine und Holzrahmen dämpfendes M3 Schaummaterial angebracht. Beide Verbesserungen führten dazu, dass die Sensoren weniger Vibrationen ausgesetzt sind und somit weniger gestört werden. Eine letzte Verbesserung ist mit dem Verbau einer RC Anlage erfolgt. Das Funksignal einer RC Anlage hat eine Verzögerung von weniger als 100 Millisekunden. Die Steuerung via Internet kann durchaus eine Verzögerung von einer oder zwei Sekunden haben oder gar nicht ankommen.

8 Ausblick Multicopter V3

Für einen zukünftigen Multicopter V3 könnten viele weitere Verbesserungen realisiert werden. Einige Ansätze werden im folgenden Kapitel «8.1 Mehr Sensoren» näher beschrieben.

8.1 Mehr Sensoren

Um das Abdriften des Multicopters zu verhindern, könnten zusätzliche Sensoren eingebaut werden. Eine Möglichkeit, um das Wegdriften von einer bestimmten Stelle zu unterbinden, wäre die Verwendung eines Optischen Flusssensors, der mit Hilfe einer nach unten gerichteten Kamera die Bewegung am Boden wahrnimmt und mit Hilfe eines geeigneten Programmcodes den Multicopter am Wegdriften hindert.

Auch könnten ein oder mehrere Sonarsensoren eingebaut werden. Ähnlich wie bei einer Fledermaus wird ein Ultraschallsignal ausgesendet. Dieses prallt an einem Objekt ab und wird wieder vom Sensor empfangen. Mit der Zeitdifferenz von Aussenden und Empfangen kann der Abstand zu einem Objekt ausgerechnet werden. Werden an alle vier Seiten solche Sensoren angebracht, könnte ebenfalls mit einem geeigneten Programm ein Zusammenstoss mit Objekten verhindert werden.

Um bei Steuerbefehlen in horizontaler Richtung eine konstante Flughöhe sicherzustellen, wäre das Verbauen eines Barometersensors vorteilhaft. Aufgrund von Druckdifferenzen in unterschiedlichen Höhen ist es mit ihm möglich zu überprüfen, ob ein Flugobjekt seine konstante Höhe beibehält.

Eine weitere Ausbaumöglichkeit, um den Multicopter am Wegdriften zu hindern, wäre die Installation eines Global Positioning System (GPS) Sensors. Mit Hilfe von Satellitendaten könnte eine Position festgelegt werden, die der Multicopter mit geeignetem Programm beibehalten würde. Allerdings taugt diese Lösung nur in Bereichen, wo ein GPS Satellitensignal empfangen werden kann.

8.2 Smartphone Steuerung

Eine Idee, die schon während des Projekts entstand, ist eine Steuerung mit Hilfe eines Smartphones. Auf der Anzeige könnten zwei Symbole, die den Steuerknüppel einer RC Anlage entsprächen, programmiert werden. Auf diese Art wäre es wiederum möglich, mehrere Achsen des Multicopters zur gleichen Zeit anzusteuern und die Internetsteuerung beizubehalten. Somit könnte auf eine zusätzliche RC Anlage verzichtet werden.

Heutzutage werden in Smartphones ebenfalls Beschleunigungs- und Kreiselsensoren eingebaut. Mit Hilfe dieser Sensoren könnte ein Multicopter anhand der Neigung des Smartphones gesteuert werden. Wie auch bei der oben erwähnten Displaysteuerung könnte die Steuerung optimiert und die Internetsteuerung beibehaltet werden.

9 Anhang

9.1 Literaturverzeichnis

Adafruit Industries. (10. 09 2016). *74AHCT125*. Von 74AHCT125 - Quad Level-Shifter (3V to 5V) [74AHCT125] Adafruit Industries, Unique; fun DIY electronics and kits: https://www.adafruit.com/product/1787 abgerufen

Arduino. (20. 09 2016). *Arduino Servo Libary* . Von Arduino Servo Libary : https://www.arduino.cc/en/Reference/Servo abgerufen

Arduino. (23. 10 2016). *Global Variable*. Von How to make global variables accessible to libraries/classes?: http://forum.arduino.cc/index.php?topic=165658.msg1235920#msg1235920 abgerufen

Arduino. (07. 10 2016). *Libraries*. Von Moving Functions to External Libraries: http://forum.arduino.cc/index.php?topic=199423.0 abgerufen

Arduino. (16. 09 2016). *Playground*. Von Playground - HomePage: http://playground.arduino.cc/Deutsch/HomePage abgerufen

Arduino. (18. 09 2016). *PWM*. Von Puls Weiten Modulation : https://www.arduino.cc/en/Tutorial/PWM abgerufen

Banggood.com. (02. 08 2016). *PDB* . Von Lantian Mini PDB Stromverteilungsplatine mit 5V / 12V Linear Regulator für FPV Multicopter: http://www.banggood.com/de/LANTIAN-Mini-PDB-Power-Distribution-Board-With-5V12V-Linear-Regulator-For-FPV-Multicopter-p-1002059.html abgerufen

