# DK5OA Funkpeiler

# für

# Mobilfuchsjagden

## Basierend auf der Technik

## von DF3AL

## Elektronik von DK5BD

## Software von SWL Jan

## 13.01.2020

# Inhalt

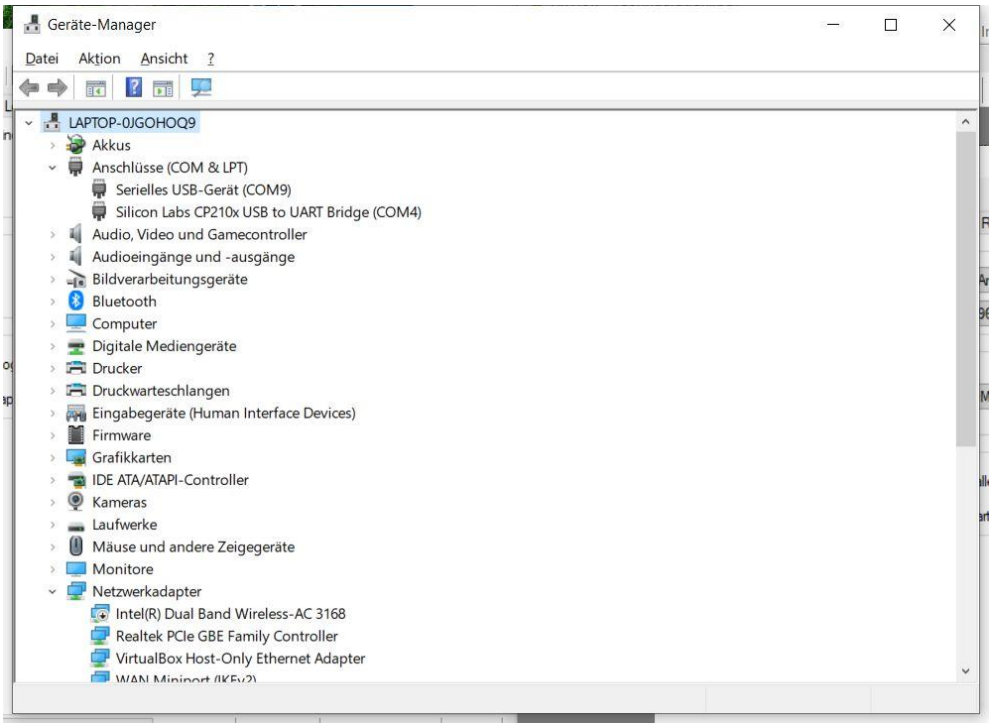
# 1 Bedienungsanleitung

## 1.1 PC-Vorbereitung

### 1.1.1 Arduino Treiber installieren

Für den Arduino wird ein virtueller COM-Port benötigt. Am besten man installiert die Arduino Entwicklungsumgebung IDE, die den Treiber beinhaltet: Hier ist der Link:
https://www.arduino.cc/en/Main/Donate

Nach der Installation muß der COM-Port im Geräte Manager gelistet sein.




### 1.1.2 GPS Maus

Wenn eine GPS Maus benutzt wird, ist dieser COM-Port im Geräte Manager ebenfalls gelistet. (siehe oben)

### 1.1.3 Android Smartphone

Sofern ein Android Smartphone zur Verfügung steht, kann dieses auch die GPS Position an Hunter übermitteln. Voraussetzung ist, daß dieses als WLAN Router (Tethering) geschaltet ist und die App „Hunter.apk“ installiert wurde.

### 1.1.4 Hunter installieren

Das aktuellste von Jan erstellte Hunter Programm inklusive aller Dateien und Unterverzeichnisse muß an einer geeigneten Stelle installiert werden.

### 1.1.5 Unterverzeichnis „Mapcache“ erstellen

In den Einladungen der Veranstalter werden die Nummern der Topographischen Karten genannt. Diese können bereits vor der eigentlichen Fuchsjagd heruntergeladen werden. Voraussetzung ist, daß das Unterverzeichnis „Mapcache“ vorhanden ist. Dann werden Karten in verschiedenen Maßstäben heruntergeladen und man erhält die Möglichkeit, bis in Google Earth hineinzuzoomen.

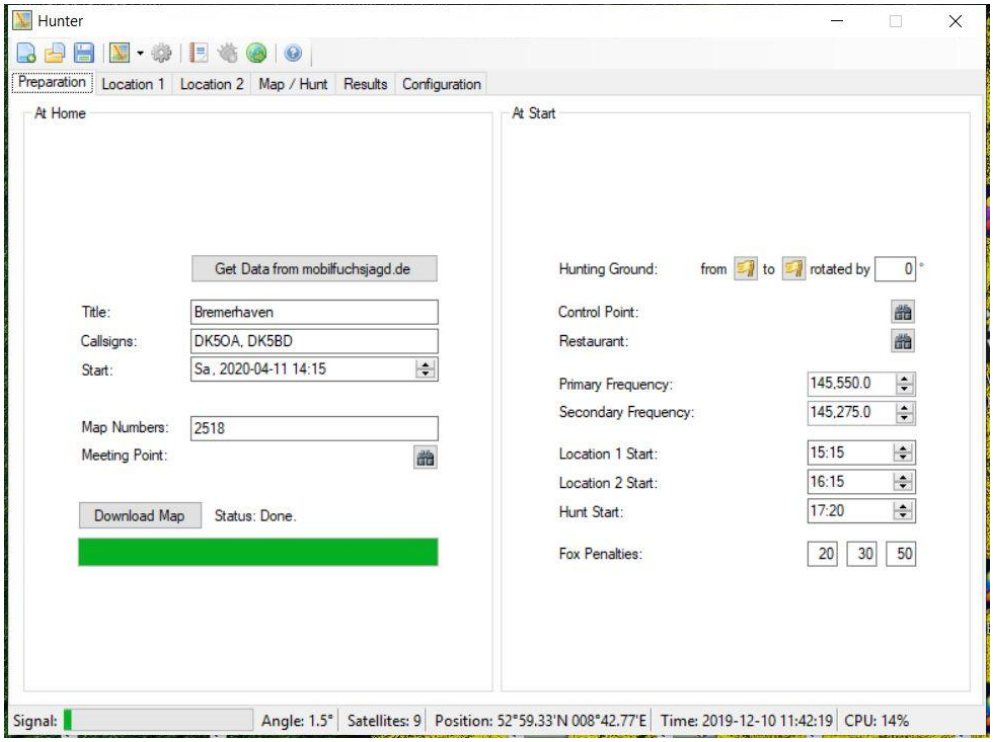
## 1.2 Hunter Programm

### 1.2.1 Hunter starten

Hunter sollte als Administrator gestartet werden. Nur dann erhält man die Möglichkeit, aus dem Programm auf TOP50 zugreifen zu können, sofern dieses installiert ist.

