„GeoRex“ – Ein Mess-System für geo-referenzierte Radioaktivitätsmessungen

Bernd Laquai, 28.3.13

Spätestens mit Fukushima ist klar geworden, wie wertvoll es sein kann, eine größere Region auf ihre Strahlungsbelastung hin relativ schnell messtechnisch zu erfassen. Unmittelbar nach der Katastrophe in dem dortigen Reaktor waren keine zuverlässigen flächendeckenden Messdaten vorhanden und es wurde eine Evakuierungszone mit relativ primitiven Annahmen anhand eines Kreisgebiets um den Reaktor definiert. Später stellte sich heraus, dass die maximale und für den Schutz der Bevölkerung relevante, radioaktive Kontamination eher außerhalb dieser Evakuierungszone lag. Es ist auch klar, dass im Falle von kerntechnischen Unfällen und Katastrophen oft biologisch hochwirksame Alphastrahler und sehr kurzlebige starke Betastrahler freigesetzt werden können. Ein Beispiel ist das Jod-131 mit 8 Tagen Halbwertszeit, das für die Induktion von Schilddrüsenkrebs in Tschernobyl berüchtigt ist. In so einem Fall wäre ist Exposition sehr stark aber von kurzer Dauer und deswegen problematisch, weil die Schilddrüse das radioaktive Jod mit dem normalen Jod verwechselt und ins Drüsengewebe einlagert. Man hat daher nicht lange Zeit, die Gebiete mit einem solchen Fallout zu identifizieren um die Bevölkerung rechtzeitig warnen zu können. Die offiziellen Zivilschutzbehörden sind damit meist hilflos überfordert, wie man selbst in einem Hochtechnologieland wie Japan sehen konnte.

Nachdem sich diese Erkenntnis auch in Japan verbreitet hatte, die Regierung aber nicht in der Lage war flächendeckende Messungen durchzuführen, bildete sich eine Gruppe von Privatpersonen (www.safecast.org) aus dem In- und Ausland, die das Problem versuchte dezentral und verteilt anzugehen. Freiwillige montierten Geigerzähler und GPS Empfänger auf Fahrzeuge und versuchten die Strahlung während der Fahrt zu messen und gleichzeitig die GPS Daten der momentanen Fahrzeug-Position aufzuzeichnen. Auf diese Weise wurden nicht wie sonst bei Flächenmessungen ein regelmäßiges Gitterraster erzeugt, sondern die Ortsabhängigkeit auf das Straßennetz abgebildet. Da das Straßennetz in Japan sehr dicht ist, konnte mit Interpolation zwischen den Straßen relativ schnell eine Flächenfunktion der Strahlung in vielen Regionen erstellt werden.

Der erste Ansatz dieser Gruppe für ein automatisierte Messgerät bestand aus einem Geigerzähler und einem Arduino Mikrocontroller Board auf das ein GPS Modul mit Speicherkarte aufgesteckt war. Mit einem solchen System erreichte die Gruppe in kurzer Zeit eine umfangreiche Kartendarstellung. Allerdings stellte sich schnell heraus das es ein nicht unerhebliches Problem war, die Messdaten vergleichbar zu machen, wenn die Geigerzähler unterschiedlicher Bauart waren. Deswegen entschloss man sich, die Entwicklung eines standardisierten Geräts an einen kommerziellen Hersteller zu vergeben. Dabei war ein weiteres Ziel für die Verbesserung einen großen schnellzählenden Detektor einzusetzen, der mehr Messpunkte pro Kilometer Fahrstrecke bei normaler Fahrgeschwindigkeit erreicht. Mit dem „Open Source Gedanken“ konzentrierte man sich dann mehr auf die Nachverarbeitung der Daten und die Kartenerstellung. Allerdings muss sich auch diesbezüglich noch Herausstellen, wie gemeinnützig die Umsetzung längerfristig wirklich ist. Laut Ankündigung der Gruppe wird ein solches kommerzielles Gerät in der Größenordung von 800 US-Dollar kosten und es ist anzunehmen, das die Entwicklung und die Schaltungsdetails nicht mehr offengelegt werden. Das ist schade, da dieses Konzept absolut einleuchtend ist und nicht nur für Katastrophen hilfreich einsetzbar wäre.