Baumann, R. (10. 08 2016). *Mini-Helikopter*. Von Studenten bauen Mini-Helikopter für Lawinenopfer - News Wissen: Technik - tagesanzeiger.ch: http://www.tagesanzeiger.ch/wissen/technik/Studenten-bauen-MiniHelikopter-fuer-Lawinenopfer/story/22710938 abgerufen

benripley. (15. 08 2016). *Arduino-Quadcopter*. Von Arduino-Quadcopter/Quadcopter at master · benripley/Arduino-Quadcopter · GitHub: https://github.com/benripley/Arduino-Quadcopter/tree/master/Quadcopter abgerufen

benripley. (13. 08 2016). *Source Code*. Von Quadcopter Source Code From Scratch: http://www.benripley.com/development/quadcopter-source-code-from-scratch/ abgerufen

BlHeli. (08. 10 2016). *BLHeliSuite*. Von BLHeliSuite | Two ESC Worlds – One Application: https://blhelisuite.wordpress.com/ abgerufen

Busch, P. (2012). *Elementare Regelungstechnik - Allgemeingültige Darstellung ohne höhere Mathematik* . Würzburg: Vogel Fachbuch.

Can_I_Trade? (03. 12 2016). *Multiple RC Channels*. Von RCArduino: How To Read Multiple RC Channels: http://rcarduino.blogspot.ch/2012/04/how-to-read-multiple-rc-channels-draft.html abgerufen

DigiKey. (04. 08 2016). *Schottky Diode*. Von SB560-T Diodes Incorporated | Discrete Semiconductor Products | DigiKey: http://www.digikey.com/product-detail/en/diodes-incorporated/SB560-T/SB560-TDICT-ND/3053319 abgerufen

Diverse. (12. 11 2016). *Cleanflight*. Von GitHub - cleanflight/cleanflight: Clean-code version of the baseflight flight controller firmware: https://github.com/cleanflight/cleanflight abgerufen

Diverse. (08. 08 2016). *Naze32*. Von cleanflight/Board - Naze32.md at master · cleanflight/cleanflight · GitHub: https://github.com/cleanflight/cleanflight/blob/master/docs/Board%20-%20Naze32.md abgerufen

Drone Matters. (12. 12 2016). *DYS Prop 3020*. Von DYS Prop 3020 - 3" Propellers - Propellers - Power Sys: https://www.dronematters.com/power-system/propellers/3-propellers/dys-prop-3020.html abgerufen

DYS, B. (02. 08 2016). *Brushless Motor*. Von Multi-Rotor Brushless Motor BE 1104: http://www.dys.hk/ProductShow.asp?ID=165 abgerufen

DYS, X. (02. 08 2016). *Speed controller*. Von Speed controller XM10A: http://www.dys.hk/ProductShow.asp?ID=170 abgerufen

Electronics, S. (15. 11 2016). *MPU-9250*. Von 9 Degrees of Freedom - MPU-9250 Breakout: https://github.com/sparkfun/MPU-9250_Breakout abgerufen

Flitetest. (18. 08 2016). *H-Quad*. Von Knuckle H-Quad | Flite Test: http://www.flitetest.com/articles/knuckle-h-quad abgerufen

Gaicher, H. (2012). *Programmieren in C - Programmieren lernen von Anfang an.* Hamburg: tredition GmbH, Hamburg.

Lauszus, K. S. (25. 08 2016). *LaunchPad*. Von LaunchPad Flight Controller: http://blog.tkjelectronics.dk/2015/01/launchpad-flight-controller/ abgerufen

Lauszus, K. S. (25. 08 2016). *Lauszus*. Von GitHub - Lauszus/LaunchPadFlightController: TM4C123G based Flight Controller: https://github.com/Lauszus/LaunchPadFlightController abgerufen

Libraries, A. (27. 09 2016). *MadgwickAHRS*. Von GitHub - arduino-libraries/MadgwickAHRS: Arduino implementation of the MadgwickAHRS algorithm: https://github.com/arduino-libraries/MadgwickAHRS abgerufen

Particle. (03. 10 2016). *ATOMIC_BLOCK*. Von Particle Reference Documentation | Firmware: https://docs.particle.io/reference/firmware/photon/#atomic_block- abgerufen

Particle. (03. 09 2016). *Interrupt*. Von Particle Reference Documentation | Firmware: https://docs.particle.io/reference/firmware/photon/#interrupts abgerufen

Particle. (08. 09 2016). *Introduction*. Von Particle Guides | Introduction: https://docs.particle.io/guide/getting-started/intro/photon/ abgerufen

Particle. (08. 09 2016). *Particle Cloud*. Von Particle Cloud - scalable, secure infrastructure for your IoT product: https://www.particle.io/products/platform/particle-cloud abgerufen

Particle. (03. 09 2016). *Particle Dev*. Von Particle Guides | Particle Dev: https://docs.particle.io/guide/tools-and-features/dev/ abgerufen

Particle. (05. 09 2016). *Photon*. Von Particle Photon Series wifi development kits and connectivity modules: https://www.particle.io/products/hardware/photon-wifi-dev-kit abgerufen

Simon Monk, Z. S. (2015). *Make: Getting Started with the Photon.* San Francisco : MakerMedia.

SparkFun Electronics. (28. 09 2016). *IMU Shield*. Von Photon IMU Shield Hookup Guide: https://learn.sparkfun.com/tutorials/photon-imu-shield-hookup-guide abgerufen

Valvano, J. W. (2014). *Real Time Operating Systems for ARM® Cortex™-M Microcontrollers - Embedded Systems.* Texas: self-published.