### 1.2.2 Hunter vorbereiten

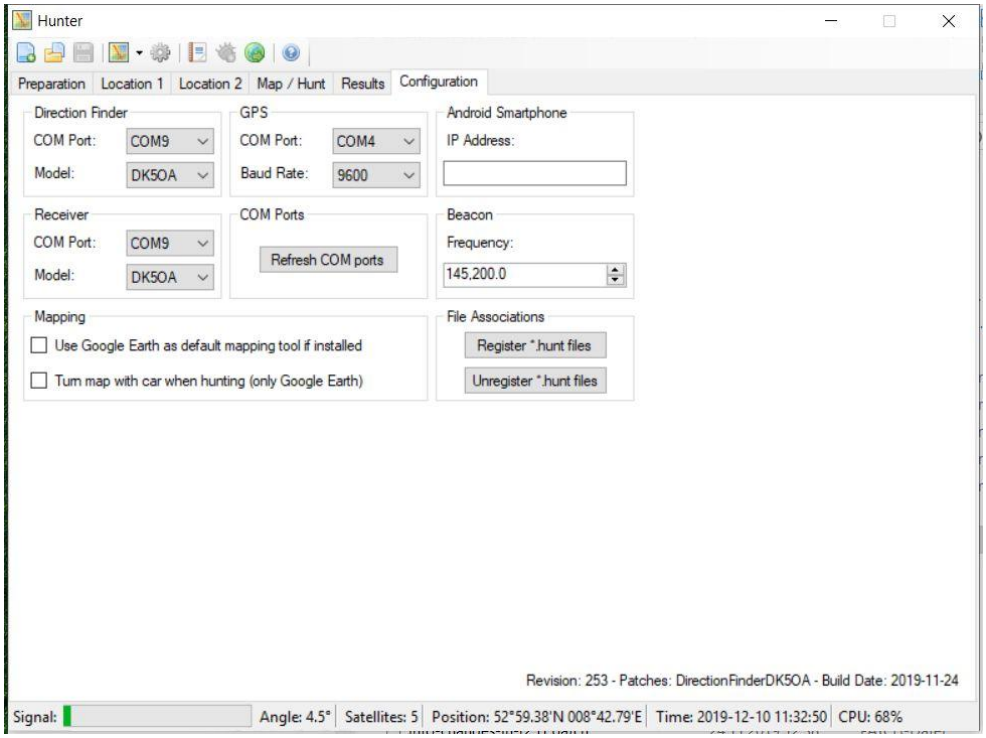
Hunter startet mit der Seite Preparation




Hier sind die spezifischen Angaben der Wettbewerbe zu machen

- Title: Veranstaltung
- Rufzeichen (über Kommas getrennt)
- Zeit
- Frequenzen
- Diverse Zeiten
- Herunterladen der Karten

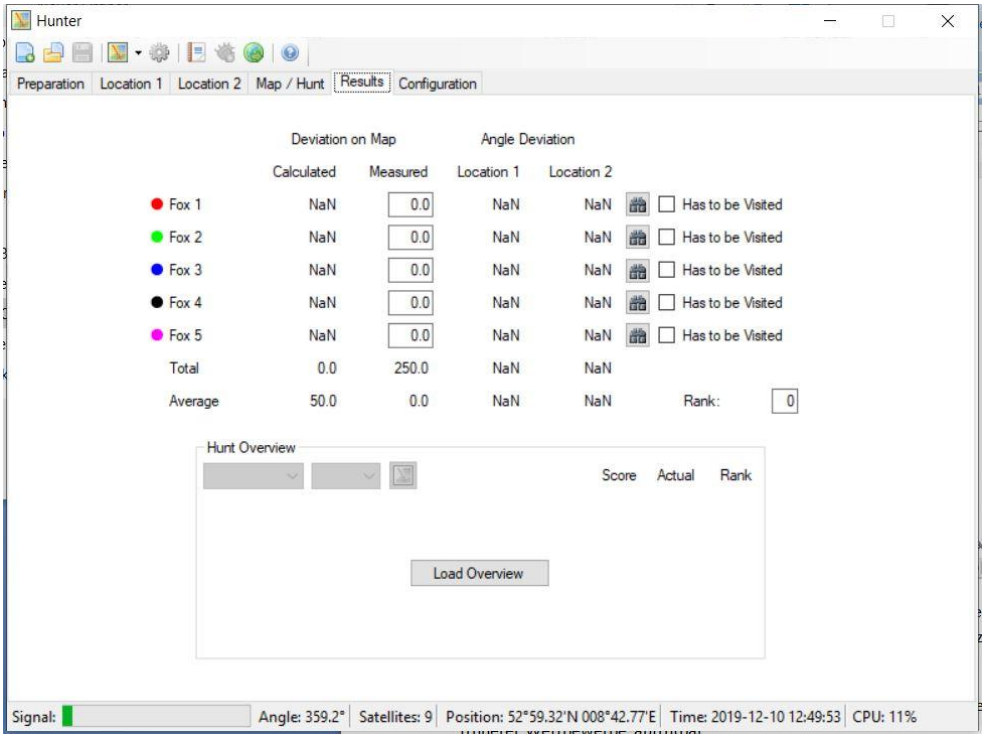
### 1.2.3 Hunter konfigurieren




Hier müssen die COM Ports für den Peiler und GPS eingetragen werden. Gegebenenfalls die IP-Nummer vom Android Smartphone, falls dies für GPS und/oder die Fernsteuerung benutzt werden soll.

