SERVICE

Aufgabe und Lösung

mit

Arduino UNO

Steuerungen für einen Propellerantrieb

Ausgeführt in Werkstätte und Labor von NW-SERVICE

Ing. Norbert Willmann

www.nw-service.at

Gesteuerter Propeller-Antrieb mittels Arduino

Angeregt durch eine Abschlussarbeit der Fachschule Elektrotechnik der HTL-Wels wurde dieses Projekt durchgeführt. Es wurden einige Aufgaben bei der Steuerung anders und eher unüblich gelöst, als bei der Abschlussarbeit.

1. Aufgabenstellung:

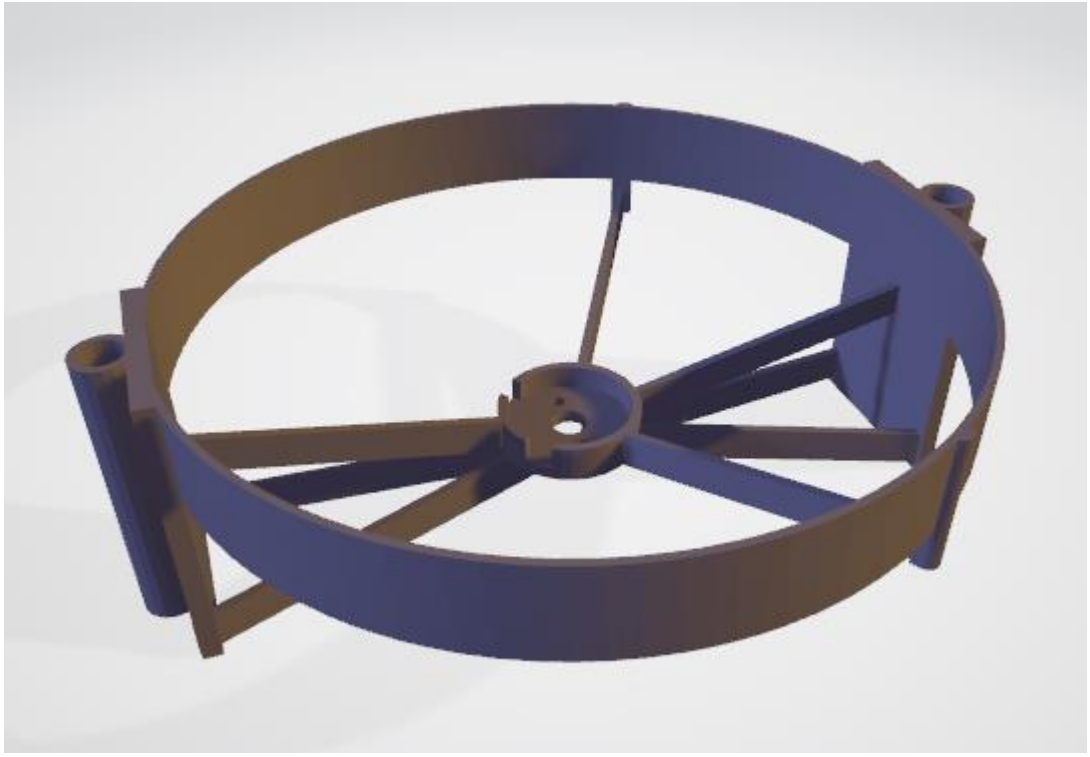
Ein Propeller-Antrieb mit einem Brushlessmotor, mit „Modul" bezeichnet, soll in beliebige Höhen gesteuert werden und danach auch sanft in der Ausgangsstellung landen können. Die Regelung erfolgt mit einem Arduino UNO und dieser bekommt über Lichtsensoren mit einem Laser-Pointer Vorgaben.

Das Programm soll so einfach als möglich gestaltet werden und möglichst ohne externe Bibliotheken und ohne Interrupts auskommen.

2. Ausführung:

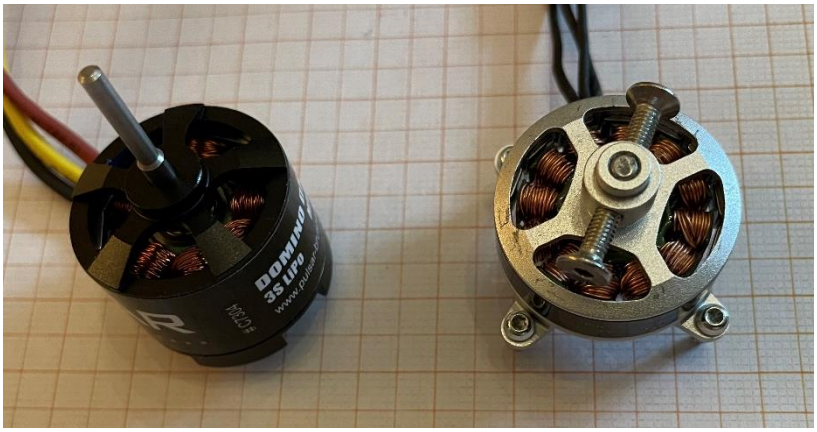
Es wurde ein stabiler Motorträger konstruiert, der auch einen Schutz vor Berührung des rotierenden Propellers bietet. Das Modul gleitet auf zwei ein Meter langen Rohren aus Aluminium mit acht Millimeter Durchmesser in die Höhe.

Die Konstruktion erfolgte mit dem 3D-Programm Tinkercad, ein sehr einfach zu be-dienendes und schnell zu erlernendes online-Zeichenprogramm aus dem Internet.



Mit einem 3D-Drucker der Firma Prusa konnte der Motorträger aus PLA-Kunststoff hergestellt werden.

2.1. Brushless-Motoren

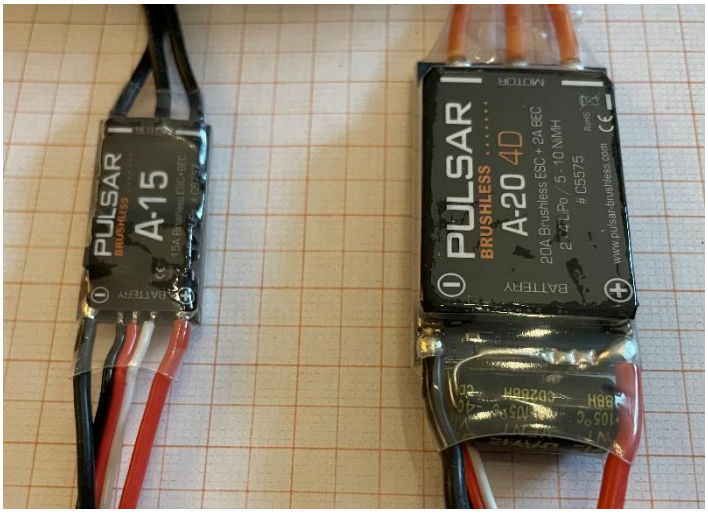


Üblicherweise sind Elektromotoren so konstruiert, dass sich der Rotor innen auf einer Welle befindet und dass außen der Stator entweder aus lamellierten Stahlblechpolen mit Wicklungen besteht oder mit Dauermagneten bestückt ist.

Brushless-Motoren bestehen aus einem Stator, der innen aus einzelnen sternförmig angeordneten, elektrisch erregten Polen besteht (z.B. 12 Pole) und einem rotierenden Rotor außen, der mit einzelnen Dauermagneten (z.B. 13 Magnete) bestückt ist. Da der Radius der Kraftwirkung außen größer ist, wird ein deutlich höheres Drehmoment auch bei kleinen Abmessungen erzielt.

Der Stator hat drei Anschlüsse, ähnlich den Anschlüssen bei Drehstrommotoren und es wird auch wie bei diesen ein Drehfeld erzeugt, dem der Rotor „nachläuft". Erzeugt werden die phasenverschobenen Ströme der drei Anschlüsse von einem ESC.

2.2. ESC, electronic-speed-controller



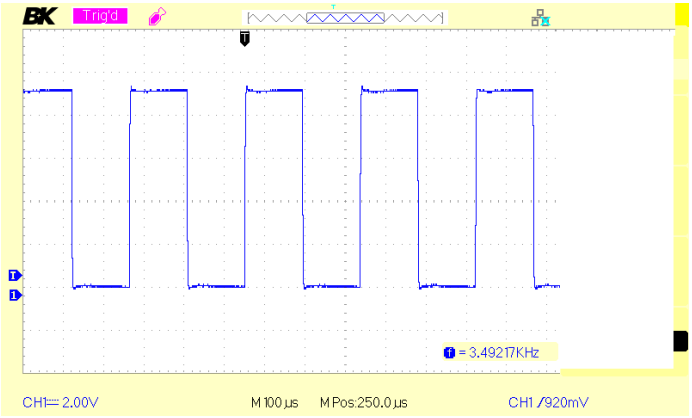
Der dreipolige Ausgang des Controllers ist mit dem Motor verbunden. Werden zwei der drei Anschlüsse zum Motor getauscht, ändert sich die Drehrichtung.

Der Eingang hat fünf Anschlüsse, zwei für die Stromversorgung mit 2, 3 oder 4 LiPo-Akkuzellen (etwa 8V bis 16V) und einem maximalen Strom von 20A.

Eine dreipolige, dünndrahtige Steuerleitung (schwarz, rot und weiß, für Minus 0V, +5V und Impulse), bestimmt mit einem PWM-Signal die Drehzahl des Motors.