Wikipedia. (11. 09 2016). *Wikipedia*. Von Quadrocopter: https://de.wikipedia.org/wiki/Quadrocopter abgerufen

9.2 Bauteilverzeichnis Multikopter V1

1. Photon, A tiny, reprogrammable Wi-Fi development kit for prototyping and scaling your Internet of Things product, Preis $ 19.00
   https://store.particle.io/
2. SparkFun Photon IMU Shield, Preis $24.95
   https://www.sparkfun.com/products/13629
3. MPU-9250 Nine-Axis (Gyro + Accelerometer + Compass) MEMS MotionTracking™ Device
   Preis Fr. 15.90
   http://www.play-zone.ch/de/sparkfun-imu-breakout-mpu-9250.html
   https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/
4. 74AHCT125 - Quad Level-Shifter (3V to 5V) - 74AHCT125, Preis $1.50
   https://www.adafruit.com/products/1787
5. Brushless FPV Racing Motor DYS BE1104, 4 Stück, Preis pro Stück CHF 13.50
   http://www.dys.hk/ProductShow.asp?ID=165
6. Electronic speed controller (ESC) DYS XM Seris for BLHeli firmware XM10A, 4 Stück, Preis pro Stück CHF 13.50
   http://www.dys.hk/ProductShow.asp?ID=170
7. LANTIAN Mini PDB Power Distribution Board With 5V/12V Linear Regulator For FPV Multicopter, Preis CHF 2.81
   http://www.banggood.com/LANTIAN-Mini-PDB-Power-Distribution-Board-With-5V12V-Linear-Regulator-For-FPV-Multicopter-p-1002059.html?rmmds=search
8. Schottky Diode 60V 5A, Pusterla Elektronik Zürich, Preis CHF. 0.70
9. Prototype Lötboard, Hartpapier, Pusterla Elektronik Zürich, Preis CHF 4.80
10. Sockel 14 Pin, Pusterla Elektronik Zürich, CHF 0.85
11. Drei Zellen Lithium Polymer Batterie, 450mAh 65 – 130C Entladungsrate, 40g, Preis CHF 11.50
    https://fpvracing.ch/de/batterien/556-turnigy-nano-tech-450mah-3s-65130c-lipo-batterie-jst.html
12. DYS Prop 3020, 4 Stück, Preis pro Propellerset CHF 1.19
    https://www.dronematters.com/power-system/propellers/3-propellers/dys-prop-3020.html

9.3 Source Code

In den folgenden Kapiteln ist der von mir geschriebene Source Code enthalten. Die verwendeten Bibliotheken sind nicht aufgeführt.