Aus diesem Grund wird hier das ursprüngliche Arduino-basierte Konzept noch einmal aufgegriffen und es wird in diesem Dokument dargestellt, wie man mit sehr einfachen Mitten und bei Kosten unter 100 Euro ein sehr effizientes Messgerät für geo-referenzierte Strahlungsmessungen selbst bauen kann. Mit Hilfe eines einfachen Kaliumchlorid-Prüfstrahlers aus der Apotheke kann man das Gerät durchaus auch grob kalibrieren.

Das System Konzept

Das GeoRex System-Konzept baut auf einem Arduino Uno (oder vergleichbarem Arduino Board) auf. Die Mikrocontroller-Platine (Atmel Atmega328 8-Bit Controller) ist mit einer USB Schnittstelle für die Programmierung und einem seriellen Monitorprogramm ausgestattet und hat darüber hinaus etliche frei programmierbare digitale und analoge Pins, die auf Steckerleisten zugänglich sind. Auf diese Steckerleisten werden Anwendungsplatinen gesteckt, sogenannte Shields, welche die Basisfunktionalität des Controllers mit zusätzlicher Hardware erweitern. Dazu gibt es eine freie Software als Entwicklungsumgebung, die extrem einfach zu bedienen ist. Zur Programmierung wird die Hochsprache C verwendet wobei der Compiler mit einer umfangreichen Bibliothek geliefert wird, so dass man sich um die internen Details des Controllers eigentlich nicht mehr kümmern muss. Auf Grund der frei verfügbaren Entwicklungsumgebung und ihrer Einfachheit ist der Arduino extrem verbreitet und es gibt unzählige Hardwareerweiterungen, Bauanleitungen und Softwareunterstützung von einer rasant wachsenden Fan-Community (siehe auch die Arduino-Webseite www.arduino.cc).

Als GPS Modul wird ein ITead Studio GPS Shield verwendet, das über Exp-Tech, http://www.exp-tech.de/Shields/ITead-Studio-GPS-shield.html bezogen werden kann. Es nutzt wiederum ein EB-365 GPS Modul von Globalsat mit einem SIRF3 GPS-Chip. Für dieses Modul gibt es eine mit einem Kabel abgesetzte, aktive GPS Antenne, die mit einem eingebauten Magneten beispielsweise auf das Autodach geklebt werden kann. Auf dem Itead Modul ist ebenfalls ein MicroSD Karten Slot angebracht, so dass man Daten loggen kann. Als Schnittstelle dient ein einfaches serielles Protokoll für das es eine Bibliotheksunterstützung gibt. Auf einer Micro SD-Karte wie sie die meisten Smartphones auch verwenden beispielsweise mit 2Gbyte Speicherkapazität kann man so für viele Tage Daten aufzeichnen.