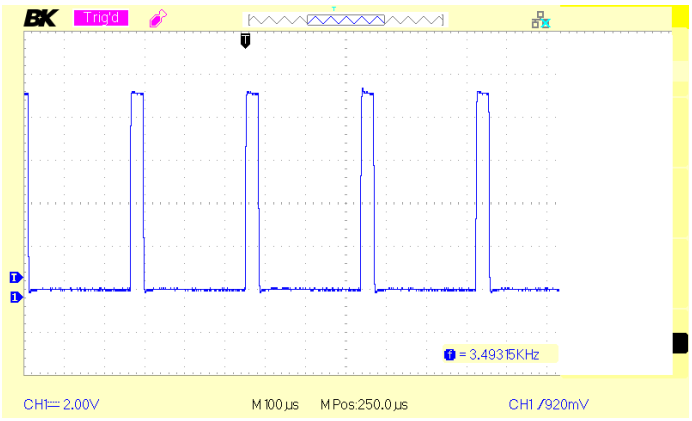
2.3. PWM-Steuerung

Die Puls-Weiten-Modulation PWM (besser Puls-Breiten-Modulation, PBM) beruht auf dem Prinzip, dass bei Rechteckimpulsen die Breite der Impulse variiert wird. Im rechten Bild sind die Impulse gleich breit, wie die Pausen, somit ist der Effektivwert genau die halbe Maximalspannung. Bei einer Impulshöhe von 9V, wie im rechten Bild, beträgt der Effektivwert 4,5V (ist derselbe Wert, wie eine äquivalente Gleich-spannung).



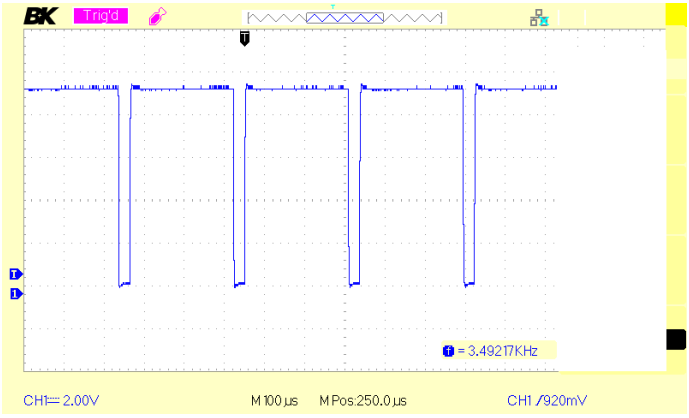
Im rechten Bild wurde die Pulsbreite stark reduziert und die Pause erhöht. Mit einem Messgerät wurde ein Effektivwert von 1,39V bei 9V Maximalwert gemessen.

Rechnerisch ist die Impulsdauer etwa 40µs und die Pause 240µs. Dies ergibt ein Verhältnis von 6,66 und somit einen rechnerischen Effektivwert von 9,5V * 14,3% = 1,36V.



Bei langem Impuls und kurzer Pause, wie im rechten Bild, wurde ein Effektivwert von 8,74V gemessen.

Rechnerisch ist das Verhältnis zum vorigen Beispiel gleich nur invers. Also Impuls 250µs und Pause von 27µs. Das Verhältnis ist 9,25 und somit ist der rechnerische Wert 9,5V * 90,25% = 8,57V.



Duty: $D = t_{imp} / (t_{imp} + t_{pause})$

Effektivwert: $U_{eff} = U_{imp} * D$

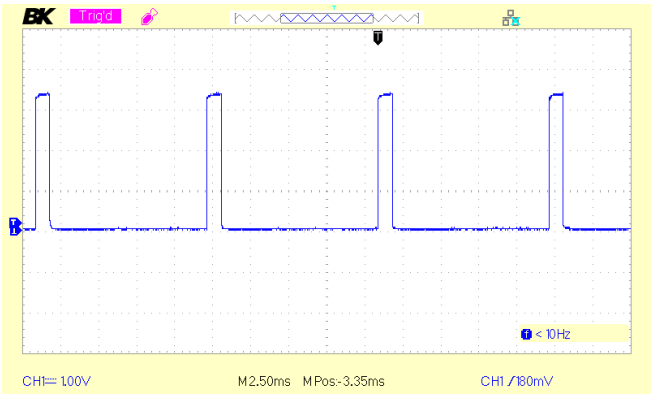
Diese Beispiele werden bei Regelungen von Leuchtmitteln (LED), von Motoren und z.B. bei getakteten Netzgeräten verwendet. Der besondere Vorteil besteht darin, dass Ströme mit elektronischen Elementen (CMOS-Transistoren) sehr verlustarm geschaltet werden können und der breite Regelbereich der PWM ein breites Anwendungsgebiet eröffnet.

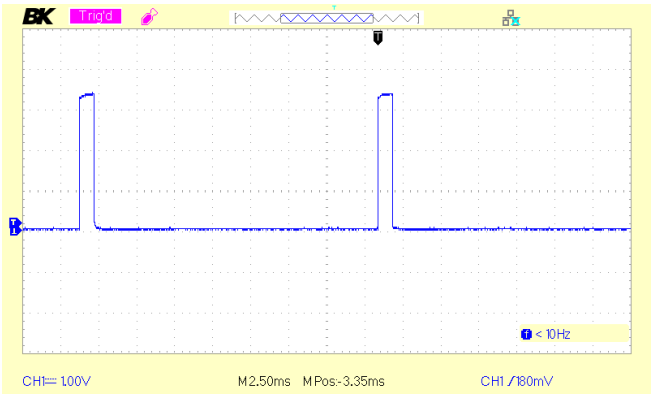
In heutigen Zügen der Bahn sind beim Anfahren höher werdende Töne zu hören, dies sind auch akustische Nebeneffekte der PWM-geregelten Antriebsmotoren.

2.3.1. PWM-Steuerung für den Brushless-Motor

Zur Steuerung für diese Motoren, für Servomotoren und für z.B. Rudermaschinen bei Flugmodellen wird auch die PWM-Steuerung verwendet. Zum Unterschied der Anwendung über den Effektivwert der vorigen Erklärung wird in diesen Fällen nur die Breite des Impulses als Maß für die Steuerung benützt. Die Dauer beträgt 1ms für Null und 2ms für Maximum. Die Pausenzeit ist sehr variabel und unerheblich für die Steuerung.

Im ESC wird die Breite des Impulses in die Frequenz des Drei-Phasenantriebes umgesetzt und außerdem wird beim Einschalten geprüft, ob die Regelung den Wert der Nullstellung hat (1ms), sodass kein Sprung auf eine beliebig hohe Drehzahl erfolgen kann.





Mit einem PWM-Impulsgenerator wurden zwei Verhältnisse dargestellt. In beiden Fällen beträgt die Impulsdauer 1ms, die Pause beträgt links 10ms, rechts 18ms. Dies ergibt links einen Effektivwert von 0,3V und rechts von 0,17V, also fast die Hälfte.

Eine Impulsdauer von 1,4ms bewirkt eine mittlere Drehzahl beim Brushlessmotor. Die Änderung der Pause von 10ms auf 18ms ergibt jedoch ***keine*** Änderung der Drehzahl. Damit ist deutlich erkennbar, dass ***nur die Impulsdauer*** die Regelung bewirkt. In diesem Video ist diese Erkenntnis sicht- und hörbar dargestellt.

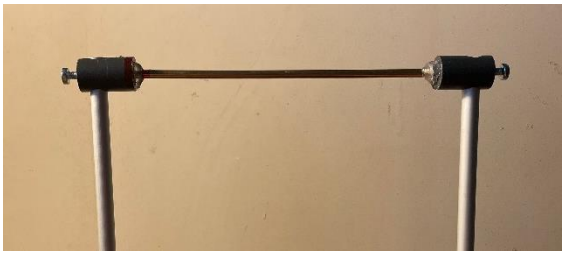
2.4. Energieversorgung

Die ersten Versuche wurden mit einem getaktetem Netzgerät 12V, 10A unternommen, wobei dieses Gerät Lastspitzen nicht abfangen konnte. So wurde ein dreizelliger LiIonen-Akku zu den Anschlussbuchsen parallelgeschaltet.

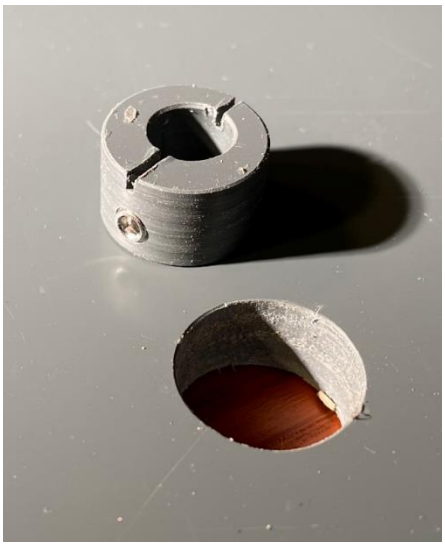
Hierbei gibt es Einiges zu beachten, die möglichen Probleme sind in diesem Beitrag erklärt.

2.5. Mechanischer Aufbau der Leitrohre

Wie aus dem Gesamtbild des Projekts (siehe Seite xx) zu sehen ist, schwebt das Modul auf zwei Rohren mit 8mm Durchmesser auf und ab.

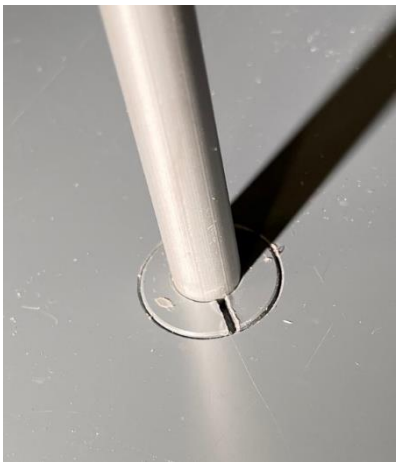


Der Abstand dieser beiden Rohre muss exakt eingehalten werden, damit das Gleiten mit nur geringem Widerstand erfolgen kann. Oben werden die Rohre mit zwei Zylindern und einem Messing-stab mit jeweils einem Rechts- und Linksgewinde an den Enden in Abstand gehalten. Durch verdrehen des Stabes kann der Abstand exakt eingestellt werden. Kontramuttern sichern die Einstellung. Außen an den Zylindern sind Klemmschrauben für die Leitrohre angebracht.



In der Grundplatte, PVC 10mm dick, steckt ein Leitrohr streng in einer angepassten (geriebenen) 8mm Bohrung fest. Die zweite Bohrung benötigt den exakten Abstand, der durch den Motorträger gegeben ist. Um ein Justieren des Abstandes zu ermöglichen, wurde der Effekt des Exzenters angewendet.