9.3.1 multicopter_code.ino

```
//////////////////////
// multicopter_code.ino - Hauptimplementation
// Emmanuel Küpfer
//////////////////////

// System Headerfiles
#include "MPU9250.h"
#include "quaternionFilters.h"
#include "math.h"
#include "application.h"

// Eigene Headerfiles
#include "pid.h"
#include "sensor.h"
#include "startup_calibration.h"
#include "error.h"
#include "throttle.h"
#include "rc_receiver.h"

#define PRINT_COUNT 20

void printPlus(float value);

float ax, ay, az;
float pitch, roll, yaw;
float pidPitch, pidRoll, pidYaw;
float rcPitchCopy, rcRollCopy, rcYawCopy, rcThrottleCopy, rcArmCopy, rcArmLast;
float rcPitchStartwert, rcRollStartwert, rcYawStartwert;
float errorPitch, errorRoll, erroryaw;
float throttle;
// Particle variable muss Type double sein
double throttleVal = 0;
float speedMotor1, speedMotor2, speedMotor3, speedMotor4;
long now;
float dt;
long lastUpdate;

int printCount = 0;


void setup()
{
  //Particle.function verbindet Photon mit Particle Cloud d.h. mit dem Web GUI
  Particle.function("arm", controlMulticopter);
  Particle.function("throttle", speedMulticopter);
  Particle.function("move", moveMulticopter);
  Particle.variable("trottle_val", throttleVal);

  Serial.begin(115200);

  sensorInit();

  // benötigt nach der Initialisierung damit ein gültiger Wert gelesen werden kann.
  delay(3000);

  startwert = kalibrierung();
```

```
  lastUpdate = micros();

  Serial.print("Startwert Pitch ");   Serial.print(startwert.pitch, 2);
  Serial.print(" Roll ");             Serial.print(startwert.roll, 2);
  Serial.print(" Yaw ");              Serial.println(startwert.yaw, 2);

  rcReceiverSetup();
  initMotors();
}

//Beginn der Programmschlaufe
void loop()
{

  // If intPin goes high, all data registers have new data
  // On interrupt, check if data ready interrupt
  if (sensor.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
  {
    // Lesen aktuelle Sensorwerte
    sensorUpdate();

    pitch = sensor.pitch;
    roll = sensor.roll;
    yaw = sensor.yaw;

    // SYNTAX
    ATOMIC_BLOCK()
    {
    // code here is executed atomically, without task switching or interrupts
      rcThrottleCopy = rcThrottle;

      // RC Ausschläge sind zu stark, durch 10 teilen
      rcPitchCopy = rcPitch / 10;
      rcRollCopy = rcRoll / 10;
      rcYawCopy = rcYaw / 10;
      rcArmCopy = rcArm;
    }

    // Sicherstellen dass arm oder dis-arm nur auferufen wird bei einem Wechsel
    // vom Kippschalter der Fernsteuerung
    if (rcArmCopy > 1500 && rcArmLast < 1500)
    {
      controlMulticopter("on");
      // Abweichnung der Fernsteuerung Kanäle von der Mitte bei Arm
      rcPitchStartwert = rcPitchCopy;
      rcRollStartwert = rcRollCopy;
      rcYawStartwert = rcYawCopy;
    }
    if (rcArmCopy < 1500 && rcArmLast > 1500)
    {
      controlMulticopter("off");
    }
    rcArmLast = rcArmCopy;

    // Rc Anlage liefert keine genauen Mittelwerte bei Mittelstellung der
    // Steuergimbals, deshalb werden die Variabeln rcxxxStartwert der RC
    // Anlage von den Variabel rcxxxCopy subtrahiert.
    rcPitchCopy = rcPitchCopy - rcPitchStartwert;
    rcRollCopy = rcRollCopy - rcRollStartwert;
    rcYawCopy = rcYawCopy - rcYawStartwert;

    // Aufrufen der Funktion zur Fehlerberechnung
    // pitch = ist_wert, startwert.pitch = soll_wert
    errorPitch = berechne_E_t_(pitch, startwert.pitch);
```

```
errorRoll = berechne_E_t_(roll, startwert.roll);
// errorYaw = berechne_E_t_(yaw, startwert.yaw);
// für Yaw wird nur die Drehgeschrwindigkeit vom Sensor berücksichtigt
errorYaw = sensor.gz / 10;

// Berechnung Zeitdifferenz: dt
now = micros();
dt = (now - lastUpdate) / 1000.0;
lastUpdate = now;

// Berechnung PID Werte
// Zum error werden die Werte der Steuerung vom GUI und der Fernsteuerung gezählt
pidPitch = berechne_PID(errorPitch + guiPitch + rcPitchCopy, dt, PID_PITCH);
pidRoll = berechne_PID(errorRoll + guiRoll + rcRollCopy, dt, PID_ROLL);
pidYaw = berechne_PID(errorYaw + guiYaw + rcYawCopy, dt, PID_YAW);

// Geschwindigkeitswertzuweisung, 990 ist kein Gas
throttle = 990 + guiThrottle + rcThrottleCopy;

// Particle variable vom Type double für Tacho Anzeige
throttleVal = throttle;

// Berechnung Motordrehzahl
speedMotor1 = throttle - pidPitch + pidRoll - pidYaw;
speedMotor2 = throttle + pidPitch + pidRoll + pidYaw;
speedMotor3 = throttle - pidPitch - pidRoll + pidYaw;
speedMotor4 = throttle + pidPitch - pidRoll - pidYaw;

// Aufruf der Funktion um auf Motoren zu schreiben
updateMotoresAll(speedMotor1, speedMotor2, speedMotor3, speedMotor4);

// Ausgabe der Steuerwerte, etwa alle 100ms (5 ms looptime * 20)
if (printCount >= PRINT_COUNT)
{
  Serial.print(" dt ");                                  Serial.print(dt, 2);
  Serial.print(" P ");          printPlus(pitch);        Serial.print(pitch, 2);
  Serial.print(" R ");          printPlus(roll);         Serial.print(roll, 2);
  Serial.print(" Y ");          printPlus(yaw);          Serial.print(yaw, 2);

  Serial.print("   ERROR P "); printPlus(errorPitch); Serial.print(errorPitch, 2);
  Serial.print(" R ");         printPlus(errorRoll);  Serial.print(errorRoll, 2);
  Serial.print(" Y ");         printPlus(errorYaw);   Serial.print(errorYaw, 2);

  Serial.print(" T ");                                   Serial.print(throttle, 2);

  Serial.print("   PID P ");    printPlus(pidPitch);    Serial.print(pidPitch, 2);
  Serial.print(" R ");          printPlus(pidRoll);     Serial.print(pidRoll, 2);
  Serial.print(" yaw ");        printPlus(pidYaw);      Serial.print(pidYaw, 2);

  Serial.print("   M1 ");                          Serial.print(speedMotor1, 0);
  Serial.print(" M2 ");                            Serial.print(speedMotor2, 0);
  Serial.print(" M3 ");                            Serial.print(speedMotor3, 0);
  Serial.print(" M4 ");                            Serial.println(speedMotor4, 0);

  Serial.print("   RC P ");                        Serial.print(rcPitchCopy, 0);
  Serial.print(" R ");                             Serial.print(rcRollCopy, 0);
  Serial.print(" Y ");                             Serial.print(rcYawCopy, 0);
  Serial.print(" T ");                             Serial.print(rcThrottleCopy, 0);
  Serial.print(" A ");                             Serial.println(rcArmCopy, 0);

  printCount = 0;
}
else
{
```

```
      printCount++;
    }
  }
}

// Damit Anzeige von negativen und positiven Werten gleich lang ist
void printPlus(float value)
{
  if (value >=0)
  {
    Serial.print ("+");
  }
}
```

9.3.2 startup_calibration.h

```
//////////////////////
// startup_calibration.h - Bestimmung der Startwerte für Pitch, Roll und Yaw
// startup Calibration
//////////////////////

// verhindern, dass Datei startup_calibration.h mehr als einmal eingebunden wird.
#ifndef STARTUP_CALIBRATION_H
#define STARTUP_CALIBRATION_H

// C Struktur "struct" mit dem Namen Position
struct Position
{
  float pitch;
  float roll;
  float yaw;
};

// Deklaration der Funktionen der Datei startup_calibration.cpp
struct Position kalibrierung();

#endif
```