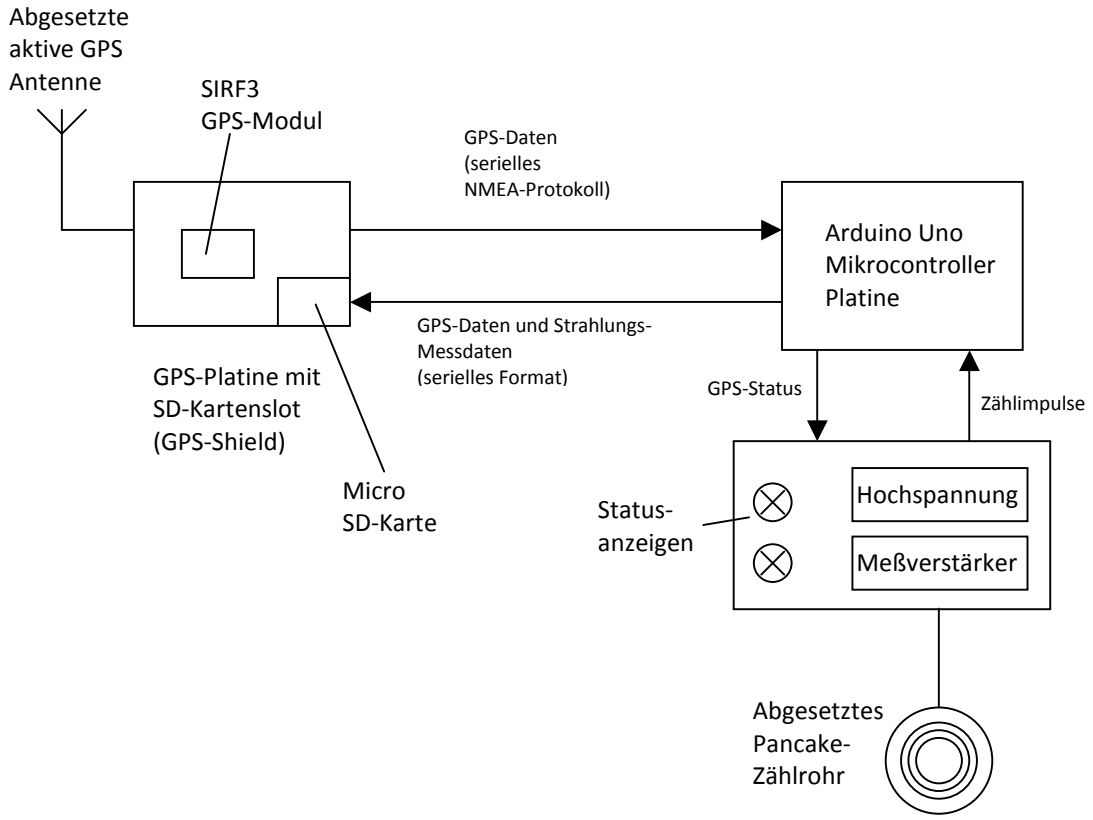

Abb. 1: Das System Konzept für GeoRex

Das GPS-Modul lässt sich ebenfalls über ein serielles Protokoll (NMEA) konfigurieren und liefert in diesem Format auch die GPS-Koordinaten und zusätzliche Zeit und Satelliteninformation an den Arduino Mikrocontroller zurück. Dabei rechnet der GPS-Chip bis zu fünf Fixes pro Sekunde (Positionsdaten). Selbst bei einem Fix pro Sekunde bekommt man noch ein recht hohe Trackpunktdichte, wenn man beispielsweise während einer Fahrradfahrt Daten loggt.

Für das Geo-Logging werden außer den Koordinaten Daten und der Zeit nur noch der Empfängerstatus benötigt, der eine Aussage darüber zulässt ob genügend Satelliten in Sicht sind. Dies wird über eine Status-LED vom Arduino an den Benutzer zur Funktionskontrolle weitergegeben.

Als Geigerzähler wird das Zählrohr-basierte „Classic Geiger“ Konzept verwendet, wobei als Zählrohr ein möglichst großes Pancake-Rohr (Beta SB-2-1 von Consensus-Group, Russland) verwendet wurde. Es zählt deutlich schneller als ein PIN-Dioden Zähler wobei bei natürlicher Umgebungsstrahlung immer noch deutlich mehr GPS Trackpunkte als Strahlungsdaten erzeugt werden. Die Ortsauflösung der Messdaten hängt ganz entscheidend von der Zählrate ab. Für eine brauchbare Statistik braucht man mindestens 100 Pulse. Wenn der Detektor für diese 100 Pulse 5 Minuten braucht, hat man mit dem Auto bei 60km/ bereits 5km zurückgelegt.

Als Information zur Gewinnung der Strahlungsdaten werden die Zählimpulse an das Arduino Board gegeben. Der Arduino nutzt die Interrupt-Steuerung um auf die unvorhersehbaren

Zählpulse zu reagieren und einen Zähler hoch zu zählen. Sobald 100 Zählimpulse erreicht sind, wird die benötigte Messzeit für die 100 Pulse aus der Systemzeit des Controllers bestimmt und zwischen zwei GPS-Datensätze auf die SD-Karte geschrieben.

Die Auswertung der Daten auf der SD-Karte und die grafische Darstellung werden nach abgeschlossener Messung am PC gemacht.

