Erneuerung der Peilersteuerung

Die Peiler von DJ4TA werden von einem Notebook gesteuert. Sie enthalten:

- Motor
- Winkelgeber
- Stromversorgung

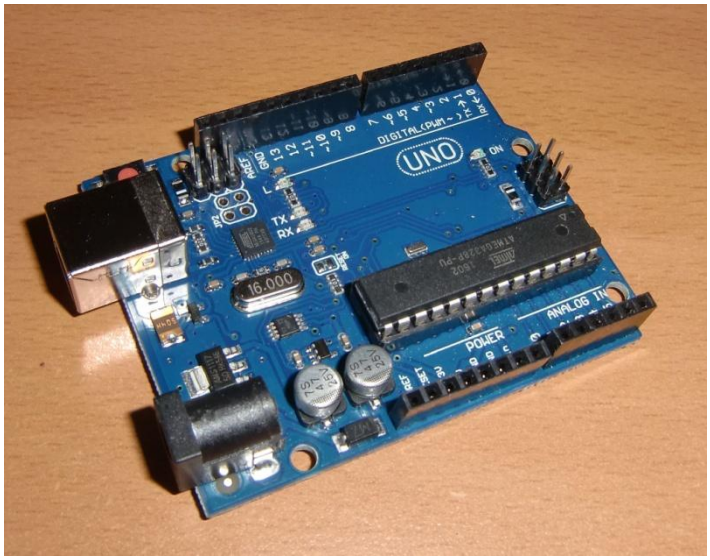
Die Steuerung erfolgt über eine RS232 Verbindung zum Notebook. Auf dem Notebook läuft das Programm „Hunter“ von Jan, das Motorsteuersignale zum Peiler schickt. Der Peiler erfaßt die richtungsabhängigen S-Meter-Werte und gibt diese als ASCII-String (Richtung/Stärke) über die RS232 Strecke an das Notebook zur Auswertung weiter.

Im Peiler sind zwei ATMEL Prozessoren, von denen einer die Winkelgebersignale in Gray-Code in ein für uns lesbares Format umwandelt und mit dem S-Meter-Signal verbindet. Der zweite steuert den Motor (rechts, links, schnell, langsam). Beide Prozessoren sind in Assembler programmiert und entsprechen einem 15 Jahre alten Technologiestand.

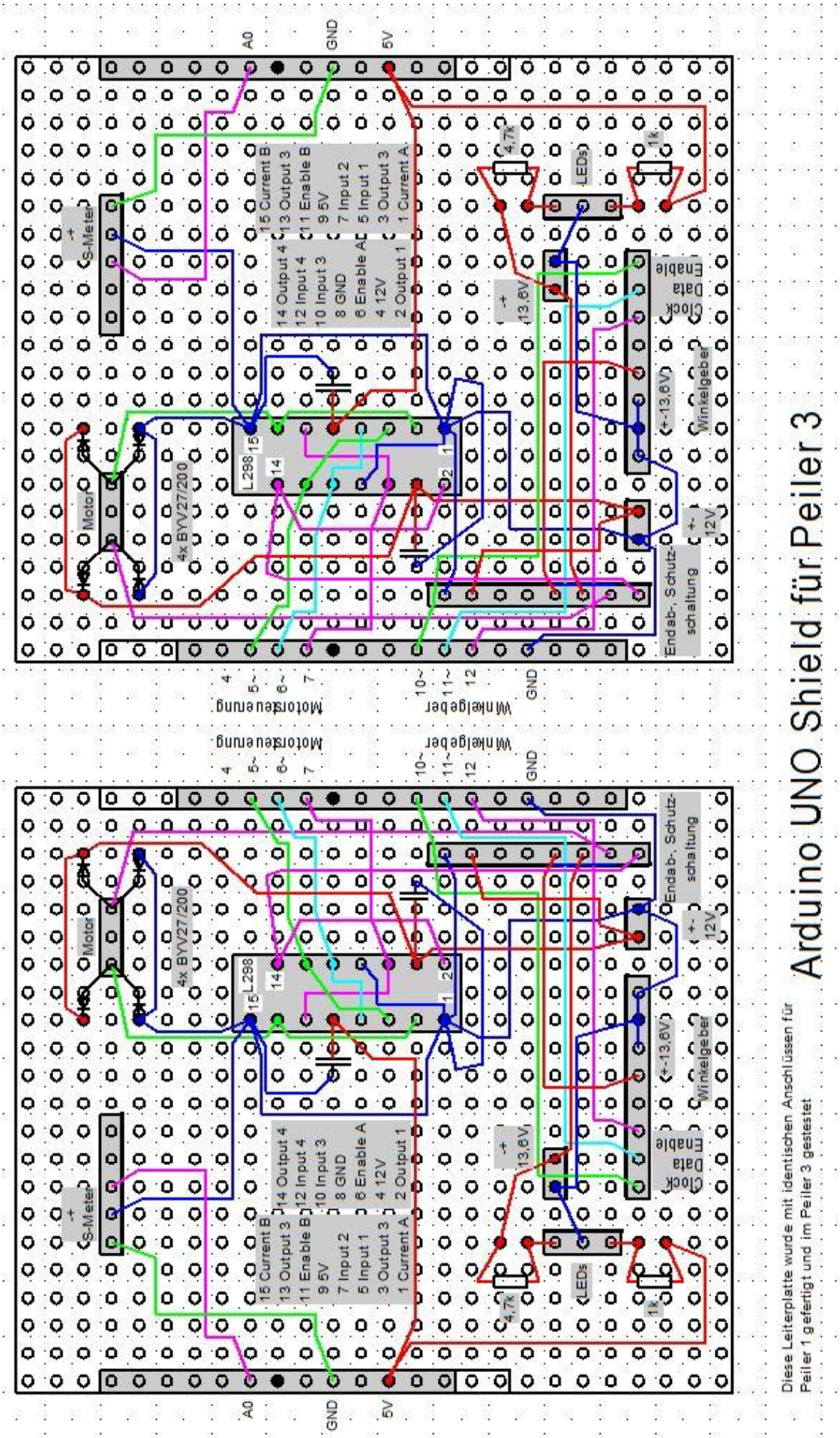
Eine funktionierende Technik sollte man nicht ändern sofern es keine Probleme gibt. Beim Peiler 3 versagte die Motorsteuerung und wir mußten eine Lösung finden. Der Verdacht fiel auf den ATMEL Prozessor.