9.3.3 startup_calibration.cpp

```
//////////////////////
// startup_calibration.cpp - Bestimmung der Startwerte für Pitch, Roll und Yaw
// Emmanuel Küpfer
//////////////////////

// Einbindung der benötigten Headerfiles
#include "startup_calibration.h"
#include "MPU9250.h"
#include "sensor.h"

// Berechnung eines Mittelwerts
struct Position kalibrierung()
{
  float summe1 = 0, summe2 = 0, summe3 = 0;
  float mittelwert_pitch, mittelwert_roll, mittelwert_yaw;

  float pitch, roll, yaw;

  struct Position mittelwert;
  int i = 0;

  while (i < 10)
```

```
  {
    // Wenn der Sensor bereit ist wird er abgefragt.
    if (sensor.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
    {
      sensorUpdate();
      pitch = sensor.pitch;
      roll = sensor.roll;
      yaw = sensor.yaw;

      summe1 = summe1 + pitch;
      summe2 = summe2 + roll;
      summe3 = summe3 + yaw;
      i++;
    }
  }

  mittelwert.pitch = summe1 / 10;
  mittelwert.roll = summe2 / 10;
  mittelwert.yaw = summe3 / 10;

  //Rückgabe der Strucktur
  return mittelwert;
}
```

9.3.4 error.h

```
//////////////////////
// error.h - Berechnung der Abweichung
// Emmanuel Küpfer
//////////////////////

// verhindern, dass Datei error.h mehr als einmal eingebunden wird.
#ifndef E_t_H
#define E_t_H

// Deklaration Fehlerberechnung
float berechne_E_t_(float ist_wert, float soll_wert);

#endif
```

9.3.5 error.cpp

```
//////////////////////
// error.cpp - Berechnung der Abweichung
// Fehler berechnen//
//////////////////////

// Fehlerberechung
float berechne_E_t_(float ist_wert, float soll_wert)
{
  // Fehler E(t) zum Zeitpunkt t
  float E_t_;

  // Fehlerberechung
  E_t_ = soll_wert - ist_wert;

  return E_t_;
}
```

9.3.6 pid.h

```
////////////////////////////////
// pid.h - Berechnung der PID Werte für alle Achsen
// Emmanuel Küpfer
////////////////////////////////

//verhindern, dass Datei pid.h mehr als einmal eingebunden wird.
#ifndef PID_H
#define PID_H

// Werte für berechnete PID Achsen
#define PID_PITCH 0
#define PID_ROLL  1
#define PID_YAW   2

// Initialisiere PID
void initPID ();

// Deklaration PID Berechnung
float berechne_PID(float E_t_, float dt, int achse);

#endif
```

9.3.7 pid.cpp

```
////////////////////////////////
// pid.cpp - Berechnung der PID Werte für alle Achsen
// Emmanuel Küpfer
////////////////////////////////

// Einbindung des benötigten Headerfile
#include "application.h"

// Konstante
const float Kp = 2.5;
const float Ki = 0.05;
const float Kd = 30.0;

// Definition der Werte für alternativ Differenzialanteil Berechnung
#define ERR_1 0
#define ERR_2 1
#define ERR_3 2

// Definition Variabeln
// Speicherung pro Achse E(t-1) bis E(t-3) für Berechnung des Differenzialanteil
static int E_t_1[3][3]   = { { 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 } };

// Speicherung pro Achse I(t-1)
static float I_t_1[3]    = { 0, 0, 0 };

// Integral wird bei jedem aktivieren auf 0 gesetzt.
void initPID ()
{
  I_t_1[0] = 0;
  I_t_1[1] = 0;
  I_t_1[2] = 0;
}

// Berechne PID E_t_ Fehler E(t)
// dt = Zeitdifferenz seit letztem Aufruf
```

```
// achse ist PID_ROLL, PID_PITCH oder PID_YAW
float berechne_PID(float E_t_, float dt, int achse)
{
  // Definition PID Variabeln
  float P_t_ = 0;
  float I_t_ = 0;
  float D_t_ = 0;
  float U_t_ = 0;

  // Proportionalanteil
  P_t_ = Kp * E_t_;

  // Integralanteil
  // Multiplikaton mit konstantem Faktor dt wegen konstanter Looptime weggelassen
  I_t_ = I_t_1[achse] + Ki * E_t_; // * dt;
  I_t_1[achse] = I_t_;

  // unklar ob Integralanteil begrenzt werden soll
  // begrenze Integralanteil
  if (I_t_1[achse] > 10) {
    I_t_1[achse] = 10;
    I_t_ = 10;
  }
  if (I_t_1[achse] < -10) {
    I_t_1[achse] = -10;
    I_t_ = -10;
  }

  // Differenzialanteil
  // Division durch konstanten Faktor dt wegen konstanter Looptime weggelassen
  // Eifache berechnung des Differenzialanteils mittles Subtraktion E(t) - E(t-1).
  // Aufgrund von allfälligen Sprungwerte wurde eine weniger fehleranfällige Methode
  // verwendet.
  // D_t_ = Kd * (E_t_ - E_t_1[achse]); // / dt;
  // E_t_1[achse] = E_t_;

  // Allternativberechnung vom Differenzialanteil
  // Division durch konstanten Faktor dt wegen konstanter Looptime weggelassen.
  // Mit der Durchschnittsberechnung fallen Abweichungen nicht stark ins Gewicht.
  // Differenzial Berechung gemäss Buch 'Real-time operating system for ARM Corex-M
  // Microcontrollers,
  // J.W Valvano, Seite 381'.
  D_t_ = Kd * (E_t_ + 3 * E_t_1[achse][ERR_1] - 3 * E_t_1[achse][ERR_2] -
E_t_1[achse][ERR_3]) / 6;
  E_t_1[achse][ERR_3] = E_t_1[achse][ERR_2];
  E_t_1[achse][ERR_2] = E_t_1[achse][ERR_1];
  E_t_1[achse][ERR_1] = E_t_;

  // PID-Summe
  U_t_ = P_t_ + I_t_ + D_t_;

  // Rückgabewert
  return U_t_;
}
```