Das GPS Modul

Das GPS-Modul ist ein sogenanntes Shield für den Arduino, d.h. eine Platine, die direkt auf das Arduino Mikrocontroller-Board aufgesteckt werden kann. Das Shield stammt von ITeadStudio (iteadstudio.com Model: IM120417017) und wird in Deutschland genau wie die Arduino-Boards von der Firma Exp-Tech (exp-tech.de) vertrieben. Das Shield verwendet ein serielles UART Modul für die Kommunikation mit dem Arduino-Board und verwendet dazu das für GPS Geräte übliche NMEA-Protokoll. Außerdem bietet es eine Anschlussmöglichkeit für eine externe aktive GPS Antenne (Model: IM120717003), die man ebenfalls über Exp-Tech beziehen kann. In dem GPS Modul von Globalsat ist ein SIRF III Chip von der früheren Firma SiRF heute CSR untergebracht. Zu den SiRF Chips gibt es im Internet jede Menge Dokumentation und ein Programm namens Sirfdemo, mit dem man den Chip konfigurieren kann und sich die Performance des Moduls am PC direkt anschauen kann. Bei der Firma sparkfun.com kann man sich das NMEA Reference Manual runterladen, welches das Datenprotokoll für die serielle GPS-Schnittstelle beschreibt. Hardware-technisch verwendet das NMEA Protokoll die selbe Signalisierung wie eine RS232 Schnittstelle.

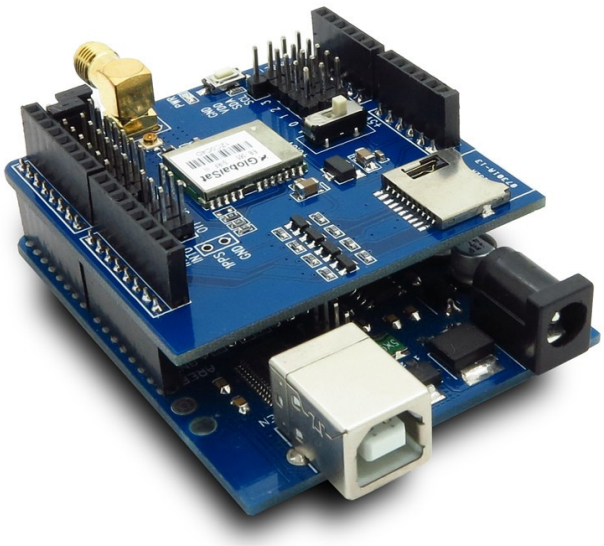

Abb. 1: Auf das Arduino Board aufgestecktes GPS Shield

Abb. 2: Aktive GPS Antenne für das GPS Shield (Rückseite mit Befestigungs-Magnet unter dem Label)

Das GPS-Shield ist in der Regel so vorkonfiguriert, dass es mit 9600Baud auf den Pins D0 (Tx) und D1 (Rx) des Arduino Boards kommuniziert, sobald das Anwendungsprogramm vom Bootloader geladen ist und ausgeführt wird. Diese Pinwahl ist allerdings etwas ungünstig, da die Programmier- und Debugschnittstelle ebenfalls die Pins D0 und D1 für die Kommunikation benutzt. Man muss in diesem Fall zum Programmieren des Arduino Boards das GPS Shield immer abziehen und zum Betreiben wieder aufstecken.

Das GPS Shield hat allerdings ein Jumperfeld (UART Multiplexer) wo mit Hilfe von aufsteckbaren Jumpern die Transmit (Tx) und die Receive (Rx) Leitung des GPS Moduls auf andere Arduino Pins gelegt werden können. Beim GeoRex wurde Tx auf Pin D3 und Rx auf Pin D4 gelegt. Für die Zählimpulse wurde die Interrupt-Steuerung auf Pin D2 verwendet, so dass für die Programmierung und das Debugging (Serial Monitor) die Pins D0 und D1 verwendet werden konnten und daher das GPS-shield aufgesteckt bleiben konnte. Allerdings müssen für die serielle Kommunikation auf den Pins D3 und D4 die SoftwareSerial Methoden verwendet werden, die normalen Serial.print und Serial.println Routinen funktionieren nur die Pins D0 und D1. Von den digitalen Pins wird beim GeoRex noch der Pin D7 für eine Status LED verwendet, die anzeigt, wann das GPS Modul genügend Satelliten sieht, um die Position berechnen zu können.