### 1.2.4 Übernahme der vorbereiteten Einstellungen

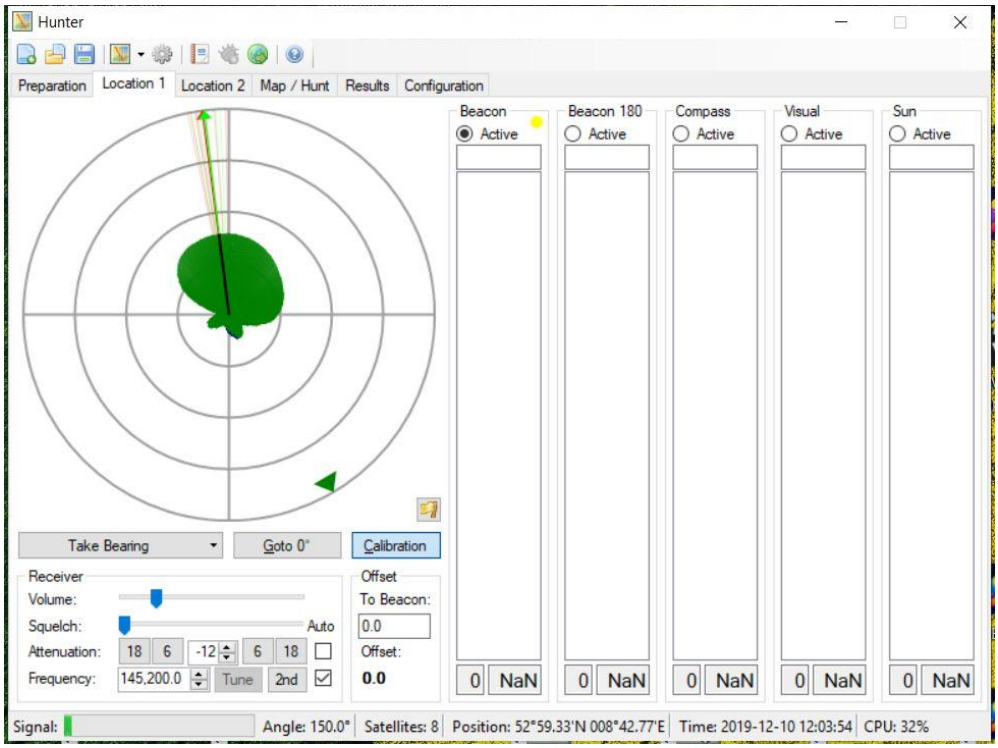
Im Fenster Results können unter Hunt Overview die zuhause eingestellen Daten geladen werden. Des weiteren sind hier alle Daten früherer Wettbewerbe aufrufbar.



## 1.3 Peilungen

### 1.3.1 Wo ist Norden?

Die Ausrichtung der Antenne gegenüber Nord ist die wichtigste Aktion. Da gibt es mehrere Möglichkeiten.

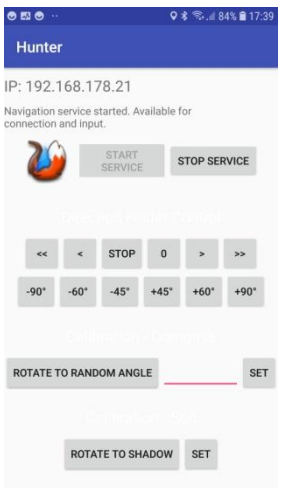



#### *1.3.1.1 Ausmessen der Antenne*

Die Antenne auf 0° Fahrzeug bezogen fahren und dann die Richtung mit dem Kompass ausmessen. Den so ermittelten Wert als Offset eintragen. Die Genauigkeit kann erhöht werden, wenn dies von vorn und von hinten erfolgt.

#### *1.3.1.2 Peilen einer Peilbake*

Ca 100m vor oder hinter dem Fahrzeug wird ein schwacher Sender (Bake bzw. Beacon) gestellt, der dann gepeilt werden muß. Im Fenster Location1,2 wird „Calibration“ aufgerufen und die Spalte „Beacon“ aktiviert. Typisch machen wir 20 Peilungen, die die Straßenrichtung ergeben sollen. Jetzt wird das Fenster „Map/Hunt“ aufgerufen. Auf der Karte hat sich ein gelber Strich ergeben, den man jetzt parallel zur Straße legen muß. Das geschieht durch Markierung der Bakenposition mit der Maus. „Rechte Maustaste“, „Calibrierung“, „The beacon ist here“. Damit ist das Offset eingestellt.