9.3.8 throttle.h

```
////////////////////////////////
// throttle.h - Einlesen der Internetsteuerung sowie Arm und Disarm Funktion
// Emmanuel Küpfer
////////////////////////////////

//verhindern, dass Datei throttle.h mehr als einmal eingebunden wird.
```

```
#ifndef THROTTLE_H
#define THROTTLE_H

// Einbindung des benötigten Headerfile
#include "startup_calibration.h"

//extern da in thottle.cpp definiert und da von multicopter.ino gelesen wird
extern struct Position startwert;
extern boolean armed;

//"extern": Variabel ist im throttle.cpp bereits definiert.
extern float guiRoll;
extern float guiPitch;
extern float guiYaw;
extern float guiThrottle;

// Deklaration der Funktion der Datei throttle.cpp
int controlMulticopter(String command);

int speedMulticopter(String speedCommand);

int moveMulticopter(String moveCommand);

void setupMotorsAll();

void updateMotoresAll(float speedMotor1, float speedMotor2, float speedMotor3, float
speedMotor4);

void initMotors();

#endif
```

9.3.9 throttle.cpp

```
////////////////////////////////
// throttle.cpp - Einlesen der Internetsteuerung sowie Arm und Disarm Funktion
// Emmanuel Küpfer
////////////////////////////////

// Einbindung der benötigten Headerfiles
#include "application.h"
#include "startup_calibration.h"
#include "throttle.h"
#include "pid.h"

//Position = Typ, startwert = Variabel, struct muss übernommen werden
//Globale Variabel muss static sein, falls nicht im ino File?
struct Position startwert;
boolean armed = false;

float guiRoll = 0;
float guiPitch = 0;
float guiYaw = 0;
float guiThrottle = 0;


//"int speedMulticopter(String speedComamnd)" wird aufgerufen durch Particle Cloud
// Nach Buch Make: Getting Started with the Photon, Simon Monk
// https://github.com/simonmonk/photon_book/blob/master/firmware/exam-
ples/p_06_LED_Function.cpp

// Arm (Aktivierung) Funktion
int controlMulticopter(String command)
```

```
{
  Serial.print("controlMulticopter ");
  Serial.println(command);

    if (command=="on")
    {
        armed = true;
        digitalWrite(D7, HIGH);
        startwert = kalibrierung();
        initMotors();
        initPID ();
        return 1;
    }
    else if (command=="off")
    {
        armed = false;
        digitalWrite(D7, LOW);
        guiThrottle = 0;
        return 0;
    }
    else
    {
        return -1;
    }
}


// wird von Particle Cloud aufgerufen
// Nach Buch Make: Getting Started with the Photon, Simon Monk
// https://github.com/simonmonk/photon_book/blob/master/firmware/exam-
ples/p_06_LED_Function.cpp
int speedMulticopter(String speedCommand)
{
    Serial.print("speedMulticopter ");
    Serial.println(speedCommand);

    //"atoi" = "asccii", wandelt Zahlen in Stings um
    guiThrottle = guiThrottle + atoi(speedCommand);
    return 1;
}

// wird von Particle Cloud aufgerufen
// Nach Buch Make: Getting Started with the Photon, Simon Monk
// https://github.com/simonmonk/photon_book/blob/master/firmware/exam-
ples/p_06_LED_Function.cpp
int moveMulticopter(String moveCommand)
{
    Serial.print("moveMulticopter ");
    Serial.println(moveCommand);

    if (moveCommand.equals("forward")) {
      guiPitch = guiPitch - 5;
    } else if (moveCommand.equals("back")) {
      guiPitch = guiPitch + 5;
    } else if (moveCommand.equals("left")) {
      guiRoll = guiRoll - 5;
    } else if (moveCommand.equals("right")) {
      guiRoll = guiRoll + 5;
    } else if (moveCommand.equals("rotateLeft")) {
      guiYaw = guiYaw - 5;
    } else if (moveCommand.equals("rotateRight")) {
      guiYaw = guiYaw + 5;
    } else if (moveCommand.equals("stop")) {
      guiRoll = 0;
```

```
      guiPitch = 0;
      guiYaw = 0;
    }
    return 1;
}

//Servo = Typ, motor = Variabel
void updateMotoresAll(float speedMotor1, float speedMotor2, float speedMotor3, float
speedMotor4)
{
  if (armed) //armed
  {
      // analogWrite(Pin, Bereich 0 bis 255, Frequenz)
      // speedMotor1-4 werden durch 9.803... geteilt damit
      // sie im bereich von 0 bis 255 sind, wobei 1000 muss
      // auf 102 umgerechnet werden für Standschub.
      // 9.803... = 1000 / 102
      analogWrite(D2, speedMotor1 / 9.80392156863, 400);
      analogWrite(TX, speedMotor2 / 9.80392156863, 400);
      analogWrite(D3, speedMotor3 / 9.80392156863, 400);
      analogWrite(RX, speedMotor4 / 9.80392156863, 400);
  }
  else
  {
    // not armed
    // Motoren werden 1ms Pulsbreite beschrieben,
    // für Stillstand.
    analogWrite(D2, 100, 400);
    analogWrite(TX, 100, 400);
    analogWrite(D3, 100, 400);
    analogWrite(RX, 100, 400);
  }
}

// Initialisierung Motoren PWM Pins.
// Pulsweite 1ms fuer Stillstand.
void initMotors()
{
  //pinMode(Pin,Modus) Definition ob Pin Input oder Output
  pinMode(D2, OUTPUT);
  pinMode(TX, OUTPUT);
  pinMode(D3, OUTPUT);
  pinMode(RX, OUTPUT);

  // analogWrite(Pin, Bereich 0 bis 255, Frequenz)
  // für eine Millisekunde Pulsbreite bei 400Hz mit Peridendauer von2.5 ms
  // ist value 102 = 255 / 2.5 ms.
  analogWrite(D2, 100, 400);
  analogWrite(TX, 100, 400);
  analogWrite(D3, 100, 400);
  analogWrite(RX, 100, 400);
}
```