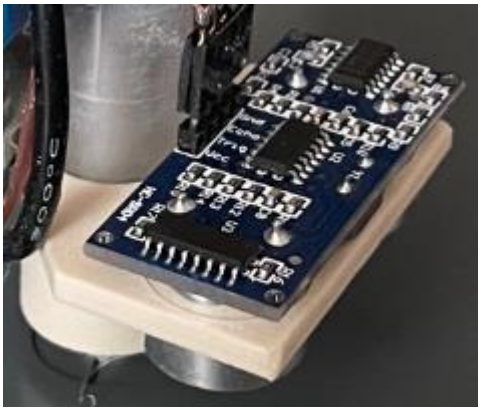
In der 16mm-Bohrung der Grundplatte wird ein Ring mit einer exzentrischen 8mm-Bohrung eingesetzt. Zuvor wird aber das Leitrohr in diesem Ring mittels Wurmschraube fixiert.



Nun kann durch Verdrehen des Ringes der Abstand der Leitrohre exakt eingestellt werden.

Es ist empfehlenswert, dass Elektroniker, auch in alten Maschinenbau-Büchern stöbern, wenn sie nicht bereits Erfundenes nochmals erfinden wollen!

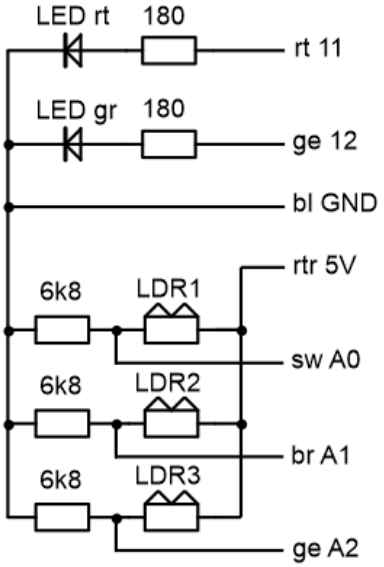
2.6. Halterung für das Ultraschallmodul



Aus PLA wurde ein Halter für das USM HC-SR04 gedruckt, welches an dem UNO gegenüberliegenden Rohrfuß angebracht ist.

Die vier Anschlüsse, Gnd, Echo, Trig und Ucc sind mittels vierpoliger Leitung mit dem UNO, GND, D8, D9, +5V auf einer Aufsteckplatine für den UNO verbunden (siehe nächste Seite, links unten)

2.7. Laser-Empfänger



Zur Fernsteuerung des Moduls werden drei Befehle benötigt:

- Schwebe auf Höhe 1
- Schwebe auf Höhe 2
- Lande (Höhe 3)

Zusätzlich sind zwei LED zur Anzeige des Betriebszustandes erforderlich:

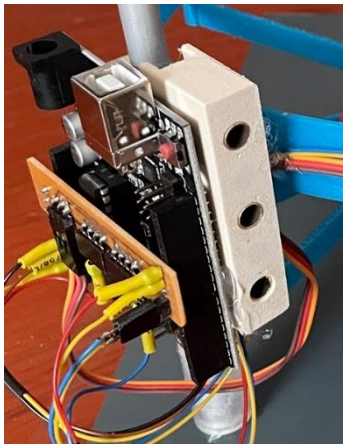
- ROT, gestartet
- GRÜN, Landung eingeleitet

Die Empfänger für diese Befehle sind drei LDR (lichtempfindliche Widerstände) die mit einem Laser-Pointer aus beliebiger Entfernung „getroffen" werden können.

Die LDR sind an 20mm langen Messingröhrchen angebracht. Die drei Röhrchen und die beiden LED sind in einem quaderförmigen Teil untergebracht, der mit PLA gedruckt wurde.

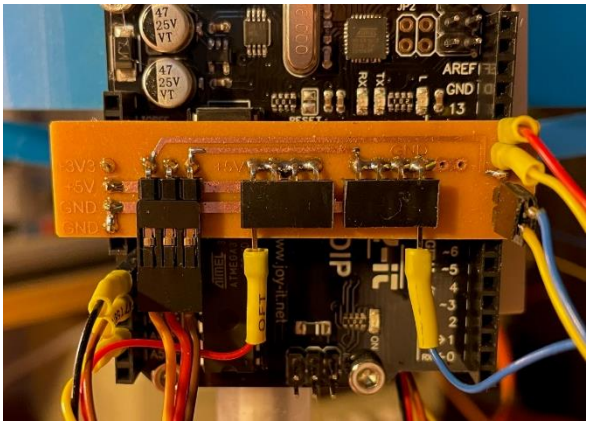
Alle Widerstände der Schaltung sind SMD-Bauteile, daher sehr klein und mit Pinzette zu löten, siehe linkes Bild.

Die beiden LED befinden sich zwischen den drei Röhrchen, die Bohrungen sind mit einer 0,4mm dicken PLA-Schicht abgedeckt.



Der fertig verdrahtete Empfänger ist hinter dem UNO befestigt und mit kurzen Verbindungsdrähten angeschlossen, wie im rechten Bild zu sehen ist.

Um die Buchsen-Anschlussleisten des UNO zu schonen wurde eine kleine Zusatzplatine gefertigt, die dem UNO-Raster angepasst ist und ein leichtes Anschließen von GND, +5V, vom PWM-Anschluss und dem Ultraschallmodul ermöglicht.



Layout für die Stiftplatine (seitenverkehrt) zum schonenden Anschluss an den UNO.

Der Rahmen ist 60x14 mm. Die Steckerleisten sind auf der Lötseite angebracht, zweimal vierpolige Buchsenleiste, einmal dreipolige Steckerleiste.

3. Regelung

Es sollen drei Regelungen mit dem Modul erprobt werden:

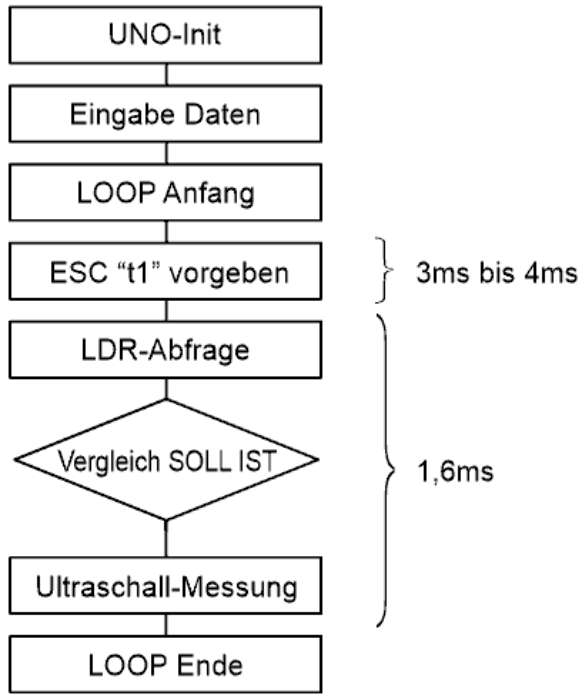
- Zweipunktregelung
- Dreipunktregelung
- Aperiodische Regelung

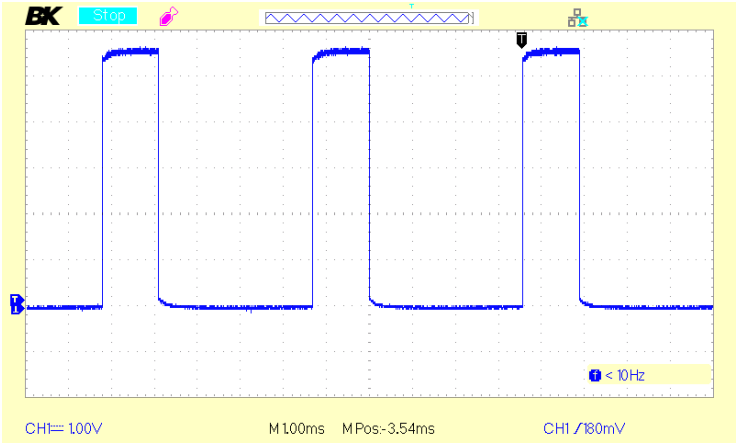
Die Steuerung erfolgt mit einem Arduino UNO mit einer Taktfrequenz von 16MHz.

3.1. Zweipunkt-Regelung

Bei dieser gibt es nur zwei Betriebsarten, Steigen oder Sinken. Mit der Höhenmessung mittels Ultraschallgebers wird der IST-Wert der Höhe bestimmt und dann im Vergleich mit dem SOLL-Wert entschieden, ob die Höhe vermindert oder gesteigert werden muss.

Beim „Landeprozess" wird einfach der Höhen-SOLL-Wert auf 2cm gesetzt und das Modul sinkt bis zu dieser Höhe und schaltet den Motor ab.





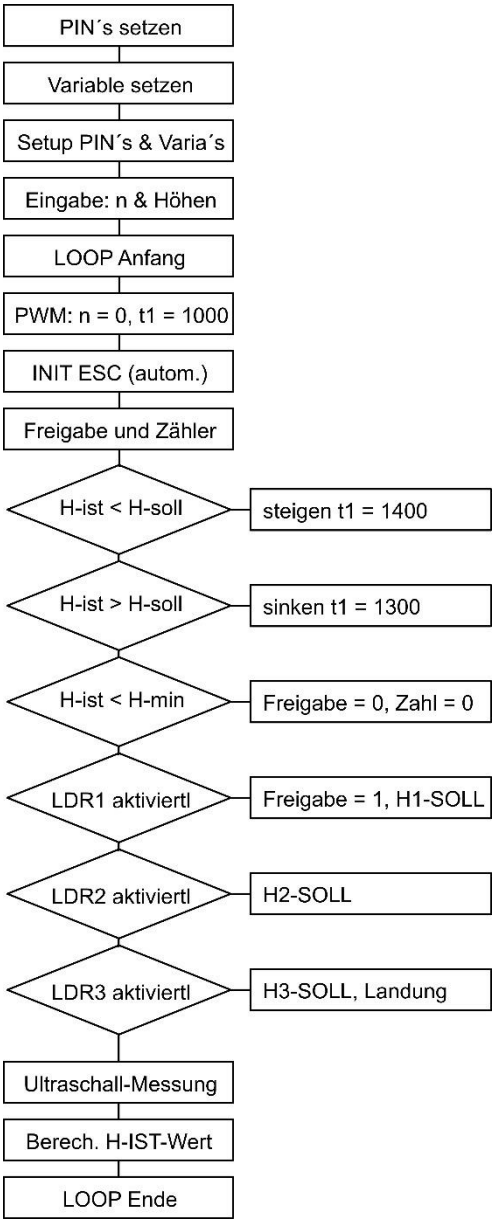
t_{imp} = 1,3ms für Sinken

t_{pause} = 3,6ms, davon sind 2ms-Verzögerung im PWM-Befehl bereits enthalten, also benötigt der UNO nur 1,6ms für sämtliche, restliche Schritte.