Ein neuer Prozessor muß aber erst programmiert werden. Da es aber heute eine modernere Technik gibt, die leistungsfähiger ist und sich über eine Hochsprache programmieren läßt, haben wir uns entschlossen, die Steuerung im Peiler komplett zu erneuern. Infrage dafür kam ein Arduino UNO, der so leistungsfähig ist, daß er beide Prozessoren ersetzen kann und darüber hinaus in einem einfachen C zu programmieren ist.

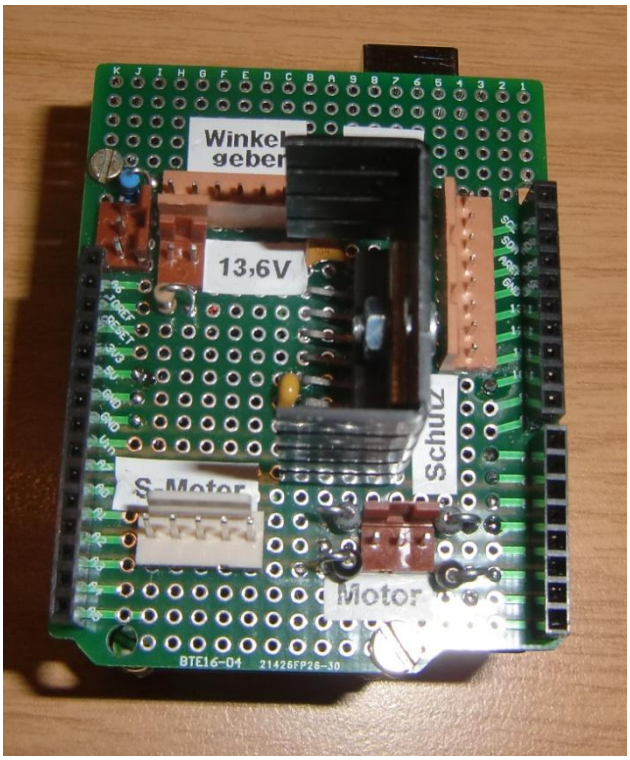
Die Verbindung zum Notebook erfolgt nun über eine USB Schnittstelle und wird sowohl für die Programmierung als auch für den Betrieb genutzt.



Der Arduino UNO verfügt über zwei Sockelleisten, die huckepack sogenannte Shield-Platinen aufnehmen können. Die Stromversorgung erfolgt über die USB-Schnittstelle. Eine externe Versorgung ist natürlich auch möglich.


Arduino UNO Shield für Peiler 3
Diese Leiterplatte wurde mit identischen Anschlüssen für Peiler 1 gefertigt und im Peiler 3 gestestet
A0
GND
5V
S-Meter
Motor
4x BYV27/200
L298
Motorsteuerung
Winkelgeber
LEDs
4,7k
1k
13,8V
Endab-, Schutz-schaltung
(+-13,6V)
12V
Winkelgeber
Enable
Data
Clock

Nun müssen die Hardwarekomponenten mit dem Arduino verbunden werden. Dazu wurde eine kleine Lochrasterplatine zugeschnitten und mit passenden Steckerleisten verbunden.

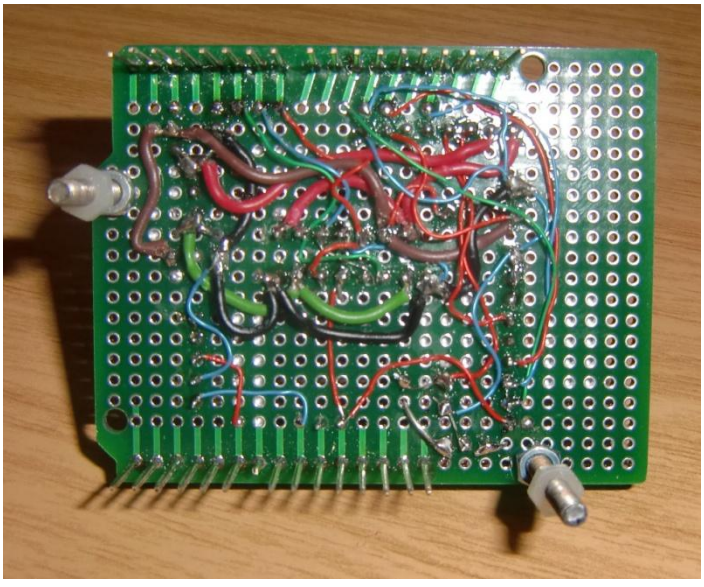


Es sind nur sehr wenige Bauteile erforderlich:

- Motorsteuer-IC L298N
- 4 Dioden (z. B. BYV27/200)
- 2 Kondensatoren 0,1uF
- 6 Steckerleisten für
  - 12V
  - 13,6V
  - Motor
  - Winkelgeber
  - S-Meter
  - Endabschaltung/ Überspannungs-schutz

Darüber hinaus sind auf der Rückseite die Steckerleisten zum Arduino und natürlich die Verdrahtung.

Da der Motor ca. 2A zieht, werden die beiden Steuerbrücken des ICs parallel geschaltet.



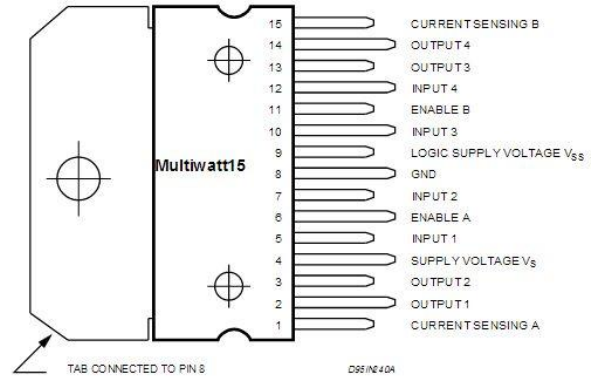
Die S-Meter-Werte gehen direkt auf den Analog-Digital-Wandler-Eingang des Arduino. Die Spannung muß entsprechend verstärkt werden, damit der gesamte AD-Wandler-Bereich ausgenutzt werden kann.