### 1.3.2 Erweiterte Kompassausrichtung

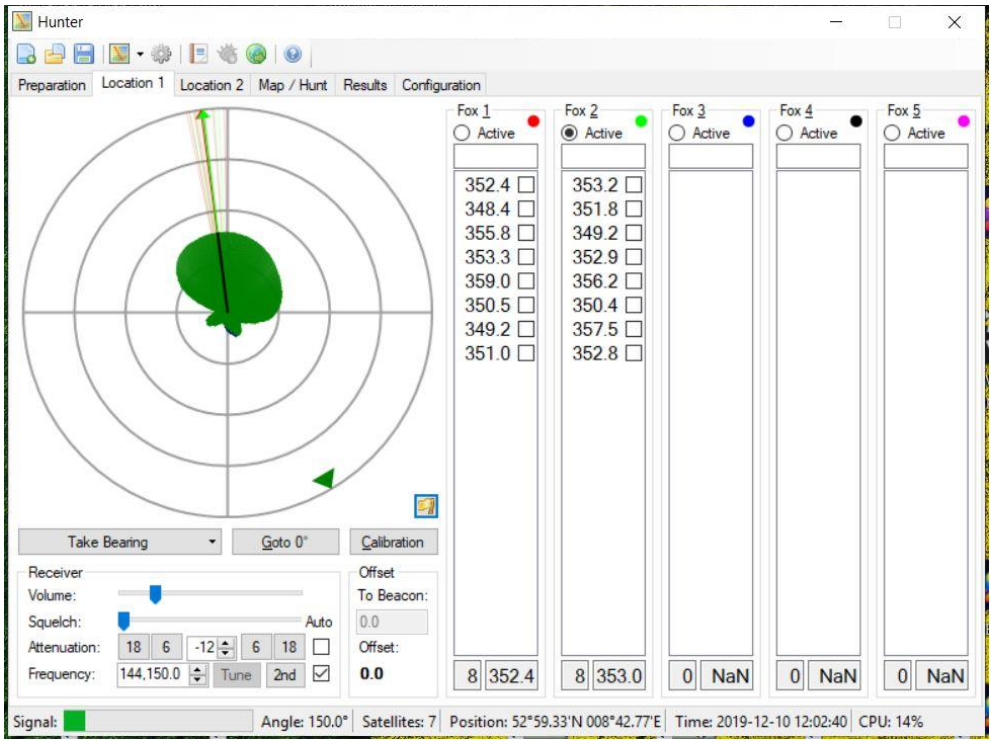
Ist die Android Fernbedienung vorhanden, können Kompasseinstellungen (die Peilung der Antenne) aus vielen Richtungen gemacht werden. Dann ist im „Calibration“ Fenster die Spalte „Compass“ zu aktivieren. Die jeweiligen Richtungen trägt man ins Smartphone ein, das die unmittelbar ins Hunterprogramm sendet. Da Hunter die Fahrzeug bezogene Richtung kennt, werden in der Spalte sofort die Offsetwerte berechnet und angezeigt. Typisch ist hier ein Durchschnitt von ca. 8 bis 10 Messungen, die einen sehr genauen Mittelwert und damit Offset bilden.

Hunter berücksichtigt für den jeweiligen Standort und Zeit die magnetische Deviation, die sich laufend ändert, jedoch durch einen Algorithmus im Hunter aktuell berechnet wird.

### *1.3.2.1 Welches Offset gilt nun*

Basis für das Offset ist die jeweils aktivierte „Calibration“ Spalte. Ergeben sich Differenzen zwischen Straßen- und Kompasspeilung, so kann ein Mittelwert manuell eingetragen werden.

## 1.3.3 Peilungen von Standort 1 und 2




Die Frequenz wurde bereits unter „Preparation“ eingestellt. Jetzt können die Peilungen erfolgen. Wenn das Signal zu stark ist, ist die Dämpfung (Attenuation) zu aktivieren.

Das Empfangssignal kann unter „Attenuation“ stufenweise (+-6dB bzw. +-18dB) oder auch kontinuierlich gedämpft werden. Die Dämpfung bewirkt eine Reduzierung der Betriebsspannung der Eingangsstufe (5-0V).

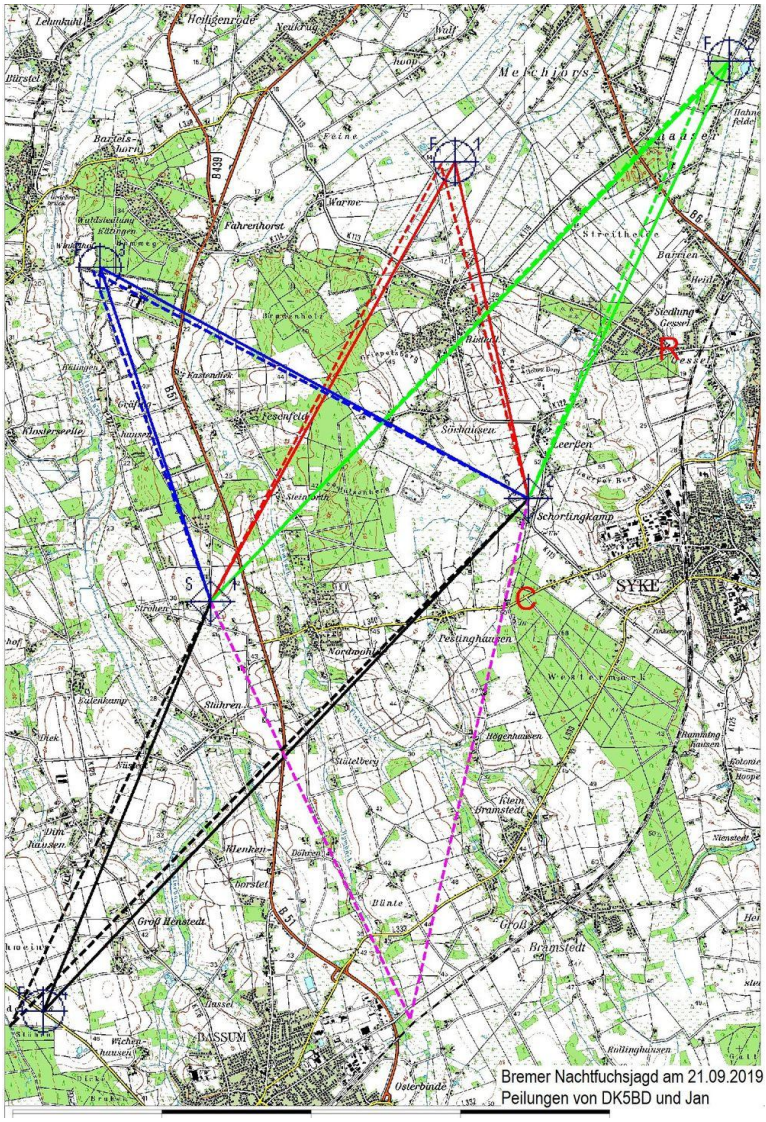
Nachdem der Pegel richtig eingestellt ist, kann die Mehrfachpeilung unter „Take Bearing“ aufgerufen werden. Typisch wären hier 10 bis 20 Peilungen, die dann in der jeweiligen Fuchsspalte gelistet werden. Ausreißer können durch Anticken inaktiv geschaltet werden. Das System macht immer 2 Peilungen auf einmal: Rechts- und Linkspeilung, da diese sich systembedingt geringfügig unterscheiden und dadurch kompensieren.