Üblicher Weise würden mehrere Aufgaben eines Programmes in Programmteile zerlegt und diese dann vom Gesamtprogramm z.B. mit Interrupt aufgerufen. Das bringt mehr Übersicht und Ordnung in ein Programm. In diesem Beispiel wären dies Teile wie, PWM, Abstandsmessung vom Boden und die Auswahl der entsprechenden Drehzahl.

Trotzdem wurde bei dieser Aufgabe die Methode „alles in einer Wurst" gewählt, da ja genügend Zeit im PWM-Signal übrig ist und statt einer leeren Pause, die Verarbeitung aller notwendigen Vorgänge in diese Pause integriert wurde.

3.1.1. Ausführliches Flussdiagramm der Zweipunkt-Regelung



Im Arduino Initialisierungs-Programm-Teil (wird einmalig vor der Schleife nach dem Programmstart durchlaufen) werden mehrere Aufgaben erfüllt, wie:

- Bestimmung der Anschlüsse (PIN´s)
- Festlegen deren Bestimmung (INPUT, OUTPUT)
- Definition der Variablen
- Setzen der PIN´s
- Setzen der Variablen auf Startbedingung
- Eingabe der Werte der Variablen

Die Programm-Schleife (LOOP) wird permanent, mit wenigen Ausnahmen (aber nicht in diesem Beispiel), durchlaufen. Im konkreten Beispiel wird:

- PWM-Signal mit Nulldrehzahl ausgegeben
- Bedingte Freigabe und bedingter Zahlenstart (derzeit nicht im Programm vorhanden)
- Nach Prüfung der aktuellen Höhe der Vergleich mit der jeweilig gegebenen Höhenvorgabe und danach die Anweisung „STEIGEN oder SINKEN"
- Prüfen welcher LDR aktiviert wurde und die entsprechenden Vorgaben setzen
- Ultraschallmessung
- Berechnen der tatsächlichen Höhe in cm
- Schleife schließen und beim Schleifenanfang wieder beginnen

Besonders ist dabei, dass beim ersten Durchgang weder die aktuelle Höhe bestimmt wird, noch die Befehlsabfrage über die LDR durchgeführt wird, dass also die PWM nur die Null-Drehzahl an den ESC ausgibt.

Dies ist auch erforderlich, da nur mit der Null-Drehzahl (Impulsdauer 1ms) der ESC initialisiert werden kann und dieser meldet die vollzogene Initialisierung hörbar mit einer Tonimpuls-Sequenz. Ertönt diese Sequenz nicht, liegt ein Fehler vor und das Programm muss überprüft und neu gestartet werden.

Im folgenden Programm sind drei Programmzeilen mit „**DREIPUNKT-Regelung**" markiert, aber nicht aktiviert //. Diese sind für die folgende Dreipunktregelung vorgesehen. Der Unterschied zwischen der Dreipunktregelung und der Zweipunktregelung besteht in der Ergänzung um die Schwebedrehzahl. Dies bedeutet, dass das Modul nicht nur „STEIGT" oder „SINKT", sondern bei richtiger Höhe ohne Höhenänderung „SCHWEBT".

3.1.2. Programm für die Initialisierung des UNO, Anschlüsse und Variable

```
/* modul-gesamtprogramm ultraschall plus pwm
datei: 20210327-gesamtprogramm-16-fertig-mit_motor
*/

// defines pins numbers ultraschall
const int trigPin = 9;
const int echoPin = 8;
const int aPin = 3;
const int impPin = 5;

// defines pins led
const int out1Pin = 11;      // LED-Pin 11
const int out2Pin = 12;      // LED-Pin 12
const int ldr1Pin = A0;      // Variable ldr übernimmt den Wert von A0, Start und Höhe 1
const int ldr2Pin = A1;      // Variable ldr übernimmt den Wert von A1, Start und Höhe 2
const int ldr3Pin = A2;      // Variable ldr übernimmt den Wert von A2, Landung

int ldr1 = 0;
int ldr2 = 0;
int ldr3 = 0;

// defines pins pwm
int pwPin = 10;     // PWM-Ausgang DigitalPin 10

// defines variables ultraschall
long duration;
int distance;
int anzeige;         //derzeit nicht verwendet

// definitionen des variablen-typs
int t1 = 0;
int zahl = 0;
int freigabe = 0;    //derzeit nicht verwendet
int go = 0;
int hoehe = 0;
int hoehe1 = 0;
int hoehe2 = 0;
int hoehe3 = 0;
int steig = 0;
int sink = 0;
//int schweb = 0; …. Für DREIPUNKT-Regelung

void setup() {
// setup für ultraschall
pinMode(trigPin, OUTPUT); // Setzen des trigPin auf Ausgang
pinMode(echoPin, INPUT);  // Setzen des echoPin auf Eingang
pinMode(impPin, OUTPUT);  // Setzen des impPin auf Ausgang

// setup für pwm
  pinMode(ldr1Pin, INPUT);    // setzen des LDR1Pin auf Eingang
  pinMode(ldr2Pin, INPUT);    // setzen des LDR2Pin auf Eingang
  pinMode(ldr3Pin, INPUT);    // setzen des LDR3Pin auf Eingang
  pinMode(out1Pin, OUTPUT);   // setzen des out1Pin auf Ausgang
  pinMode(out2Pin, OUTPUT);   // setzen des out2Pin auf Ausgang
  pinMode(pwPin, OUTPUT);     // setzen des pwPin auf Ausgang

// EINGABE! Folgende Werte VOR dem Start ermitteln und eingeben
  steig = 1400;               // Steigdrhezahl - diesen Wert am Modell ermitteln
  sink = 1350;                // Sinkdrehzahl - diesen Wert am Modell ermitteln
  //schweb = 1375;              // Schwebedrehzahl - diesen Wert am Modell ermitteln…DREIPUNKT-Regelung
  hoehe1 = 10;                // erste Höhe in cm angeben
  hoehe2 = 20;                // zweite Höhe in cm angeben
  hoehe3 = 2;                 // dritte Höhe in cm angeben, Landestellung
  t1 = 1000;                  // Für den Start des ESC mit Drehzahl Null
  freigabe = 0;               // derzeit nicht verwendet
//  Serial.begin(9600);         // Start der seriellen communication
}
```

Die Drehzahlwerte für t1 müssen vor dem Start experimentell ermittelt (Drehzahl für Steig- und Sinkflug) und in das Programm eingegeben werden.

3.1.3. Programm für die Programmschleife des UNO der Zweipunktregelung

```
void loop() {

// PWM
  digitalWrite(pwPin,HIGH);  // pwPin HIGH-setzen
  delayMicroseconds(t1);     // HIGH-Zeit von 1 bis 2 ms, t1 wird im SETUP erstmalig auf Wert 1 gesetzt,
Motorstillstand, aber ESC aktiv
  digitalWrite(pwPin,LOW);   // pwPin Null-setzen
  delay(5);                  // LOW-Zeit, beliebig von 2ms bis 30ms, dieser Wert wird um die restliche
Prozesszeit (1,6ms) erhöht.

//Höhenfeststellung und Entscheidung
     // folgendes darf nicht ohne vorher mit t1=1000 gestartet zu haben, ausgeführt werden
     //if ((go == 1) && (distance == hoehe)) {(t1 = schweb);}    //richtige Höhe …DREIPUNKT-Regelung
     if ((go == 1) && (distance < hoehe)) {(t1 = steig);}      //zu tief
     if ((go == 1) && (distance > hoehe)) {(t1 = sink);}       //zu hoch

     if ((go == 2) && (distance > (hoehe  - 1))) {(t1 = sink);}
     if ((go == 2) && (distance < hoehe)) {t1 = 1000; go = 0; digitalWrite(out2Pin, LOW);}
//freigabe und go ausschalten

//LDR
     ldr1 = analogRead(ldr1Pin);        // Auslesen des ldrPin und Zuweisung zu "ldr1" (0 bis 1024)
     if (ldr1 > 400) {go = 1;}                                          // go auf 1 setzen, start
     if ((go == 1) && (ldr1 > 400)) {hoehe = hoehe1; digitalWrite(out1Pin, HIGH); t1 = steig; }
//   Höhe 1, 10cm, LED-rot=1

     ldr2 = analogRead(ldr2Pin);        // Auslesen des ldrPin und Zuweisung zu "ldr2" (0 bis 1024)
     if ((go == 1) && (ldr2 > 400)) {hoehe = hoehe2; digitalWrite(out1Pin, HIGH); } // Höhe 2, 20cm

     ldr3 = analogRead(ldr3Pin);        // Auslesen des ldrPin und Zuweisung zu "ldr3" (0 bis 1024)
     if (ldr3 > 400) {go = 2; digitalWrite(out2Pin, HIGH); }                       // go auf 2 setzen
     if ((go == 2) && (ldr3 > 400)) {hoehe = hoehe3; digitalWrite(out1Pin, LOW); } // Höhe 3, LED-rot=0

// ultraschall:

   digitalWrite(trigPin, LOW);    // LÖSCHE trigPin
   delayMicroseconds(2);          // Verzögerung 2µs
   digitalWrite(trigPin, HIGH);   // Setze trigPin auf HIGH für 10 micro seconds
   delayMicroseconds(10);         // Verzögerung 10µs
   digitalWrite(trigPin, LOW);    // LÖSCHE trigPin

   duration = pulseIn(echoPin, HIGH);   // Lese echoPin, Schallreflexion in µs

   distance= duration*0.034/2;          // Berechnung der Distance


   digitalWrite(impPin, HIGH);
   delayMicroseconds(100);
   digitalWrite(impPin, LOW);

}
```

Dieses Programm funktioniert, wie in diesem Video zu sehen ist. Der UNO ist noch extern vom Modul getrennt auf der Tischplatte liegend und die Aktivierung mit dem Laser-Pointer beim weißen LED-Träger ist schwierig zu sehen, aber doch erkennbar.