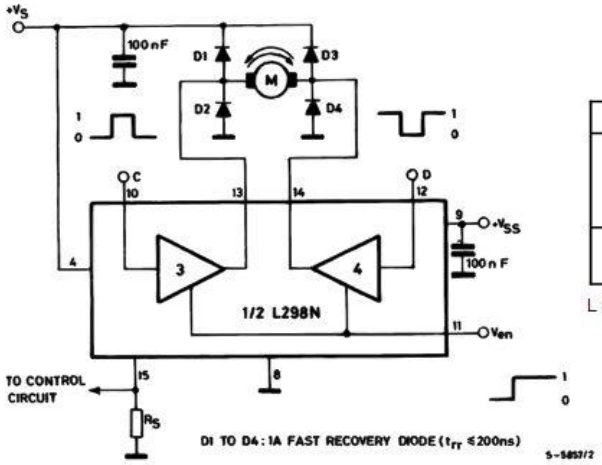
(0-5V in 1024 Stufen)

Motorsteuerung

L298N Anschlußbelegung. Beide Steuerbrücken werden parallel geschaltet.

Prinzip der Motorsteuerung mit L298 (Vorwärts, rückwärts, Stop):

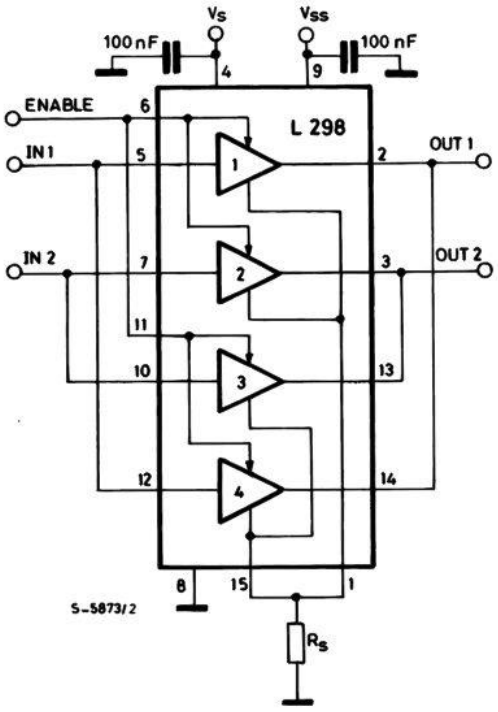




Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

DieVerdrahtung ist relativ einfach.

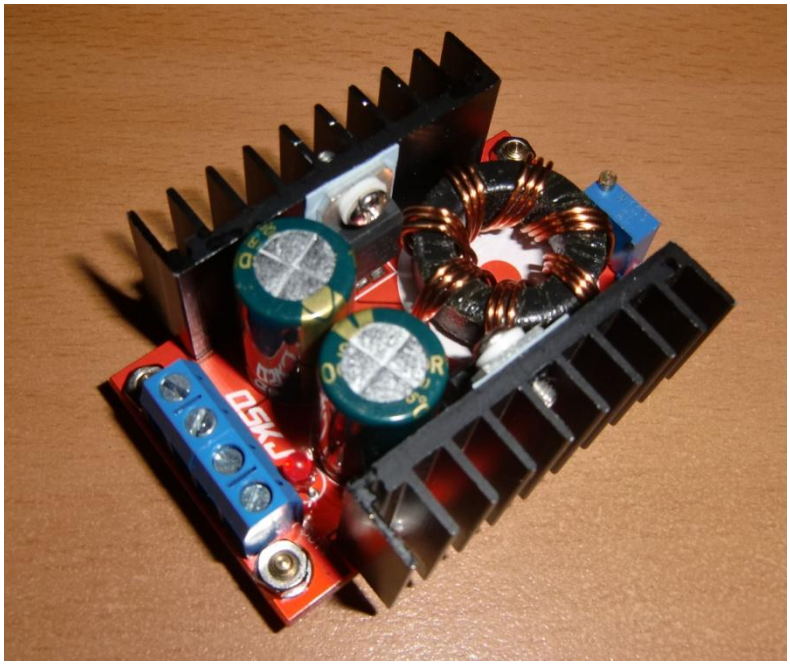


Hier die Parallelschaltung der

beiden Brücken

Winkelgeber

Der Stegmann Winkelgeber arbeitet nur zuverlässig, sofern die Betriebsspannung >13V ist. Das ist bei Batteriebetrieb nicht sicher gegeben. In der Vergangenheit habe ich Notebook Netzteile so abgeändert, dass die normalerweise niedrigste Ausgangsspannung von 15V um 1,4V durch Änderung eines Widerstandes reduziert habe.



Bei Amazon gibt es einen Step-up Regler für nur €7,-, bei dem praktisch eine beliebige Ausgangsspannung einstellbar ist

Ich versorge nun mit der transformierten Ausgangsspannung von 13,6V nur den Winkelgeber und nutze so die hohe Leistung nicht aus. Somit werden die Kühlkörper nicht einmal warm.

(Spannung 10-32V Eingang, 12-35V Ausgang, Boost Converter DC-DC Step-Up Adjustable Power)

Bisher habe ich keinerlei Beeinträchtigung durch Oberwellen des Schaltreglers feststellen können. Mal sehen, ob das so bleibt.