9.3.10 rc_receiver.h

```
////////////////////////////////
// rc_receiver.h - Einlesen des PWM RC Empfänger
// Emmanuel Küpfer
////////////////////////////////

// verhindern, dass Datei rc_receiver.h mehr als einmal eingebunden wird.
#ifndef rc_receiver_h
#define rc_receiver_h
```

```
// Deklaration der Funktion der Datei rc_receiver.cpp
void rcReceiverSetup();

//"extern": Variabel ist im rc_receiver.cpp bereits definiert.
extern volatile int rcPitch;
extern volatile int rcRoll;
extern volatile int rcYaw;
extern volatile int rcThrottle;
extern volatile int rcArm;

#endif
```

9.3.11 rc_receiver.cpp

```
////////////////////////////////
// rc_receiver.cpp - Einlesen des PWM RC Empfänger
// Emmanuel Kuepfer
////////////////////////////////

// http://rcarduino.blogspot.ch/2012/04/how-to-read-multiple-rc-channels-draft.html
// Schlüsselwort volatile
// http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/009_c_funktio-
nen_010.htm#mj347a8bbe7206e189cfc3764ade226922

// Einbindung des benötigten Headerfile
#include "application.h"

int startPitch;
int startRoll;
int startYaw;
int startThrottle;
int startArm;

// Damit Variablen bei einer interuptmethode (ISR) korrekt übergeben,
// Deklaration mit volatile
volatile int rcPitch = 0;
volatile int rcRoll = 0;
volatile int rcYaw = 0;
volatile int rcThrottle = 0;
volatile int rcArm = 0;

void interruptPinA0();
void interruptPinA1();
void interruptPinA2();
void interruptPinA4();
void interruptPinA6();

//  Initialisierung RC Empfängerpins
void rcReceiverSetup()
{
  attachInterrupt(A0, interruptPinA0, CHANGE);
  attachInterrupt(A1, interruptPinA1, CHANGE);
  attachInterrupt(A2, interruptPinA2, CHANGE);
  attachInterrupt(A4, interruptPinA4, CHANGE);
  attachInterrupt(A6, interruptPinA6, CHANGE);
}


// Kanal CH1 Pin A0, roll
// Abweichung vom Mittelwert 1500,
// weil sich die Fersteurungsgimbal in der
// Mittelposition befinden.
```

```
void interruptPinA0()
{

  if (digitalRead(A0) == HIGH)
  {
    // start Zeit PWM Pulsbreite HIGH
    startRoll = micros();
  }
  else
  {
    // Ende Zeit PWM Pulsbreite HIGH
    rcRoll = micros() - startRoll - 1500;
  }
}


// Kanal CH2 Pin A1, pitch
// Abweichung vom Mittelwert 1500
void interruptPinA1()
 {
  if (digitalRead(A1) == HIGH)
  {
    //Start Zeit PWM Pulsbreite HIGH
    startPitch = micros();
  }
  else
  {
    // Ende Zeit PWM Pulsbreite HIGH
    rcPitch = micros() - startPitch - 1500;
  }
}

// Kanal CH3 Pin A2, throttle
// Abweichung vom Mittelwert 1500
void interruptPinA2()
{
  if (digitalRead(A2) == HIGH)
  {
    //Start Zeit PWM Pulsbreite HIGH
    startThrottle = micros();
  }
  else
  {
    // Ende Zeit PWM Pulsbreite HIGH
    rcThrottle = micros() - startThrottle - 1000;
  }
}

// Kanal CH4 Pin A3, yaw
// Abweichung vom Nullwert 1000
void interruptPinA4()
{
  if (digitalRead(A4) == HIGH)
  {
    //Start Zeit PWM Pulsbreite HIGH
    startYaw = micros();
  }
   else
  {
    // Ende Zeit PWM Pulsbreite HIGH
    rcYaw = micros() - startYaw - 1500;
  }
}
```

```
// Kanal CH5 Pin A4, arm
// Abweichung vom Nullwert 1000
void interruptPinA6()
{
  if (digitalRead(A6) == HIGH)
  {
    //Start Zeit PWM Pulsbreite HIGH
    startArm = micros();
  }
  else
  {
    // Ende Zeit PWM Pulsbreite HIGH
    rcArm = micros() - startArm;
  }
}
```