Ablauf:

- Mit einem Laser-Pointer wird der erste LDR (rechts) aktiviert, das Modul steigt bis zur ersten Höhe (10cm).
- Nach einigen Sekunden wird der zweite LDR (Mitte) aktiviert, das Modul steigt bis zur zweiten Höhe (20cm).
- Nach weiteren Sekunden wird wieder der erste LDR (rechts) aktiviert, das Modul senkt sich bis auf 10cm ab.
- Nach weiteren Sekunden wird der dritte LDR (links) aktiviert, das Modul senkt sich bis zum Startpunkt und schaltet den Motor aus.

3.2. Dreipunkt-Regelung

Das Programm der Zweipunktregelung wird im Initialisierungs- und im Loop-Programmteil um insgesamt drei Zeilen ergänzt. Die Zweipunktregelung kennt nur zwei Funktionen, nämlich STEIGEN und SINKEN, in der Dreipunktregelung kommt die Funktion SCHWEBEN hinzu.

3.2.2. Programmergänzung in Initialisierung und Loop des UNO

In der Initialisierung sind zwei Zeilen einzufügen:

```
In init()
int t1 = 0;
int zahl = 0;
int freigabe = 0;    //derzeit nicht verwendet
int go = 0;
int hoehe = 0;
int hoehe1 = 0;
int hoehe2 = 0;
int hoehe3 = 0;
int steig = 0;
int sink = 0;
int schweb = 0;
```

```
In void setup()
// EINGABE! Folgende Werte VOR dem Start ermitteln und eingeben
  steig = 1400;               // Steigdrhezahl - diesen Wert am Modell ermitteln
  sink = 1350;                // Sinkdrehzahl - diesen Wert am Modell ermitteln
  schweb = 1375;              // Schwebedrehzahl - diesen Wert am Modell ermitteln…DREIPUNKT-Regelung
  hoehe1 = 10;                // erste Höhe in cm angeben
  hoehe2 = 20;                // zweite Höhe in cm angeben
  hoehe3 = 2;                 // dritte Höhe in cm angeben, Landestellung
  t1 = 1000;                  // Für den Start des ESC mit Drehzahl Null
  freigabe = 0;               // derzeit nicht verwendet
```

In der Programm-Schleife (Loop) ist eine Zeile bei den IF-Bedingungen eingefügt:

```
//Höhenfeststellung und Entscheidung
     // folgendes darf nicht ohne vorher mit t1=1000 gestartet zu haben, ausgeführt werden
     if ((go == 1) && (distance == hoehe)) {(t1 = schweb);}    //richtige Höhe …DREIPUNKT-Regelung
     if ((go == 1) && (distance < hoehe)) {(t1 = steig);}      //zu tief
     if ((go == 1) && (distance > hoehe)) {(t1 = sink);}       //zu hoch

     if ((go == 2) && (distance > (hoehe  - 1))) {(t1 = sink);}
     if ((go == 2) && (distance < hoehe)) {t1 = 1000; go = 0; digitalWrite(out2Pin, LOW);}
```

Diese Art von Regelung lässt aber keinen zusätzlichen größeren Einfluss von außen zu, wie deutliche Änderung des Gewichts, stark veränderte Reibung, etc. Kleine Änderungen werden durch Verlangsamung oder Erhöhung der Bewegungsgeschwindigkeit ausgeglichen.

In diesem Video ist deutlich zu erkennen, dass in der Bewegung kaum ein Unterschied zum Video der Zweipunktregelung ist, siehe vorige Seite.

Obwohl in der Steuerung vorgesehen ist, dass bei Erreichung der SOLL-Höhe der Motor auf SCHWEBE-Drehzahl schaltet (`if ((go == 1) && (distance == hoehe)) {(t1 = schweb);}`), ist zu sehen, dass das Modul durch die schnelle Bewegung über diese Höhe hinaus gleitet und beim Sinken unter die SOLL-Höhe fällt, sodass nahezu die gleiche Pendelbewegung, wie bei der Zweipunktregelung entsteht.

3.3. Aperiodische-Regelung (eigentlich exponentielle Regelung)

Die Pendelbewegung bei der Zwei- und Dreipunktregelung könnte durch definierte, gleichmäßige Reibung gedämpft, aber nicht verhindert werden! Diese Reibung muss aber auch durch mehr Schub des Propellers kompensiert werden.

Die Schwingung entsteht, da das Modul mit konstantem Schub, der deutlich **größer** als der Schwebeschub ist, bis zum Sollwert fliegt und mit seiner kinetischen Energie weiter darüber hinaus gleitet.

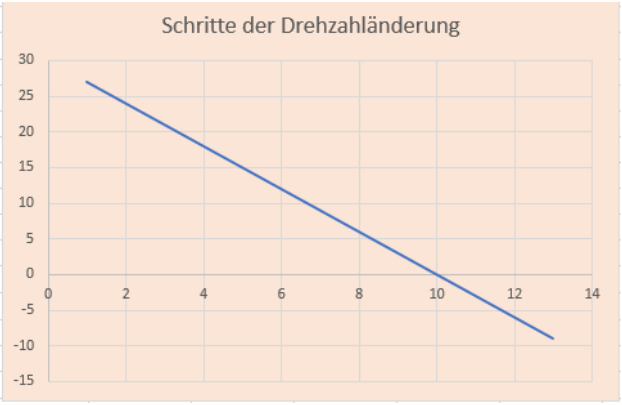
Die Regelung stellt dies fest und reagiert mit einem Schub, der **unter** dem Schwebeschub liegt. Das Modul sinkt, da aber auch dieser Sinkschub konstant ist, wird wieder der Sollwert erreicht und mit der kinetischen Energie weit unterschritten. Dass bei diesem Vorgang kurzzeitig der Sollwert erreicht und das Modul auf Schwebeschub geregelt wird, wirkt sich wegen der Kurzzeitigkeit kaum aus. Daher haben Zwei- und Dreipunktregelungen die gleichen Regel-Schwingungen.

Eine Lösung wäre eine Regelung, die den Schub bei Annäherung an den Sollwert reduziert und so immer „langsamer werdend" das Ziel „erreicht". Langsamer deshalb, weil ein reduzierter Schub die Bewegungsgeschwindigkeit des Moduls auch reduziert.

3.3.1 LÖSUNG:

Es wurde eine neue Variable eingeführt „schritt", die sich aus der Differenz von Soll- und Ist-Wert der Höhe errechnet und den aktuellen Schub an den Schwebeschub mit immer kleiner werdenden Schritten anpasst.

Sollhöhe	Isthöhe	Differenz	Schritt	t1-aktuell	t1-Schwebe	
cm	cm	cm	Wert	µs	µs	
10	1	9	27	1402	1375	Steigen
10	2	8	24	1399	1375	
10	3	7	21	1396	1375	
10	4	6	18	1393	1375	
10	5	5	15	1390	1375	
10	6	4	12	1387	1375	
10	7	3	9	1384	1375	
10	8	2	6	1381	1375	
10	9	1	3	1378	1375	
10	**10**	**0**	0	**1375**	**1375**	Ziel
10	11	-1	-3	1372	1375	Sinken
10	12	-2	-6	1369	1375	
10	13	-3	-9	1366	1375	



In dieser Tabelle ist auf der vertikalen (y) Achse die Schrittgröße, also die Änderung der Schubkraft, um den Schwebeschub zu erreichen, in Abhängigkeit der Höhe dargestellt. Dies ist ein linearer Verlauf, was aber bedeutet, dass der zeitliche Verlauf der Regelung anders aussieht, da wie schon vor der Tabelle erklärt, bei geändertem Schub sich auch die Gleitgeschwindigkeit ändert.

Bei Überschreitung des Schwebeschubs wird dieser mit negativen Schritten verringert!

In diesem Video ist der Regelvorgang zu sehen und auch was geschieht, wenn das Modul durch externen Einfluss in seiner Lage verändert wird. Hier das Video mit höherer Auflösung.

Auch in diesem Beispiel werden zwei Höhen „angeflogen" und dann „gelandet".

Für Mathematik-Fans ist in diesem Link eine Erklärung für aperiodische Schwingung zu finden: https://www.youtube.com/watch?v=6zzsdGYUPaA

Allerdings hätte der Autor dieser Beschreibung daraus keine Schlüsse für die Regelung des Moduls ziehen können! Er hat die Erklärungen nicht verstanden und trotzdem die Regelung zustande gebracht. Praxis geht vor Theorie!