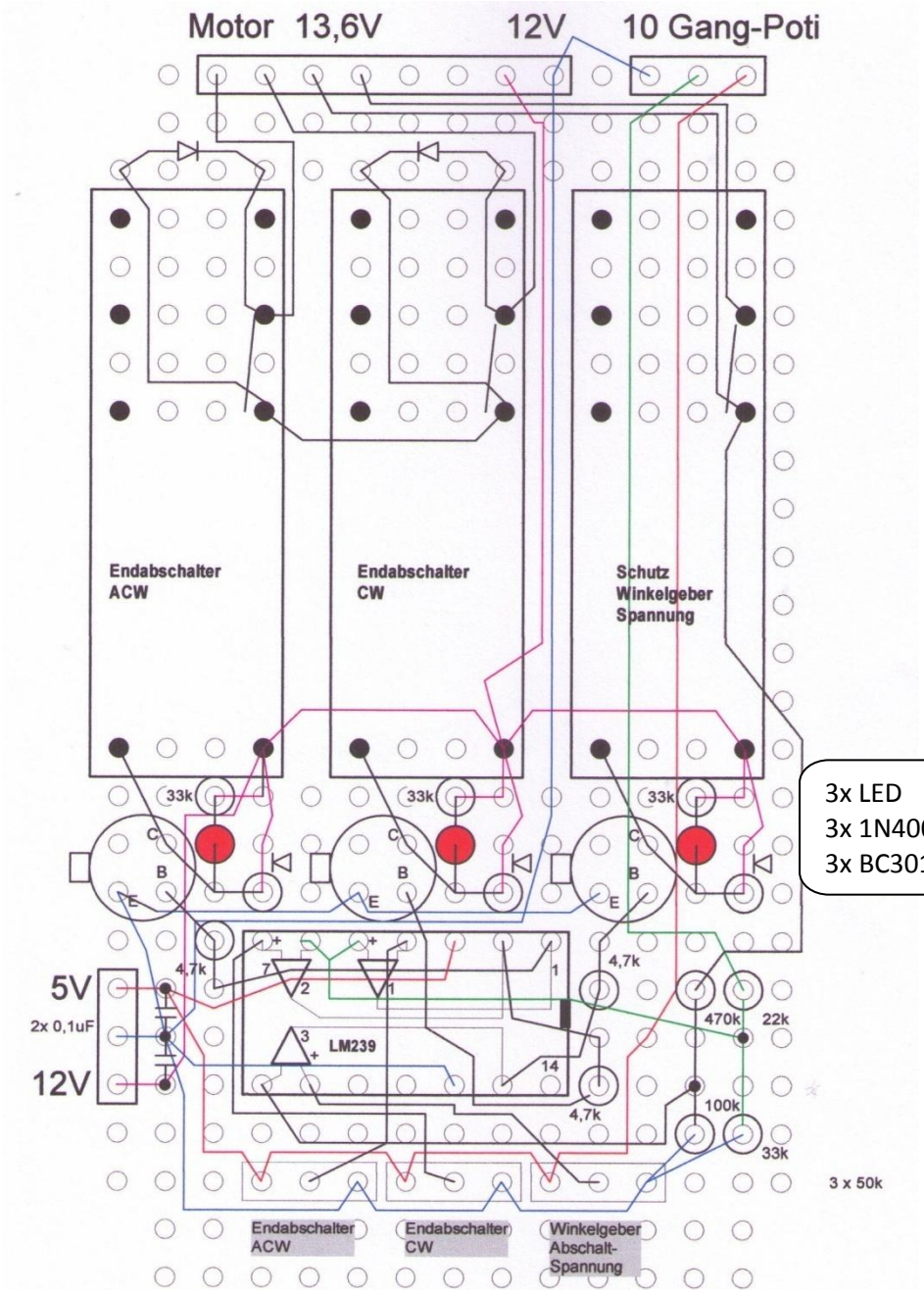
Schutzschaltung (Peiler 3)

Nachdem nun alles funktionierte, wollte ich doch eine Schutzschaltung vorsehen, die folgendes sicherstellt:

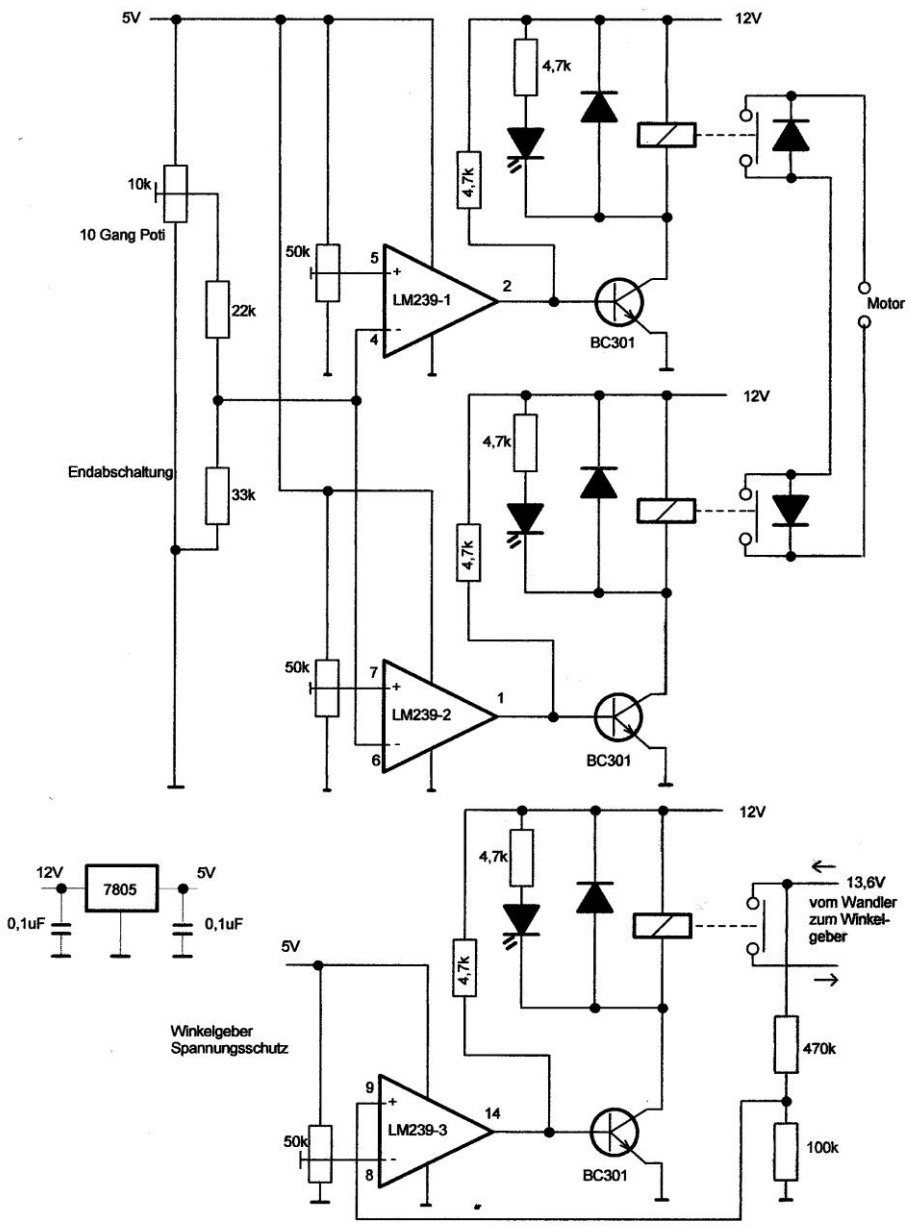
1. Eine Endabschaltung der Motoren, damit bei Störung im Rechner oder in der Verbindung zum Peiler verhindert wird, dass die Antenne sich unendlich dreht und damit die Kabel abreißt.