Sind die Füchse gepeilt, können unter „Map/Hunt“ die Richtungen eingesehen werden. Die 5 Peilstrahlen werden in unterschiedlicher Stärke dargestellt. Die Stärke ist ein Maß für die Feldstärke. Jetzt kann ein geeigneter Standort für Location2 gewählt werden. Dazu sind die zuhause heruntergeladenen Karten sehr hilfreich. Man kann Windräder und sonstige Hindernisse meiden.

An Standort 2 wiederholen sich „Calibration“ und Peilungen wie oben beschrieben.

### 1.3.4 Peilergebnis

Nachdem alle Peilungen vollendet sind, zeigt Map/Hunt die vermeintlichen Positionen der Füchse.




Die gestrichelten Linien stehen erst zur Verfügung, nachdem die tatsächlichen Standorte der Füchse nach Beendigung des Wettbewerbs eingetragen worden sind.

Wenn im zweiten Durchgang die Füchse aufgesucht werden, können die Positionen übernommen werden und man erhält bereits zu diesem Zeitpunkt einen Hinweis auf die erreichte Peilgenauigkeit.

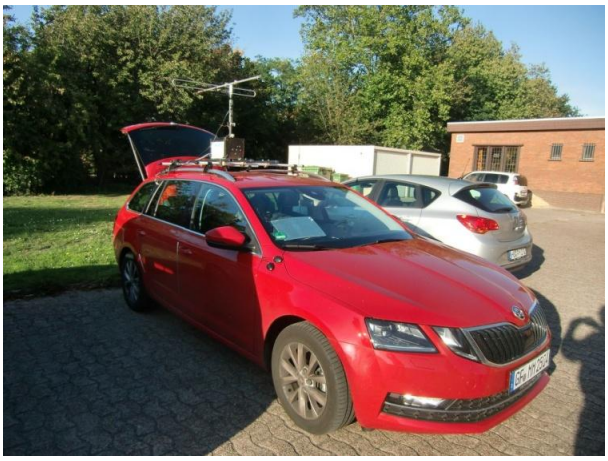
# 2 Umbau Peiler DK5OA/DF3AL für Hunter-Kopplung

Der DK5OA-Peiler ist ehemalig von DF3AL auf Conrad C-Control Technologie entwickelt worden. Er besteht aus zwei Einheiten

- Motoreinheit mit der für Rechts- und Linkslauf erforderlichen Elektronik
- Steuereinheit mit C-Control Prozessor, Empfänger, LCD Anzeige und Tastenfeld

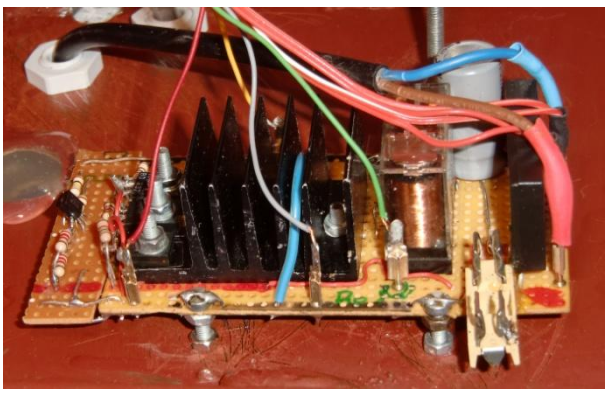
Somit steht ein selbständiges Peilsystem zur Verfügung, das in der Vergangenheit auch sehr erfolgreich war.

Molly, DK5OA hatte den Wunsch geäußert, den vorhandenen Peiler mit dem Hunter-Programm von Jan zu koppeln. Das Hunter-Programm hatte er schon vorher genutzt unter manueller Eingabe der gepeilten Richtungen.