3.3.2. Programmergänzung in der Initialisierung und im Loop-Teil des UNO

Im Initialisierungsteil:

```
int t1 = 0;           // Dauer des Impulses der PWM
int zahl = 0;         // Zahl für automatischen Ablauf
int go = 0;           // Marke für Ruhe, Betrieb oder Landung (0, 1, 2)
int hoehe = 0;        // aktuelle SOLL-Höhe
int hoehe1 = 0;       // 1. Höhenvorgabe
int hoehe2 = 0;       // 2. Höhenvorgabe
int hoehe3 = 0;       // 3. Höhenvorgabe
int schweb = 0;       // t1 für Schwebe-PWM
int schritt = 0;      // Differenz t1 zur Schwebe-PWM
```

Im Loop-Teil:

```
// Höhenfeststellung und Entscheidung

59  schritt = (3*(hoehe - distance));                                  // t1-Schritt für Drehzahländerung
60  if ((go == 1) && (distance == hoehe)) {(t1 = schweb);}             // t1 für richtige Höhe …

61  if ((go == 1) && (distance != hoehe)) {(t1 = (schweb + schritt));}// t1 für ungleiche höhe

62  if ((go == 2) && (distance > (hoehe - 1))) {(t1 = schweb-40);}    //alternativ doch sinkdrehzahl
festlegen und nutzen
63  if ((go == 2) && (distance < hoehe)) {t1 = 1000; go = 0; digitalWrite(out2Pin, LOW);}    //t1 für
nulldrehzahl, go=0

//LDR

72  ldr3 = analogRead(ldr3Pin);        // Auslesen des ldrPin und Zuweisung zu "ldr3" (0 bis 1024)
73  if (ldr3 > 400)
74  {go = 2; digitalWrite(out2Pin, HIGH); }             // go auf 2 setzen, LED-grün ein
75  if ((go == 2) && (ldr3 > 400)) {hoehe = hoehe3; digitalWrite(out1Pin, LOW); } // Höhe 3, LED-rot aus
```

Erklärung:

In der Zeile „`59 schritt = (3*(hoehe - distance));`“ wird die Größe des nächsten Schritts berechnet, große Differenz zwischen „hoehe“ (SOLL-Wert) und „distance“ (IST-Wert) ergibt einen großen Schritt und umgekehrt.

Nähert sich der IST-Wert dem SOLL-Wert werden die Schritte immer kleiner und die Gleitgeschwindigkeit ebenso. Ist der IST-Wert größer, als der SOLL-Wert wird der „Schritt“ negativ, was bedeutet, dass die Motordrehzahl unter die Schwebe-Drehzahl fällt und das Modul zu sinken beginnt.

Diese Funktion ist in der Zeile „`61 if ((go == 1) && (distance != hoehe)) {(t1 = (schweb + schritt));}`“ mit „distance ungleich (!=) hoehe“ zu erkennen.

3.3.3. Gesamtprogramm für Adruino UNO

Die fertige INO-Datei hier herunterladen

```
/* modul-gesamtprogramm ultraschall plus pwm
datei: 20210327-gesamtprogramm-17-fertig-mit_motor
*/

// defines pins numbers ultraschall
const int trigPin = 9;
const int echoPin = 8;
const int aPin = 3;
const int impPin = 5;

// defines pins led
const int out1Pin = 11;      // LED-Pin 11
const int out2Pin = 12;      // LED-Pin 12
const int ldr1Pin = A0;      // Variable ldr übernimmt den Wert von A0, Start und Höhe 1
const int ldr2Pin = A1;      // Variable ldr übernimmt den Wert von A1, Start und Höhe 2
const int ldr3Pin = A2;      // Variable ldr übernimmt den Wert von A2, Landung

int ldr1 = 0;
int ldr2 = 0;
int ldr3 = 0;

// defines pins pwm
int pwPin = 10;     // PWM-Ausgang DigitalPin 10

// defines variables ultraschall
long duration;
int distance;
int anzeige;         //derzeit nicht verwendet

// defines variables pwm
int t1 = 0;           // Dauer des Impulses der PWM
int zahl = 0;         // Zahl für automatischen Ablauf
int go = 0;           // Marke für Programmbedingung
int hoehe = 0;        // aktuelle SOLL-Höhe
int hoehe1 = 0;       // 1. Höhenvorgabe
int hoehe2 = 0;       // 2. Höhenvorgabe
int hoehe3 = 0;       // 3. Höhenvorgabe
int schweb = 0;       // t1 für Schwebe-PWM
int schritt = 0;      // Differenz t1 zur Schwebe-PWM

void setup() {
// setup für ultraschall
pinMode(trigPin, OUTPUT); // Setzen des trigPin auf Ausgang
pinMode(echoPin, INPUT);  // Setzen des echoPin auf Eingang
pinMode(impPin, OUTPUT);  // Setzen des impPin auf Ausgang

// setup für pwm
  pinMode(pwPin, OUTPUT);     // setzen des pwPin auf Ausgang

// setup für LDR und LED
  pinMode(ldr1Pin, INPUT);    // setzen des LDR1Pin auf Eingang
  pinMode(ldr2Pin, INPUT);    // setzen des LDR2Pin auf Eingang
  pinMode(ldr3Pin, INPUT);    // setzen des LDR3Pin auf Eingang
  pinMode(out1Pin, OUTPUT);   // setzen des out1Pin auf Ausgang
  pinMode(out2Pin, OUTPUT);   // setzen des out2Pin auf Ausgang

// EINGABE! Folgende Werte VOR dem Start ermitteln und eingeben
  schweb = 1375;           // t1 für Schwebedrehzahl - diesen Wert am Modell ermitteln
  hoehe1 = 10;             // erste Höhe in cm
  hoehe2 = 20;             // zweite Höhe in cm
  hoehe3 = 2;              // dritte Höhe in cm, Landestellung
  t1 = 1000;               // t1 für den Start des ESC mit Drehzahl Null
  Serial.begin(9600);      // Start der seriellen communication
}                          // Ende der Initialisierung
```

Die Variable „go" dient dazu, bestimmte Programmteile zu nutzen, bzw. auszuschalten. Sie kann den Wert „0" für Warteposition ohne Motor, „1" für Start und Betrieb in verschiedenen Höhen und „2" für den Landevorgang mit Motorabstellung nach Landung annehmen.

```
52  void loop() {

53  // PWM
54    digitalWrite(pwPin,HIGH);  // pwPin HIGH-setzen
55    delayMicroseconds(t1);     // HIGH-Zeit von 1 bis 2 ms, t1 wird im SETUP erstmalig auf Wert 1
    gesetzt, Motorstillstand, aber ESC aktiv
56    digitalWrite(pwPin,LOW);   // pwPin Null-setzen
57    delay(5);                  // LOW-Zeit, beliebig von 2ms bis 30ms, dieser Wert wird um die
    restliche Prozesszeit erhöht.

58  // Höhenfeststellung und Entscheidung

59       schritt = (3*(hoehe - distance));                                    // t1-Schritt für
    Drehzahländerung
60       if ((go == 1) && (distance == hoehe)) {(t1 = schweb);}               // t1 für richtige Höhe …

61       if ((go == 1) && (distance != hoehe)) {(t1 = (schweb + schritt));}// t1 für ungleiche höhe

62       if ((go == 2) && (distance > (hoehe - 1))) {(t1 = schweb-40);}    //alternativ doch
    sinkdrehzahl festlegen und nutzen
63       if ((go == 2) && (distance < hoehe)) {t1 = 1000; go = 0; digitalWrite(out2Pin, LOW);}    //t1
    für nulldrehzahl, go=0

64  //LDR
65       ldr1 = analogRead(ldr1Pin);      // Auslesen des ldrPin und Zuweisung zu "ldr1" (0 bis 1024)
66       if (ldr1 > 400) {go = 1;}                                              // go auf 1 setzen

67       if ((go == 1) && (ldr1 > 400))
68       {hoehe = hoehe1; digitalWrite(out1Pin, HIGH); t1 = (schweb + 30); }  // Höhe 1, LED-rot ein

69       ldr2 = analogRead(ldr2Pin);      // Auslesen des ldrPin und Zuweisung zu "ldr2" (0 bis 1024)
70       if ((go == 1) && (ldr2 > 400))
71       {hoehe = hoehe2; digitalWrite(out1Pin, HIGH); } // Höhe 2, LED-rot ein

72       ldr3 = analogRead(ldr3Pin);      // Auslesen des ldrPin und Zuweisung zu "ldr3" (0 bis 1024)
73       if (ldr3 > 400)
74       {go = 2; digitalWrite(out2Pin, HIGH); }             // go auf 2 setzen, LED-grün ein
75       if ((go == 2) && (ldr3 > 400)) {hoehe = hoehe3; digitalWrite(out1Pin, LOW); } // Höhe 3

76  // ultraschall:

77     digitalWrite(trigPin, LOW);    // LÖSCHE trigPin
78     delayMicroseconds(2);          // Verzögerung 2µs
79     digitalWrite(trigPin, HIGH);   // Setze trigPin auf HIGH für 10 micro seconds
80     delayMicroseconds(10);         // Verzögerung 10µs
81     digitalWrite(trigPin, LOW);    // LÖSCHE trigPin

82     duration = pulseIn(echoPin, HIGH);   // Lese echoPin, Schallreflexion in µs

83     distance= duration*0.034/2;          // Berechnung der Distance in cm

84      digitalWrite(impPin, HIGH);
85      delayMicroseconds(100);
86      digitalWrite(impPin, LOW);
87  }                                    // Ende der Schleife (loop)
```

3.3.4. Zur Feststellung der Schwebedrehzahl wird der ESC vom Arduino getrennt und an den PWM-Impulssimulator (Bauanleitung hier herunterladen) angeschlossen. Ebenso wird ein Oszilloskop mit dem Simulator zur Messung der Impulsbreite verbunden.

Mit dem Regler des Simulators wird das Modul auf eine beliebige Höhe gebracht und dann auf den Schwebezustand zurückgeregelt. Die Impulsbreite kann am Bildschirm gespeichert und gemessen werden. Es ist ein Wert etwa in der Mitte zwischen 1000 und 2000 µs.

Dieser gemessene Wert wird in die Zeile 45 der Initialisierung als Größe der Variablen „schweb" eingetragen (`45  schweb = 1375;     // t1 für Schwebedrehzahl`).