2. Der teure Winkelgeber darf keine Überspannung erhalten. Da ich noch keine Erfahrung mit dem chinesischen Wandler habe, habe ich eine Schutzschaltung vorgesehen.

Layout auf der Lochrasterplatine:



Schaltung:

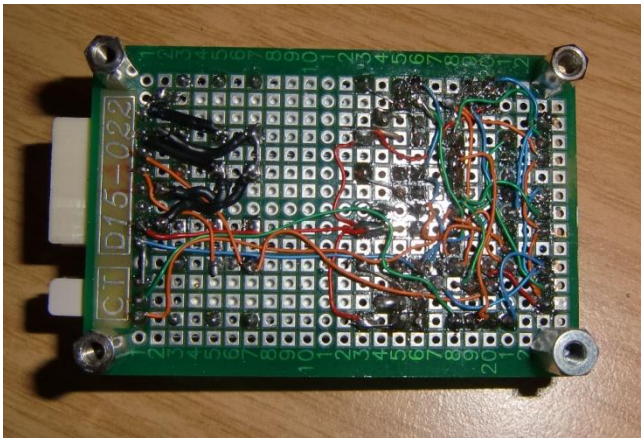


The LM239 operates solely as a voltage comparator, comparing the differential voltage between the positive and negative pins and outputting a logic low or high impedance (logic high with pullup) based on the input differential polarity.

Die Dioden über den Relais-kontakten stellen sicher, daß nach erfolgter Notendabschaltung die Antenne mit entgegengesetzter Polung der Motorspannung wieder in die gewünschte Position fahren kann.

Der LM239 verfügt über 4 Komparatoren, von denen nur 2 für die Endabschaltung benutzt werden.

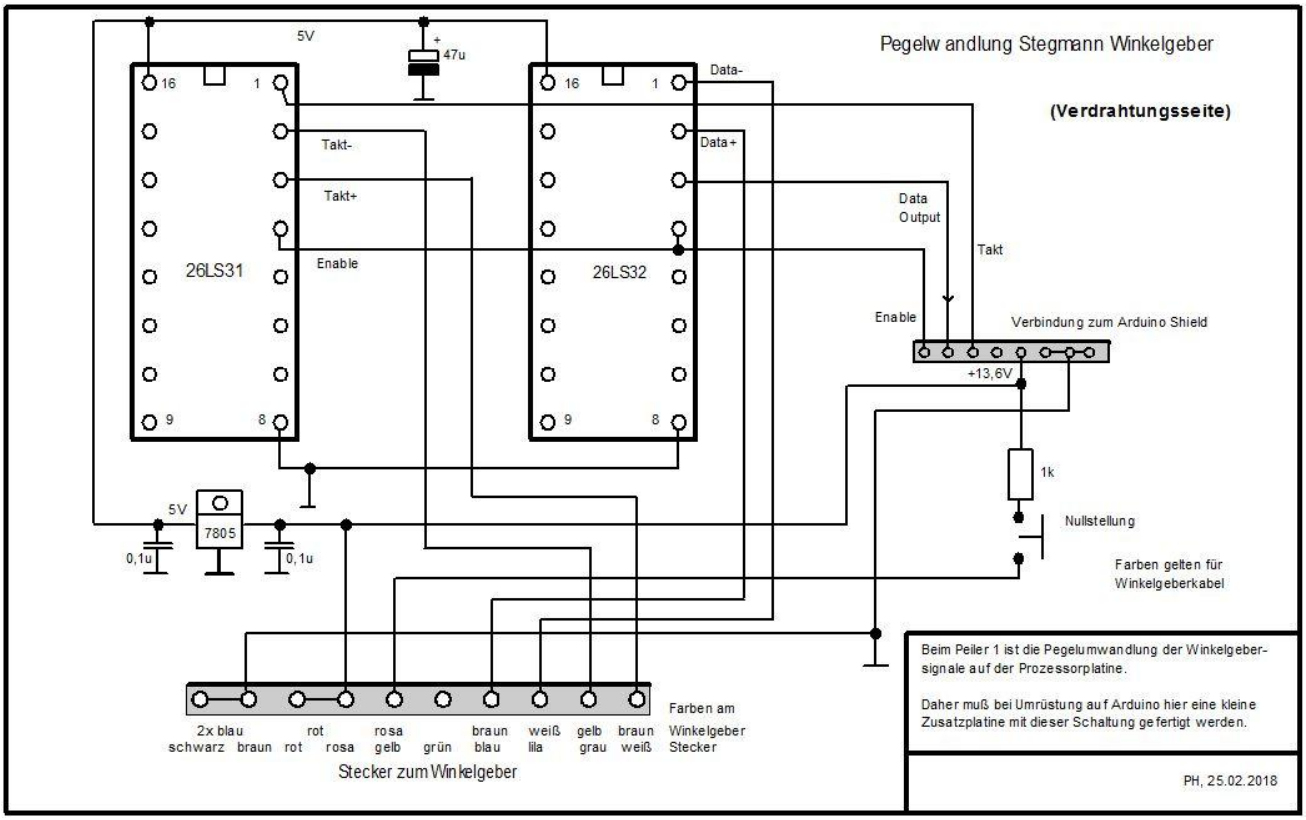
Somit konnte ich mit dem dritten die auf 13,6V transformierte Betriebsspannung für den Winkelgeber überwachen und gegebenenfalls abschalten.