D0	ArduinoUSB Rx
D1	ArduinoUSB Tx
D2	Zähler
D3	Rx for GPS Tx
D4	Tx for GPS Rx
...	
D7	GPS Status

Tab. 1: Liste der Arduino Pins die für den GeoRex verwendet werden

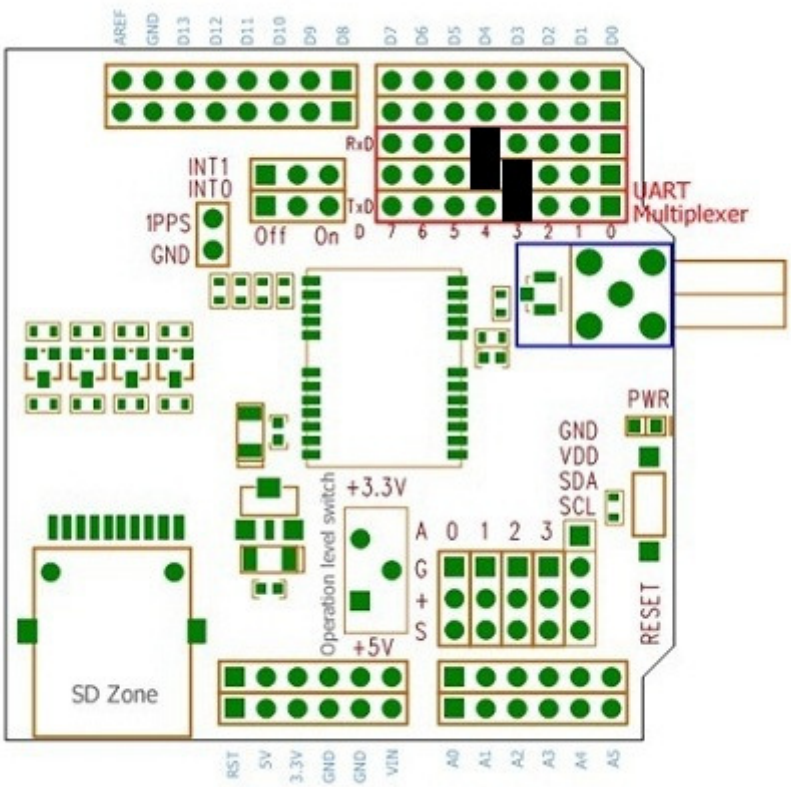

Abb. 3: Jumper Setting (schwarz) für die serielle Kommunikation mit dem GPS Modul

Ansonsten muss auf dem GPS-Shield nichts verändert werden. Der Schiebschalter für die Betriebsspannung muss auf 5V (Auslieferungszustand) stehen bleiben, da die Versorgunsspannung welche vom Arduino Board geliefert wird auch 5V beträgt.