3.3.5. Zum Erproben des Programms können statt der Ansteuerung des Brushless-Motors alle Werte auf dem Systemmonitor ausgegeben werden. Dazu ist es erforderlich:

- 1. In der Schleife (loop) folgenden Teil zu mit /* */ zu deaktivieren:

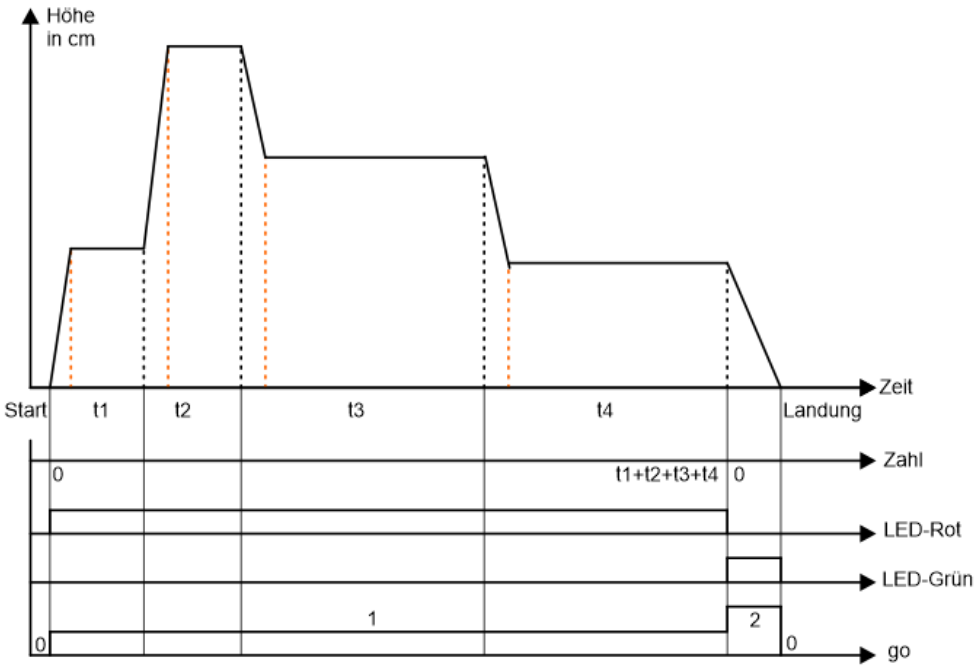
```
  54    /*  digitalWrite(pwPin,HIGH);  // pwPin HIGH-setzen
  55    delayMicroseconds(t1);     // HIGH-Zeit von 1 bis 2 ms, t1 wird im SETUP erstmalig auf Wert 1
gesetzt, Motorstillstand, aber ESC aktiv
  57    digitalWrite(pwPin,LOW);   // pwPin Null-setzen
  58    delay(5);                  // LOW-Zeit, beliebig von 2ms bis 30ms, dieser Wert wird um die
restliche Prozesszeit erhöht.
       */
```

- 2. Vor dem Schleifenende folgende Sequenz eingeben oder, wenn schon vorhanden und deaktiviert, zu aktivieren. Gleichzeitig zum Motorbetrieb ist dieses Monitoring nicht möglich.

```
088  // Variable auf Monitor
089      Serial.print("zahl: ");
090      Serial.print(zahl);
091      Serial.print(" ldr: ");
092      Serial.print(ldr1);
093      Serial.print(" / ");
094      Serial.print(ldr2);
095      Serial.print(" / ");
096      Serial.print(ldr3);
097      Serial.print("  Distance: ");
098      Serial.print(distance);
099      Serial.print("  schritt: ");
100      Serial.print(schritt);
101      Serial.print("  go: ");
102      Serial.print(go);
103      Serial.print("  t1: ");
104      Serial.print(t1);
105      Serial.print("  hoehe: ");
106      Serial.println(hoehe);
```

4. Automatische Ablaufsteuerung mittels Zählers:

Zeitliche Verläufe lassen sich beim Arduino mit dem Befehl „delay" realisieren. Die Zeit wird dabei in Milli- oder Mikrosekunden angegeben.



Um einen zeitlichen Verlauf zu programmieren, wie dieser im rechten Diagramm zu sehen ist, gibt es zwei Ansätze:

- Alle Detailprogramme, wie PWM, Distanzmessung, Startabfrage, etc. werden in Unterprogramme ausgelagert und die Hauptprogrammschleife besteht aus den Teilen der einzelnen Schwebezustände mit t_1 bis t_n.

- Oder es gibt wie in den vorigen Programmen eine einzige Schleife (Loop), in der alle Detailaufgaben in der *Pause des PWM-Signales* bearbeitet werden. Dieser Ansatz wurde in diesem Beispiel und in den anderen vorangegangenen Programmen gewählt.

Da die Zeit der einzelnen Schwebezustände nicht mit dem **delay**-Befehl gelöst werden kann (diese Zeit würde in das PWM-Signal eingehen, was die Steuerung des ESC verunmöglicht), wurde die Zeitsteuerung über einen Zähler gelöst, der bei jedem Schleifendurchgang um 1 erhöht wird.

Dazu wurde die Zeit eines Schleifendurchganges mit dem Oszillogramm des PWM-Signales (Impuls plus Pause, inklusive aller Zusatzfunktionen) mit etwa 10ms gemessen. In einer Sekunde wird der Zähler um den Wert 100 erhöht.

Der Zähler wird im SETUP und nach der Landung auf 0 gesetzt und mit dem Startbefehl des Moduls beginnt die Zählung.

4.1. Werte im SETUP, die Höhen und die Schwebedauer werden eingetragen.

```
schweb = 1365              // Schwebedrehzahl, vorher am Modul ermitteln
hoehe1 = 10;               // erste Höhenvorgabe in cm eingeben
hoehe2 = 20;               // zweite Höhenvorgabe in cm eingeben
hoehe3 = 10;               // dritte Höhenvorgabe in cm eingeben
hoeheland = 2,5;           // Lande-Höhenvorgabe in cm eingeben
dauer1 = 10;               // dauer der 1. Höhe in sec
dauer2 = 15;               // dauer der 2. Höhe in sec
dauer3 = 10;               // dauer der 3. Höhe in sec
zfakt = 100;               // Zeitfaktor für Zähler
t1 = 1000;                 // Für den Start des ESC für Drehzahl Null
zahl = 0;                  //Zählerwert während des Schwebebetriebes
```

In diesem Beispiel wurden nur drei Höhen mit jeweiliger Schwebedauer angegeben.

Die Variable „schweb" gibt dem ESC die 1,365ms (1365µs) für die Schwebedrehzahl vor. Dieser Wert muss mit dem Modul erprobt und festgelegt werden.

Die Variable „hoeheland" gibt die Höhe an, bei dem der Motor abgeschaltet wird (t_1 = 1000) und das Modul aufsetzt.

Die Variable „zfakt" ist jener Faktor, der die Eingabe der Verweilzeit in Sekunden in die Zahl des Zählers umrechnet. Damit ist die Eingabe der Zeit in Sekunden möglich.

Die Variable „t1" ist für die Nulldrehzahl des ECS nötig, nämlich 1ms (1000µs), die den ESC auch initialisiert.

Die Variable „zahl" ist die Zahl des aktuellen Wertes des Zählers.

Weitere Höhen mit zugehöriger Schwebedauer können beliebig hinzugefügt werden.

WICHTIG: Die Landung wird nach der Summe aller Schwebezeiten t_1 bis t_n eingeleitet.

4.2. Abarbeiten des Zählers, Feststellen des Schwebezustands (bei go ist gleich 1)

Die Variable „go" nimmt drei Werte an, **0** für Wartemodus des Moduls, **1** für den Schwebebetrieb und **2** für den Landebetrieb, go legt also den Betriebszustand fest. Es sind auch weitere erweiterte Betriebszustände möglich.

Gewünschte Höhe erkennen:

```
if ((go == 1) && (distance == hoehe)) {(t1 = schweb);}  //richtige Höhe
```

Wenn go gleich 1 ist **und** die gemessene Höhe **ist gleich** der gewünschten Höhe, **dann** beim ESC die Schwebedrehzahl einstellen.

In den nächsten drei plus einer IF-Abfrage werden entsprechend des Zählerwertes der Höhen-SOLL-Wert festgelegt und die aperiodische Erreichung geregelt:

```
if ((go == 1) && zahl < ((dauer1 * zfakt)+(dauer2 * zfakt)+(dauer3 * zfakt)) &&
(distance != hoehe)) {hoehe = hoehe3; (t1 = (schweb + schritt));}

if ((go == 1) && zahl < ((dauer2 * zfakt)+(dauer3 * zfakt)) && (distance != hoehe))
{hoehe = hoehe2; (t1 = (schweb + schritt));}

if ((go == 1) && zahl < ((dauer3 * zfakt)) && (distance != hoehe)) {hoehe = hoehe1;
(t1 = (schweb + schritt));}
```

Wenn go gleich 1 ist und der aktuelle Zählerwert **kleiner ist**, als der gewünschte Zählerwert **und** die gemessene Höhe **ungleich** der gewünschten Höhe ist, **dann die** Sollhöhe festlegen und die Drehzahländerung veranlassen (schweb + schritt, wie im Punkt 3.3. beschrieben).

Um ohne „else" auszukommen wird bei der Abfrage des Zählers bei steigendem Wert mit der größten Zahl (Summe aller Einzel-Schwebezeiten) begonnen und den Prozessschritten entsprechend reduziert, sodass der Wert der letzten passenden Überprüfung vom Prozess bleibend übernommen werden kann. Details dazu siehe Punkt 4.4. und 4.4.1 auf den nächsten beiden Seite.

Die letzte Zählerabfrage betrifft die Höchstzahl des Zählers (Summe aller Einzelzeit-Schwebezeiten), um die Landung einzuleiten:

```
if ((go == 1) && zahl > ((dauer1 * zfakt)+(dauer2 * zfakt)+(dauer3 * zfakt))) {go =
2; hoehe = hoeheland; digitalWrite(out2Pin, HIGH);}   // höchstzahl überschritten
```

Wenn „go" gleich „1" ist und der aktuelle Zählerwert **größer ist**, als die Summe aller Einzelzeit-Schwebezeiten, **dann „**go" auf „2" setzen (Landemodus), Sollhöhe auf Landehöhe setzen und die grüne LED aktivieren.