Winkelgeberschnittstelle RS422 in TTL wandeln (Peiler 1)

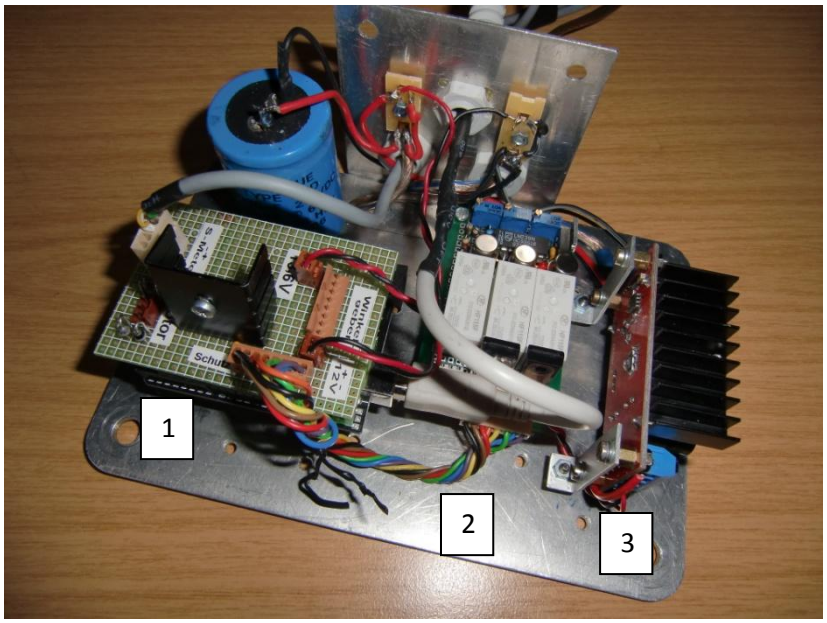
Beim Peiler 1 erfolgt die Schnittstellenanpassung auf der Prozessorplatine, die jetzt durch den Arduino ersetzt wurde. Daher habe ich eine kleine Platine mit den beiden ICs und der dazugehörigen 5V Versorgung gefertigt.

Die Schaltung:



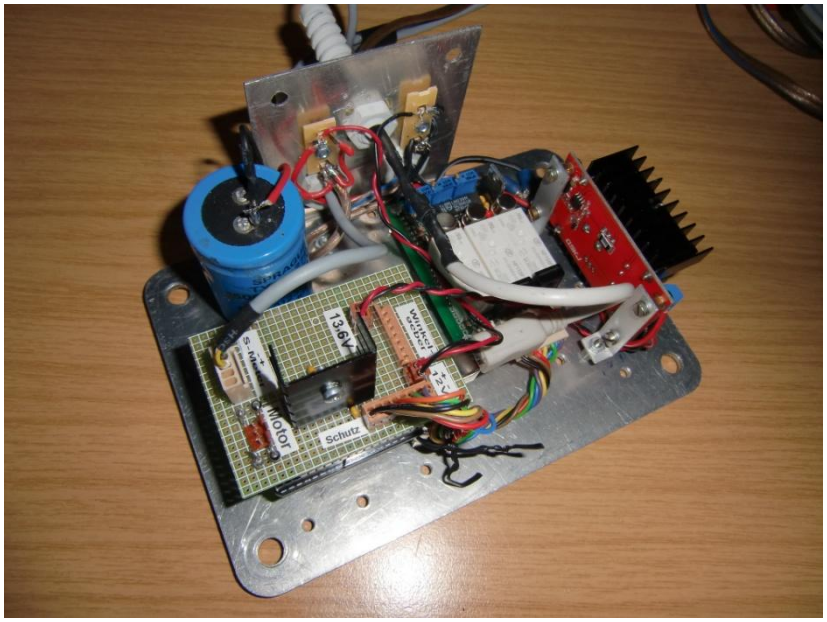
Peiler 3

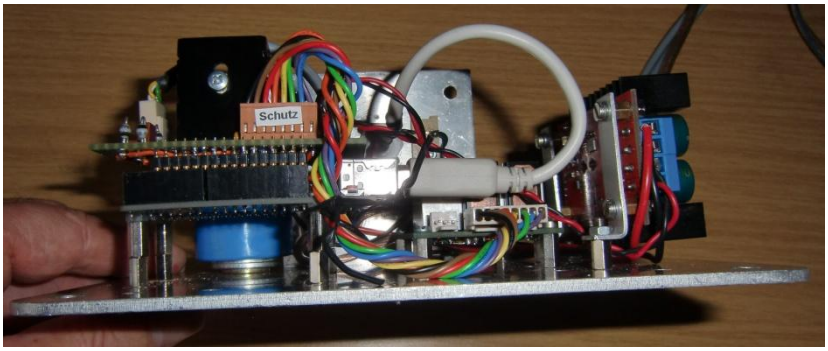
Die gesamte Elektronik des Peilers ist jetzt auf der Bodenplatte montiert.