9.3.12 multicopter-photon-kuepfer-1.html

```
<html>
<head>
    <!--
     * multicopter-photon-kuepfer-1.html
     * Emmanuel Küpfer
     *
     * Nach Beispielen aus dem Buch
     * Getting Started with Photon, https://github.com/simonmonk/photon_book
     * Simon Monk, http://simonmonk.org/
     * - Beispiel Steuerung mit Knöpfen
     *   https://github.com/simonmonk/photon_book/blob/master/html/ch_06_relays.html
     * - Beispiel Messwert Anzeige
     *   https://github.com/simonmonk/photon_book/blob/master/html/p_10_thermome-
ter.html
     -->

    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
    <script src="raphael.2.1.0.min.js"></script>
    <script src="justgage.1.0.1.min.js"></script>

    <script>
        // Photon kuepfer-1
        var accessToken = "97c6bc39……7f0b1c36b";
        var deviceID = "43002f……33633"

        // Photon kuepfer-2
        // var accessToken = "cb8b348000…….bfb57b30e";
        // var deviceID = "55ff740……81667"

        var armURL = "https://api.spark.io/v1/devices/" + deviceID + "/arm";
        var throttleURL = "https://api.spark.io/v1/devices/" + deviceID + "/throttle";
        var trottle_valueURL = "https://api.spark.io/v1/devices/" + deviceID + "/trot-
tle_val";
        var moveURL = "https://api.spark.io/v1/devices/" + deviceID + "/move";


        function arm(value)
        {
            $.post(armURL, {params: value, access_token: accessToken });
        }


        function throttle(value)
        {
```

```
            $.post(throttleURL, {params: value, access_token: accessToken });
        }

        function move(value)
        {
            $.post(moveURL, {params: value, access_token: accessToken });
        }


        function callback(dataJSON, status){
            if (status == "success") {
                data = JSON.parse(dataJSON);
                temp = parseFloat(data.result);
                temp = temp.toFixed(2);
                g.refresh(temp);
                setTimeout(getThrottle, 1000);
            }
            else {
                alert("There was a problem");
            }
        }
        function getThrottle(){
            $.get(trottle_valueURL, {access_token: accessToken}, callback);
        }

    </script>
</head>

<body>

<h1>Multicopter und Internet of Things (IoT)</h1>
<h2>Steuerung (kuepfer-photon-1)</h2>

    <table>
        <tr>
            <td><input type="button" onClick="arm('on')" value="Arm"/></td>
            <td><input type="button" onClick="arm('off')" value="Disarm"/></td>
            <td></td>
            <td></td>
            <td></td>
            <td></td>
        </tr>

        <tr>
            <td><br></td>
            <td></td>
            <td></td>
            <td></td>
            <td></td>
            <td></td>
        </tr>

        <tr>
            <td><input type="button" onClick="throttle(-1)" value="  -1"/></td>
            <td><input type="button" onClick="throttle(+1)" value="  +1"/></td>
            <td></td>
            <td><input type="button" onClick="move('rotateLeft')"
value="DrehenL"/></td>
            <td><input type="button" onClick="move('back')" value="Hinten"/></td>
            <td><input type="button" onClick="move('rotateRight')"
value="DrehenR"/></td>
        </tr>

        <tr>
```

```
            <td><input type="button" onClick="throttle(-5)" value="  -5"/></td>
            <td><input type="button" onClick="throttle(+5)" value="  +5"/></td>
            <td></td>
            <td><input type="button" onClick="move('left')" value=" Links"/></td>
            <td><input type="button" onClick="move('stop')" value="  Stop "/></td>
            <td><input type="button" onClick="move('right')" value="Rechts"/></td>
        </tr>

        <tr>
            <td><input type="button" onClick="throttle(-10)" value="-10"/></td>
            <td><input type="button" onClick="throttle(+10)" value="+10"/></td>
            <td></td>
            <td></td>
            <td><input type="button" onClick="move('front')" value="  Vorn "/></td>
            <td></td>
        </tr>

        <tr>
            <td><input type="button" onClick="throttle(-20)" value="-20"/></td>
            <td><input type="button" onClick="throttle(+20)" value="+20"/></td>
            <td></td>
            <td></td>
            <td></td>
            <td></td>
        </tr>

        <tr>
            <td><input type="button" onClick="throttle(-50)" value="-50"/></td>
            <td><input type="button" onClick="throttle(+50)" value="+50"/></td>
            <td></td>
            <td></td>
            <td></td>
            <td></td>
        </tr>

    </table>

    <br>
    <!-- <h2>Anzeige</h2> -->
    <div id="trottle" style="width:400px; height:320px"></div>
    <script>
        var g = new JustGage({
            id: "trottle",
            value: 990,
            min: 990,
            max: 2000,
            title: "Trottle"
        });
        getThrottle();
    </script>


</body>
</html>
```