Motoreinheit

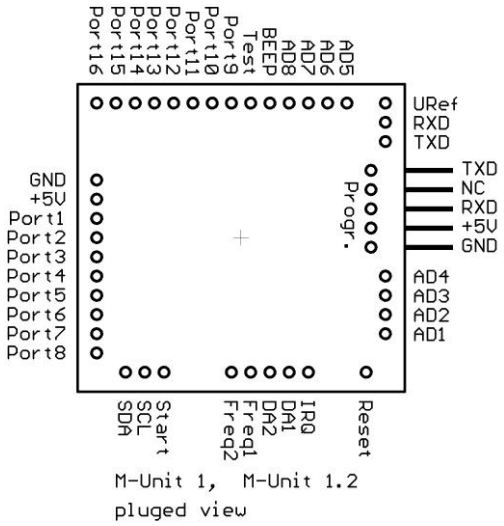
Motorsteuerung

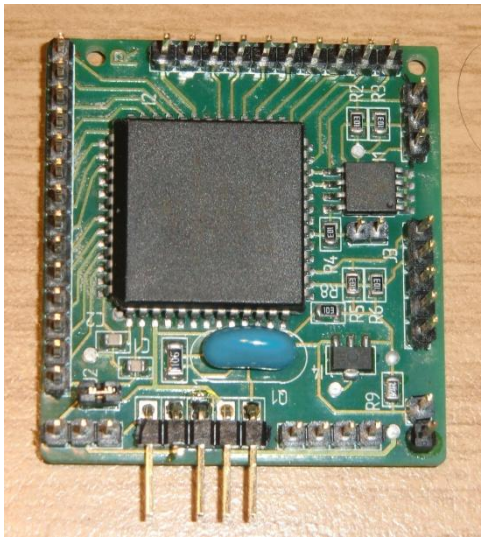


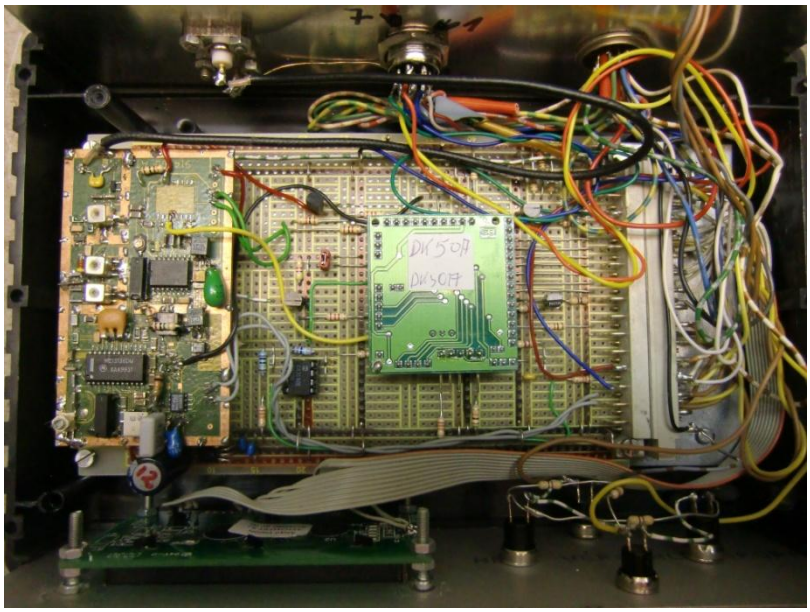
## 2.1 Realisierung

Der C-Control Prozessor ist auf einer Grundplatine gesteckt. Der Plan war nun, diesen durch einen Arduino zu ersetzen, der dann über ein USB-Kabel mit einem Laptop verbunden werden kann. Ziel war es nun, den Peiler im Original zu belassen, sodass durch einfaches Umstecken der alte Zustand wieder hergestellt werden kann.

Conrad C-Control:





Der Arduino muß jetzt folgende Schnittstellen bedienen:

- Motor Rechts/Links
- Frequenz des Empfängers (PLL über E²C)
- Lautstärke (Down/Up)
- Dämpfung

Also mußte eine Adapterplatine hergestellt werden, die die vorhandenen Anschlüsse auf die Arduino Ports leitet.

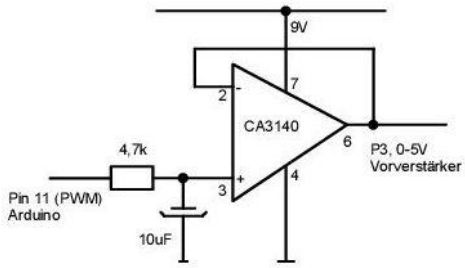
Hier mit Conrad C-Control

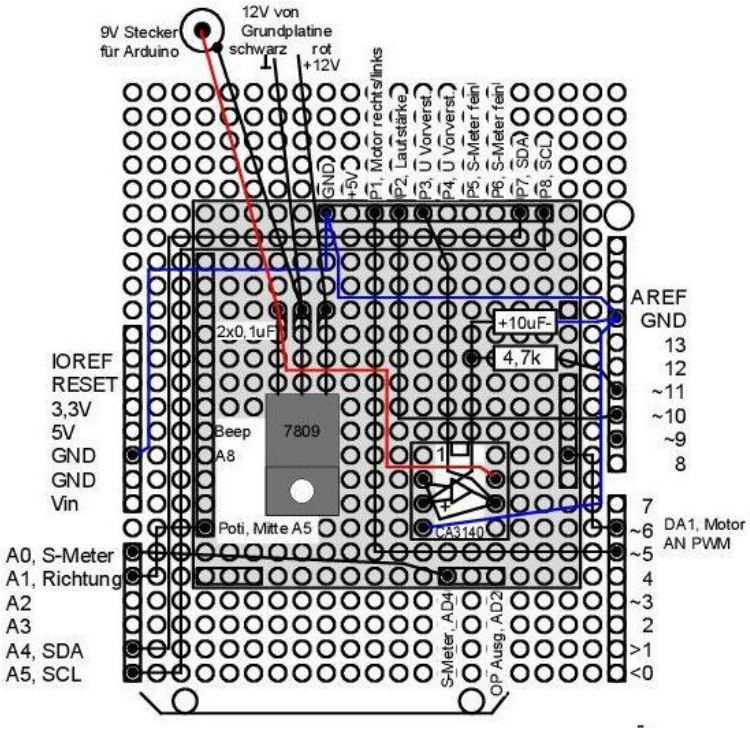
## 2.2 Adapterplatine (Shield für Arduino)

Der Adapter sollte wegen der Austauschbarkeit pinkompatibel sein. Ziel war es, die vorhandene Schaltung auf der Basisplatine und die des Empfängers nicht zu verändern.

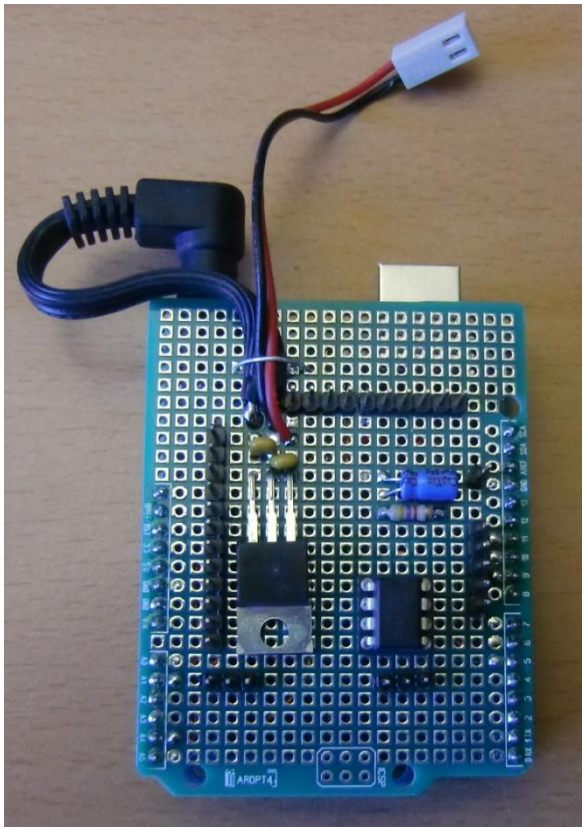
Somit mußte die Down/Up Lautstärkeregelung verbleiben.