Auf der Bodenplatte sind folgende Bausteine vorhanden:

1. Arduino UNO mit der Motorsteuerung huckepack
2. Endab- und Schutzschaltung
3. Step-up Wandler für 13,6V

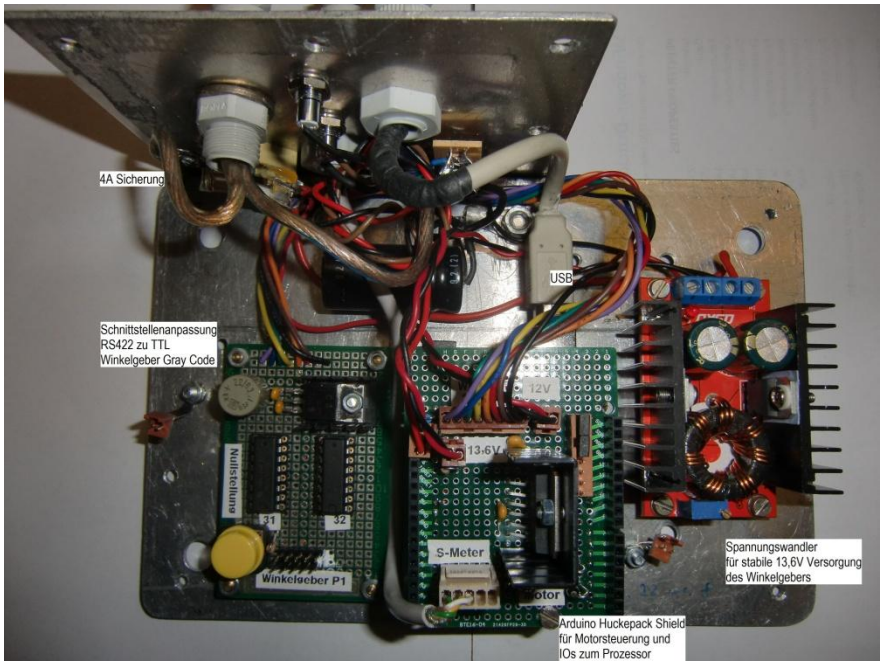

13,6V
Motor
Schutz

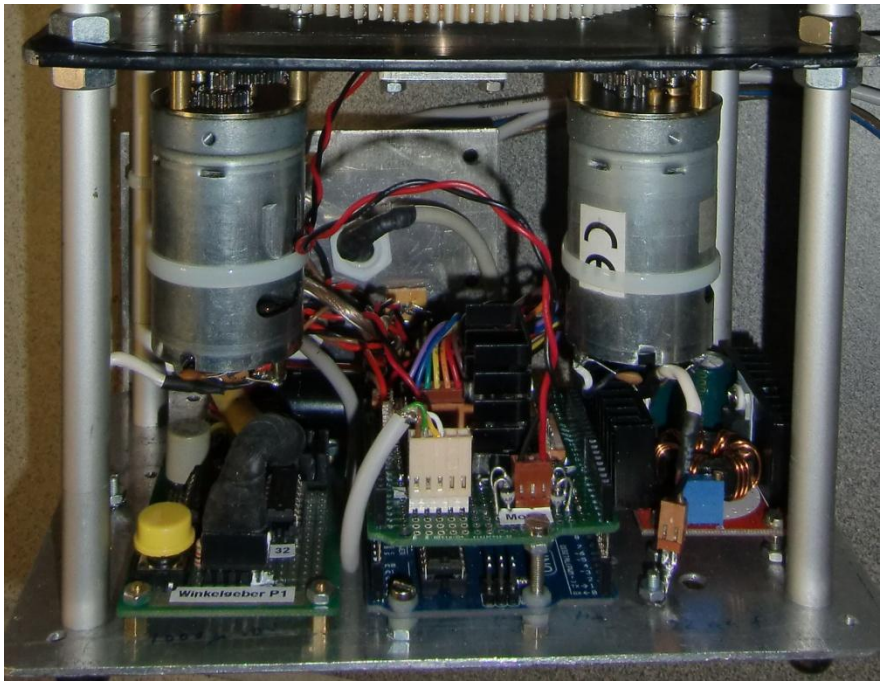

Schutz

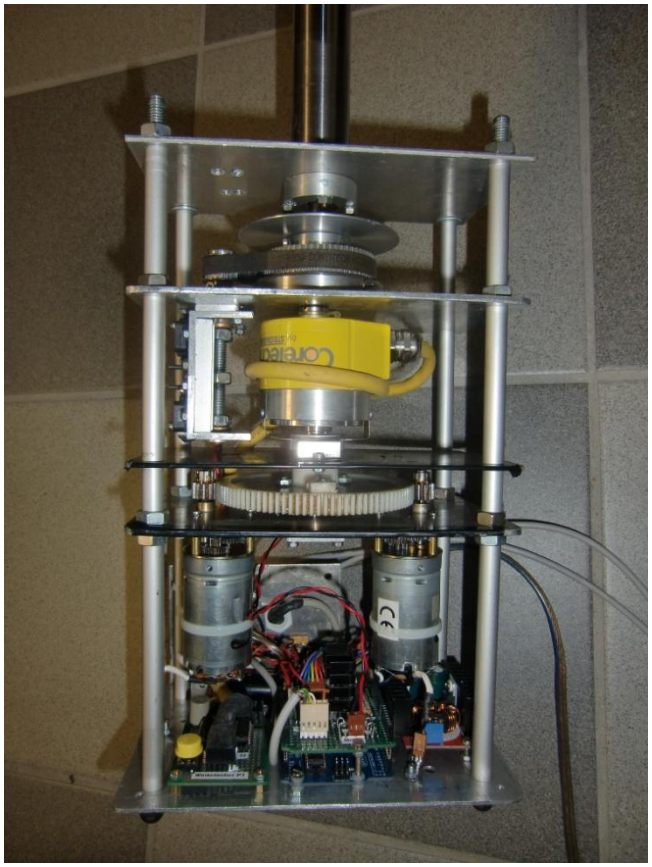
Peiler 1

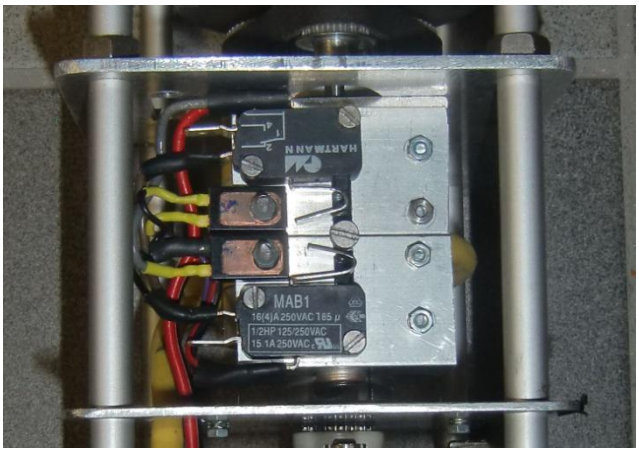
Beim Peiler 1 sind 3 Platinen auf der Bodenplatte montiert:

1. Arduino Prozessor mit dem Huckepack-Shield. Das letztere ist identisch zum Peiler 3 Shield mit dem gleichen Softwarestand und läßt sich eins zu eins austauschen. Der Stecker für die Schutzschaltung wurde mit zwei Brücken versehen, da im Peiler 1 eine mechanische Schutzschaltung existiert, die zuverlässig arbeitet.
2. Die Platine für die Winkelgeber-Schnittstellenanpassung
3. Der Spannungswandler wie im Peiler 3, diesmal bis jetzt ohne Spannungsüberwachung.









Für den Peiler 1 hatte ich eine mechanische Endlagenabschaltung konstruiert, die bisher sehr zuverlässig gearbeitet hat.

Auf einer Spindel wird eine Mutter rauf und runter bewegt, die die Mikroschalter aktiviert.

Software für den Arduino

Jan hat das Workaround abstellen können, da er jetzt die Ursache der zeitweiligen falschen Richtungen beseitigen konnte. Hier ist der korrigierte Code. Das Hunter Programm hat er ebenfalls angepaßt. Damit sollten jetzt die 3 Peiler (P1, P3 und P4) mit der gleichen Software funktionieren.

Arduino C Programm

©Jan

```
const int pinMotorEnable              = 7;
const int pinMotorClockwise           = 5;
const int pinMotorCounterClockwise    = 6;
const int pinSignalStrength           = A0;
const int pinDirectionEnable          = 12;
const int pinDirectionClock           = 10; // (has to be between 8 and 13 because it uses direct access to
PINB and PORTB - see https://is.gd/Kf85yW)
const int pinDirectionData            = 11; // (has to be between 8 and 13)

const byte charFastLeft               =  60; // <
const byte charSlowLeft               = 123; // {
const byte charStepLeft               =  91; // [
const byte charStop                   = 124; // |
const byte charStepRight              =  93; // ]
const byte charSlowRight              = 125; // }
const byte charFastRight              =  62; // >

const int millisecondsForMotorStep    = 40;
const int slowSpeedPWMdutyCycle       = 100;
const int bitsInGrayCode              = 22;