Theoretisch ist die Kommunikation zum GPS Modul bidirektional, d.h. man kann das GPS-Modul auch vom Arduino anwendungsspezifisch konfigurieren. So könnte man beispielsweise die Baudrate verändern oder die Ausgabe auf ganz spezielle NMEA Messages beschränken (z.B. die RMC Message, Recommended Minimum Specific GNSS Data), in denen alle wichtigen Daten enthalten sind. Allerdings sollte man beim Umkonfigurieren sehr vorsichtig sein, da es zum Beispiel möglich ist, vom NMEA Protokoll in ein SiRF spezifisches Binärprotokoll zu wechseln mit dem der Arduino nicht mehr so ohne weiteres klar kommt. Im Prinzip ist diese Umkonfiguration aus dem voreingestellten Zustand für die GeoRex Anwendung nicht nötig.

```
$GPRMC,193702.000,A,4877.0749,N,00918.7926,E,0.00,,240213,,*10
$GPVTG,,T,,M,0.00,N,0.0,K*7E
$GPGGA,193703.000,4877.0749,N,00918.7926,E,1,05,5.0,487.0,M,48.0,M,,0000*51
$GPGLL,4877.0749,N,00918.7926,E,193703.000,A*3E
$GPGSA,A,3,15,12,25,18,26,,,,,,,,5.3,5.0,1.9*34
$GPGSV,3,1,12,12,52,242,49,15,36,181,48,25,17,242,50,18,12,259,47*77
$GPGSV,3,2,12,26,05,156,44,31,70,302,,16,65,237,,17,28,167,*72
$GPGSV,3,3,12,09,25,294,,22,16,299,,30,12,243,,28,06,059,*7B
$GPRMC,193703.000,A,4877.0749,N,00918.7926,E,0.00,,240213,,*11
$GPVTG,,T,,M,0.00,N,0.0,K*7E
$GPGGA,193704.000,4877.0749,N,00918.7926,E,1,05,5.0,487.0,M,48.0,M,,0000*56
$GPGLL,4877.0749,N,00918.7926,E,193704.000,A*39
$GPGSA,A,3,15,12,25,18,26,,,,,,,,5.3,5.0,1.9*34
```

Abb. 4: Beispiel für die NMEA-Ausgabe mit der voreingestellten Konfiguration

Der Geigerzähler

Das GeoRex System nutzt eine einfache digitale 3-Draht Schnittstelle zum Geigerzähler. Dabei wird angenommen, dass ein mit dem Detektor registriertes Strahlungsquantum durch die fallende Flanke eines Zählimpulses (Übergang von logisch 1 nach logisch 0) signalisiert wird. Da praktisch alle Typen von Geigerzählern ein solches Signal liefern (ob mit PIN-Dioden Detektor oder mit Geiger-Müller Zählrohr) ist der Typ des Geiegerzählers zunächst ohne Bedeutung. Die Verbindung zum Arduino wird über den Pin 2, für den eine Interrupt-Logik zur Verfügung steht hergestellt. Dazu muss noch die Ground (Gnd) und die 5Volt Versorgungsspannuns am Arduino Board angeschlossen werden.

Beim GeoRex System wird eine klassische Zählrohrvariante als Geigerzähler eingesetzt (Konzept und Aufbau siehe separates Dokument). Als Zählrohr wurde das Pancake-Zählrohr Beta-2-1 des Russischen Herstellers Consensus-Group (www.consensus-group.ru) verwendet, welches ein sehr großes Fenster hat und damit entsprechend schnell zählt. Wenn man während der Bewegung messen möchte, ist es entscheidend, dass man möglichst schnell eine statistisch repräsentative Menge an Zählimpulsen registriert. Das kann man aber nur mit einem entsprechenden Zählrohr mit großer aktiver Fläche erreichen.