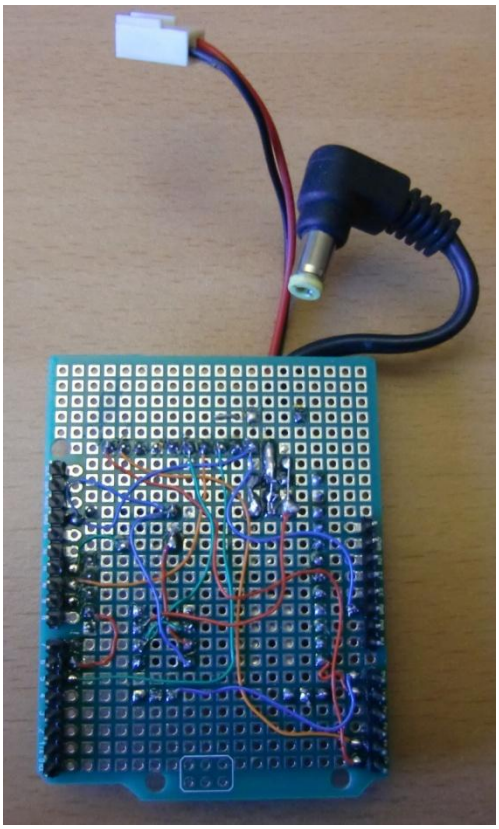
Allerdings haben wir hier mit Hilfe eines Operationsverstärker und der PWM-Steuerung des Arduinos jetzt eine stufenlose Dämpfungseinstellmöglichkeit geschaffen.




Auf den unteren Bildern sind die Anschlüsse für die 12V Versorgung, des 9V Längsreglers und der Stecker für die Arduino-Versorgungsspannung zu sehen. Ebenso der CA3140 Operationsverstärker, der die Versorgungsspannung der RX-Eingangsstufe liefert. Das PWM Signal wird über ein R-C-Glied geglättet.





Die Shieldplatinen in Lötpunkt-Ausführung stehen speziell für den Arduino zur Verfügung. Sie können mit den dazugehörigen Steckerleisten bestückt werden.

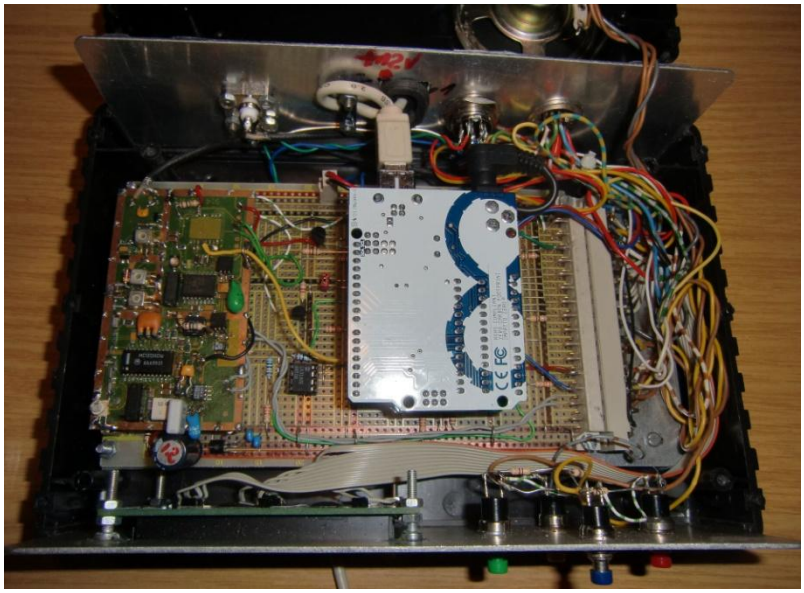
Die Verdrahtung des Shields erfolgte dann mit dünnem Schaltdraht.

Das PWM Signal wird über ein R-C-Glied geglättet und dann mit dem Operationsverstärker impedanzgewandelt, damit der erforderliche Strom zur Verfügung steht.


Das Arduino Prozessorboard:

## 2.3 Die DK5OA Peiler-Steuerung

Hier mit Arduino



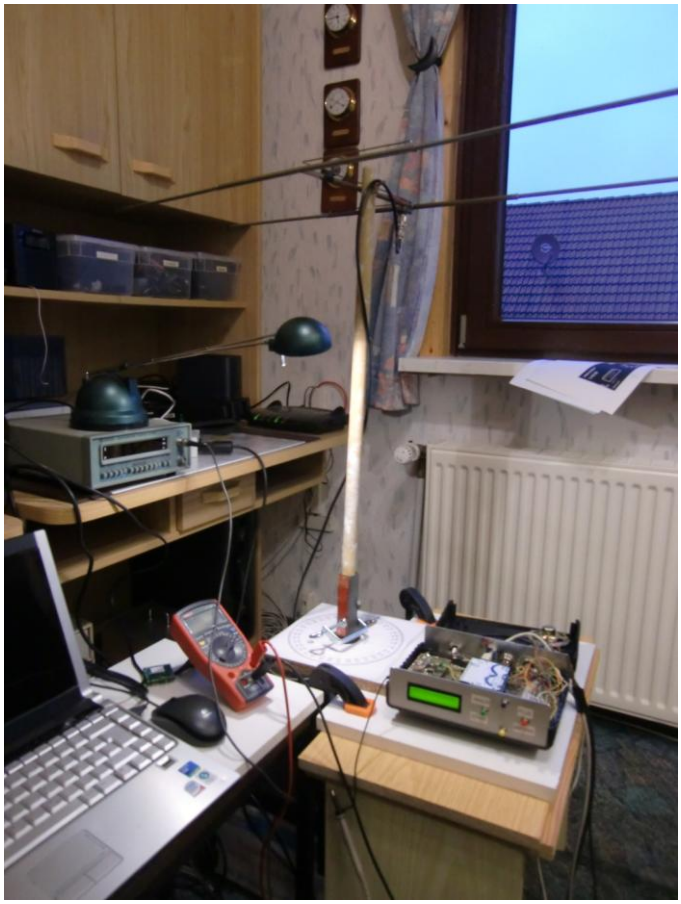
Die Adapterplatine ist hier nicht sichtbar, da sie vom Arduino verdeckt wird.