Wie schon auf Seite 15 beschrieben, bestimmt die Variable „go" den Betriebszustand des Moduls.

4.3 Landung (go ist gleich 2 - Modus)

```
if ((go == 2) && (distance > (hoehe - 1))) {(t1 = schweb-80);}
```

Wenn go gleich 2 ist **und** die gemessene Höhe **größer** als die Landehöhe ist, **dann** sinkt das Modul mit Schwebedrehzahl minus 80.

```
if ((go == 2) && (distance < hoehe)) {t1 = 1000; go = 0; digitalWrite(out2Pin, LOW);
digitalWrite(out1Pin, LOW); zahl = 0;}    //nulldrehzahl, Warteposition
```

Wenn go gleich 2 ist und die gemessene Höhe **kleiner** als die Landehöhe ist, **dann** go auf Null setzen (Warteposition), die grüne und rote LED löschen und „zahl" auf 0 setzen.

Der Unterschied zwischen „**go == 2**" und „**go = 2**" besteht darin, dass „**go == 2**" bedeutet go ist gleich 2 und „**go = 2**" bedeutet die Zuweisung der Variablen „go" mit dem Wert „2".

In diesem Programm wurden folgende Operanden verwendet:

„==" …. ist gleich,
„!=" …. ist ungleich,
„>" ……. größer als,
„<" ……. kleiner als.

Das **fertige Programm** als gezipte INO-Datei ist hier zum Herunterladen bereit.

4.4. Reihenfolge der IF-Abfrage, einfaches Beispiel:

```
/* datei: zaehler.ino */

int zahl = 0;                 //Variable deklarieren und 0-setzen
int a = 0;                    //Variable deklarieren und 0-setzen

void setup() {

  Serial.begin(9600);         // Start der seriellen Kommunikation
}

void loop() {

 zahl = zahl + 5;             //Zähler in 5er-Schritten erhöhen

     if (zahl>500) {zahl = 0;}

     if (zahl <500) {a = 1;}
     if (zahl <400) {a = 2;}
     if (zahl <300) {a = 3;}
     if (zahl <200) {a = 4;}
     if (zahl <100) {a = 5;}

     delay (100);

// Variable auf Monitor
    Serial.print("zahl: ");
    Serial.print(zahl);
    Serial.print(" Wert: ");
    Serial.println(a);

}
```

```
Die umgekehrte Reihenfolge ergibt für a immer 1

if (zahl <100) {a = 5;} Datei:zaehler-verkehrt.ino
if (zahl <200) {a = 4;}
if (zahl <300) {a = 3;}
if (zahl <400) {a = 2;}
if (zahl <500) {a = 1;}
```

4.4.1 Erklärung der richtigen Reihenfolge:

```
1   if (zahl <500) {a = 1;}
2   if (zahl <400) {a = 2;}
3   if (zahl <300) {a = 3;}
4   if (zahl <200) {a = 4;}
5   if (zahl <100) {a = 5;}
```

Ist der Wert von „zahl" 50, stimmt die Abfrage in Zeile 1 und „a" wird der Wert „1" zugewiesen. Aber die Abfrage stimmt auch für die Zeilen 2 bis 5, „a" wird dann jeweils 2, 3, 4, 5, zugewiesen. Nur für den weiteren Prozess behält „a" den letzten zugewiesenen Wert nämlich „5".

Die „unnötigen" Zuweisungen bedürfen nur wenige µs im Programm und sind daher zu vernachlässigen.

Ist der Wert von „zahl" 250, stimmt die Abfrage in Zeile 1 und „a" wird der Wert „1" zugewiesen. Für die Zeilen 2 und 3 stimmt die Abfrage auch und „a" wird jeweils, 2 und 3 zugewiesen. Für die Zeilen 4 und 5 stimmt die **Abfrage nicht** und so ist für den weiteren Prozess „a" der letzte zugewiesene Wert nämlich „3" gültig.

Ist der Wert von „zahl" 450, stimmt die Abfrage nur in Zeile 1 und „a" wird der Wert „1" zugewiesen. Die Beispiel-Dateien „zaehler.ino" und „zaehler-verkehrt.ino" sind hier zum Herunterladen und Experimentieren bereit.

5. Linkliste:

Video-PWM-Erklärung
https://nw-service.at/wp-content/uploads/2021/04/pwm-19MB-korr.webm

Probleme bei Parallelschaltung von Akku und Netzgerät
https://nw-service.at/wp-content/uploads/2021/04/Netzgeraet-Akku-2.pdf

Video Zweipunktregelung
https://nw-service.at/wp-content/uploads/2021/04/einfluegel-2Punktreg-9MB.webm

Dreipunktregelung
https://nw-service.at/wp-content/uploads/2021/04/modul-3-punkt-5MB.webm

Video Aperiodische Regelung
https://nw-service.at/wp-content/uploads/2021/04/modul-aperi-regel-1-100MB.mp4
https://nw-service.at/wp-content/uploads/2021/04/modul-aperi-regel-1-7MB.webm

Programm Aperiodische Regelung
https://nw-service.at/wp-content/uploads/2021/04/20210403-gesamtprogramm-17-motor.zip

3D-Drucker-STL-Dateien
https://nw-service.at/wp-content/uploads/2021/04/modul-stl-dateien.zip

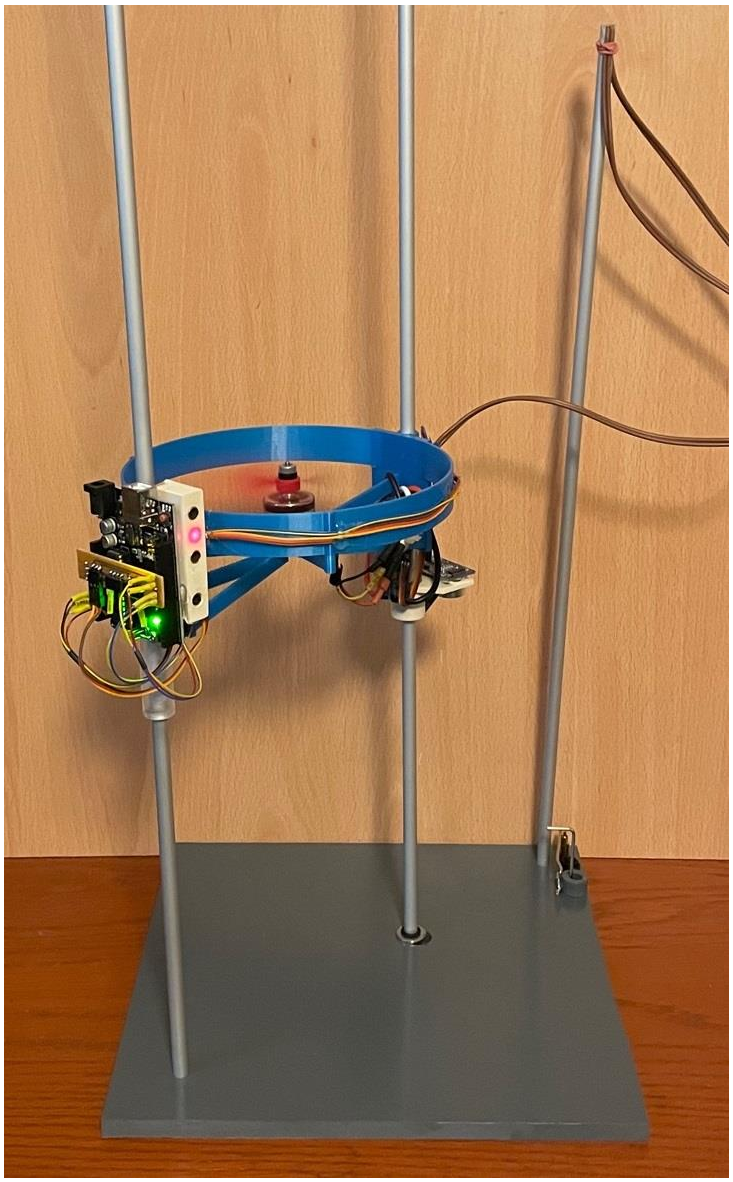
Fertiges Programm
https://nw-service.at/wp-content/uploads/2021/05/20210529-gesamtprogramm-24-zahl-4-posit.zip

6. Kontakt:

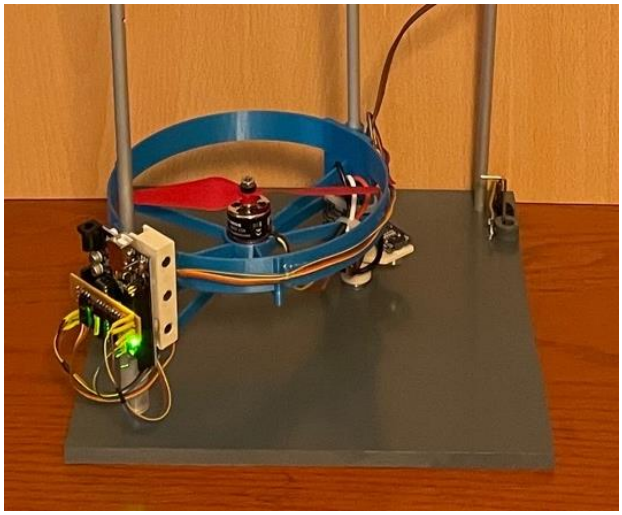
https://nw-service.at/, n.willmann@liwest.at

7. Bilder

7.1. Rechts, fertiges Modul im Flug:



7.2. Unten, Modul gelandet:



8. Technische Daten:

Masse: 215g

Schub: 10N bei 10A Stromaufnahme

Stromaufnahme beim Schweben: 5A

Maximaler Strom: 20A

Zellenzahl (LiIonen): 3 für 12V Betriebsspannung, maximal 4

Abmessung mit Verdrahtung L, B, H: 260 x 170 x 115

Propellerdurchmesser: 150mm

Inhaltsverzeichnis:

Gesteuerter Propeller-Antrieb mittels Arduino