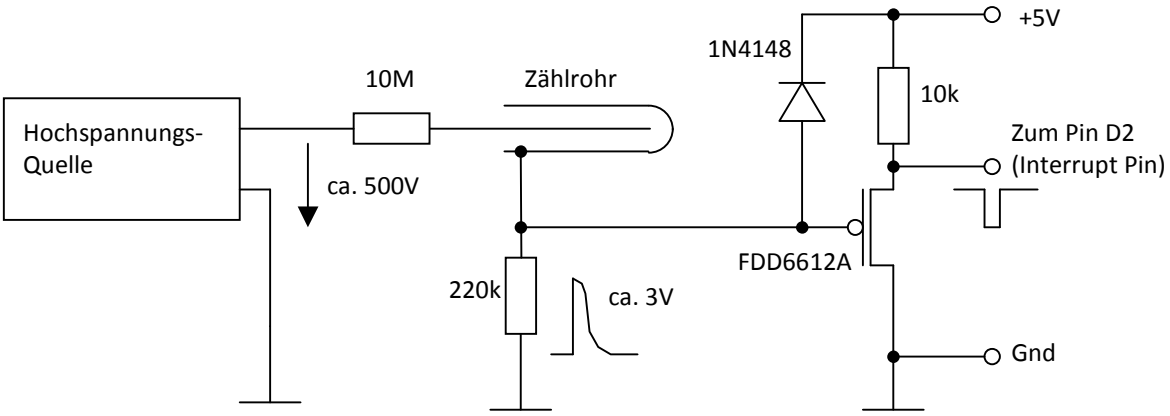

Abb. 5: Zählrohrbasierter Geigerzähler des GeoRex Systems

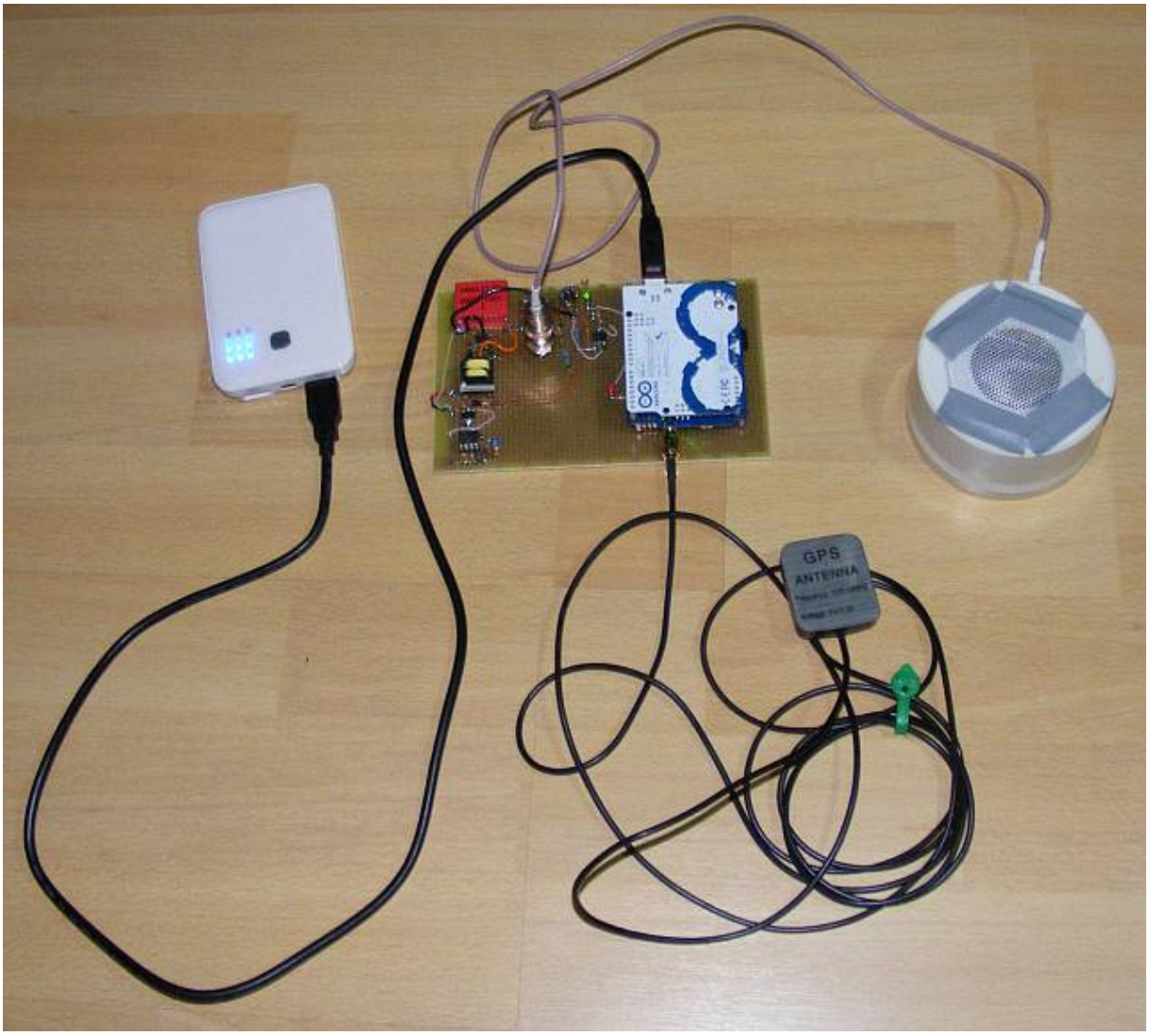

Abb. 6: GeoRex System mit Akku (links), Geigerzähler-Platine mit aufgestecktem Arduino Board und GPS-Shield (Mitte), GPS-Antenne (unten) und Pancake-Zählrohr (rechts)