Die im Original vorgesehene Feinpeilung konnte hier entfallen, da der Arduino über A/D Wandler verfügt, die eine höhere Auflösung haben.

## 2.4 Kalibrierung

Bei der Kalibrierung, das heißt die Eichung der Richtungsskala, haben wir bemerkt, dass bei Anschluß verschiedener Rechner es zu unterschiedlichen Richtungswerten kam. Da der Arduino seine 5V Betriebsspannung aus dem USB-Port erhielt, kam es bei geringfügigen Spannungsdifferenzen zu unterschiedlichen Ergebnissen. Die 5V speisen das 10 Gang-Poti und gehen damit direkt auf das Ergebnis ein.

Wir haben dann dem Arduino eine eigene stabile Spannungsversorgung spendiert (siehe 7809 auf dem Shield)



## 2.5 DK5OA-Peiler Testaufbau

Zunächst mußte eine Testumgebung geschaffen werden. Dazu mußte ein Zeiger am Mast befestigt werden, der die Richtung anzeigt.

Eine 360° Skala wurde auf ein Brettchen geklebt.

Danach mußte die Software getestet werden. Hier gibt es 2 Baustellen:

- Anpassung der Arduino Software
- Anpassung der Hunter Software

Für den Betrieb ist ein Arduino Treiber zu installieren, der einen virtuellen COM Port erzeugt. Dieser muß bei der Hunter-Configuration eingetragen werden. (siehe 1.1.1)

# 3 Arduino Software

Die Software wurde von Jan Kemper, Sohn von DK5BD, geschrieben:

```
#include <Wire.h>

const int directionFinderModel     = 2; // 0 for DJ4TA, 1 for Eimer, 2 for DF3AL

const int pinMotorEnable          = directionFinderModel == 2 ? 6 : 7;
const int pinMotorClockwise       = 5;
const int pinMotorCounterClockwise = directionFinderModel == 2 ? 7 : 6;

const int pinSignalStrength       = A0;
const int pinDirectionPoti        = A1;

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 int pinVolume               = 10;
const int pinAmplifier            = 11;

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 byte charI2C                =  73; // I
const byte charIncreaseVolume     =  43; // +
const byte charDecreaseVolume     =  45; // -
const byte charAmplifier          =  65; // A

const int millisecondsForMotorStep = 40;
const int slowSpeedPWMdutyCycle    = 100;
const int fastSpeedPWMdutyCycle    = directionFinderModel == 0 ? 255 : 200;
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;
int directionPoti;
bool currentDirection[bitsInGrayCode];

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

  pinMode(pinMotorEnable, OUTPUT);
```

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

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

  if(directionFinderModel == 2) {
    pinMode(pinAmplifier, OUTPUT);
  }

  digitalWrite(pinDirectionEnable, LOW);
}

void loop() {
  motorAndReceiverControl();
  readSignalStrength();
  if(directionFinderModel == 0)
    readDirection();
  else
    readDirectionPoti();
  sendDatagram();
}

void motorAndReceiverControl() {
  bool stopMotorCommandReceived = 0;
  if(Serial.available() > 0) {
    int inByte = Serial.read();
    if(inByte == charI2C) {
      while(Serial.available() < 2)
        delay(1);
      int address   = Serial.read();
      int byteCount = Serial.read();
      while(Serial.available() < byteCount)
        delay(1);

      Wire.beginTransmission(address);
      for(int iByte=0; iByte<byteCount; ++iByte)
        Wire.write(byte(Serial.read()));
      Wire.endTransmission();
    }
    if(inByte == charAmplifier) {
      while(Serial.available() < 1)
        delay(1);
      analogWrite(pinAmplifier, Serial.read());
    }
    if(inByte == charIncreaseVolume) {
      pinMode(pinVolume, OUTPUT);
      digitalWrite(pinVolume, HIGH);
      pinMode(pinVolume, INPUT);
    }
    if(inByte == charDecreaseVolume) {
      pinMode(pinVolume, OUTPUT);
      digitalWrite(pinVolume, LOW);
      pinMode(pinVolume, INPUT);
    }
```

```
   if(inByte == charFastLeft || inByte == charStepLeft || inByte == charSlowLeft) {
    digitalWrite(pinMotorClockwise, LOW);
    digitalWrite(directionFinderModel == 2 ? pinMotorCounterClockwise : pinMotorEnable, HIGH);
    analogWrite(directionFinderModel == 2 ? pinMotorEnable : pinMotorCounterClockwise, inByte ==
charFastLeft ? fastSpeedPWMdutyCycle : slowSpeedPWMdutyCycle);
   }
   if(inByte == charFastRight || inByte == charStepRight || inByte == charSlowRight) {
    digitalWrite(directionFinderModel == 2 ? pinMotorClockwise : pinMotorEnable, HIGH);
    digitalWrite(pinMotorCounterClockwise, LOW);
    analogWrite(directionFinderModel == 2 ? pinMotorEnable : pinMotorClockwise, inByte == charFastRight ?
fastSpeedPWMdutyCycle : slowSpeedPWMdutyCycle);
   }
   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 readDirectionPoti() {
 directionPoti = analogRead(pinDirectionPoti);
}

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() {
 if(directionFinderModel == 0) {
  for(int i=0; i<bitsInGrayCode; ++i)
   Serial.print(currentDirection[i]);
 } else {
  Serial.print(directionPoti);
 }
 Serial.print(";");
 Serial.println(signalStrength);
}
```