int directionClockToLow  = (1 << (pinDirectionClock-8)) ^ B11111111;
int directionClockToHigh =  1 << (pinDirectionClock-8);
int isDirectionDataHigh  =  1 << (pinDirectionData -8);


unsigned long stopMotorAtMillis;
int signalStrength;
bool currentDirection[bitsInGrayCode];


void setup() {
  Serial.begin(57600);

  pinMode(pinMotorEnable, OUTPUT);
  pinMode(pinMotorClockwise, OUTPUT);
  pinMode(pinMotorCounterClockwise, OUTPUT);

  pinMode(pinDirectionEnable, OUTPUT);
  pinMode(pinDirectionClock, OUTPUT);

  digitalWrite(pinDirectionEnable, LOW);
}

void loop() {
  motorControl();
  readSignalStrength();
  readDirection();
  sendDatagram();
```

```
}

void motorControl() {
  bool stopMotorCommandReceived = 0;
  if(Serial.available() > 0) {
    int inByte = Serial.read();
    if(inByte == charFastLeft || inByte == charStepLeft || inByte == charSlowLeft) {
      digitalWrite(pinMotorClockwise, LOW);
      analogWrite(pinMotorCounterClockwise, inByte == charFastLeft ? 255 : slowSpeedPWMdutyCycle);
      digitalWrite(pinMotorEnable, HIGH);
    }
    if(inByte == charFastRight || inByte == charStepRight || inByte == charSlowRight) {
      analogWrite(pinMotorClockwise, inByte == charFastRight ? 255 : slowSpeedPWMdutyCycle);
      digitalWrite(pinMotorCounterClockwise, LOW);
      digitalWrite(pinMotorEnable, HIGH);
    }
    if(inByte == charStop)
      stopMotorCommandReceived = 1;
    if(inByte == charStepLeft || inByte == charStepRight) {
      stopMotorAtMillis = millis() + millisecondsForMotorStep;
    }
  }
  if(stopMotorCommandReceived || stopMotorAtMillis > 0 && stopMotorAtMillis < millis()) {
    digitalWrite(pinMotorClockwise, LOW);
    digitalWrite(pinMotorCounterClockwise, LOW);
    digitalWrite(pinMotorEnable, LOW);
    stopMotorAtMillis = 0;
  }
}

void readSignalStrength() {
  signalStrength = analogRead(pinSignalStrength);
}

void readDirection() {
  /* inspired by http://forum.arduino.cc/index.php?topic=47045.msg339531#msg339531 (2016-10-05) */
  while(!digitalRead(pinDirectionData))
    delayMicroseconds(1);
  for(int i=0; i<bitsInGrayCode; ++i) {
    PORTB &= directionClockToLow;
    currentDirection[i]= PINB & isDirectionDataHigh ? 1 : 0;
    PORTB |= directionClockToHigh;
  }
}

void sendDatagram() {
  for(int i=0; i<bitsInGrayCode; ++i)
    Serial.print(currentDirection[i]);
  Serial.print(";");
  Serial.println(signalStrength);
}
```