Die Prototypen-Version des GeoRex Systems ist auf einer Europakarte aufgebaut, auf der sich links der Geigerzähler mit der Hochspannungserzeugung befindet. Rechts sind Stiftleisten angebracht, auf die das Arduino-Board mit aufgestecktem GPS-Shield aufgesteckt wird (kopfüber). Das Zählrohr ist über eine Kabel mit BNC-Buchse angeschlossen. Das Arduino-Board wird von einem Lithium-Polymer Akku mit USB-Stromversorgungsausgang (Powerbank) versorgt. Das Arduino-Board reicht die 5V-Stromversorgung an die Hochspannungserzeugung und das Geigerzähler-Interface auf der Europakarte weiter.

Die GeoRex Software für den Arduino-Mikrocontroller

Die Aufgabe der Software ist es, die NMEA Messages, welche vom GPS-Modul angeliefert werden von der seriellen Schnittstelle an Pin 3 und 4 einzulesen, die Position und Zeit aus einem bestimmten Message-Typs zu extrahieren und auf die SD-Karte zu schreiben. Gleichzeitig sollen die Zählimpulse vom Geigerzähler registriert werden und ein Zähler bis zu einem bestimmten Wert hochgezählt werden. Während des Hochzählens des Zählers bis zu diesem Maximalwert wird die Zeit gemessen. Sobald der maximale Wert für den Zähler

erreicht ist, wird die dafür benötigte Zählzeit ebenfalls auf die SD-Karte geschrieben und der Zähler zurückgesetzt. Die Daten auf der SD-Karte werden später am PC ausgewertet.

Die Arduino-Software wurde im wesentlichen aus Beispielen von der Arduino Webseite aufgebaut. Um die serielle Schnittstelle auf den Pins 3 und 4 implementieren zu können, muss der Headerfile SoftwareSerial.h eingebunden werden. Für die Kommunikation mit der Micro-SD-Karte wird SD.h benötigt. Für die String-Routinen wird string.h benötigt.

Die Variablen, welche global sind, aber in der Interrupt Service Routine verändert werden, sollten als volatile deklariert werden.

Wichtig ist nun das Statement:
SoftwareSerial gpsSerial = SoftwareSerial(3,4);

Dies erklärt die Pins 3 und 4 als serielle Schnittstelle von der später mit gpsSerial.read() die GPS-Daten gelesen werden können.

Da das GPS-Shield mit 9600 Baud Daten sendet, ist der Arduino ziemlich damit beschäftigt, diese Daten auch korrekt auf die SD-Karte zu schreiben. Viel Zeit für das Interrupt-Handling bleibt daher nicht übrig. Daher ist der gesamte Ablauf etwas zeitkritisch. Es hat sich gezeigt, das es günstiger ist, den ganzen Ablauf in der Setup() Routine zu halten und nicht die Loop() Routine zu verwenden. Allerdings wurde auch noch eine Monitor-Schnittstelle implementiert um mit dem PC die Daten per USB Verbindung überwachen zu können. Läuft alles so wie es soll, kann man diesen Teil auch wieder entfernen und hat dann mehr zeitliche Sicherheit.

Zunächst wird die Kommunikation mit dem GPS-Shield mit 9600 Baud und das Schreiben auf die SD-Karte initialisiert. Zum Loggen wird eine Datei mit dem Namen „datalog.txt“ angelegt. Existiert diese Datei, wird als Abschnittskennzeichnung der String "-----" und dann die neuen Daten angehängt. Sobald die Initialisierung abgeschlossen ist, wir der Interrupt auf Pin 2 (Int 0) vereinbart. Dabei wird vereinbart, dass bei Auftreten einer fallenden Flanke am Interrupteingang die normale Programmausführung unterbrochen und in die Routine „count“ verzweigt wird, welche den Zähler hochzählt.

Das Lesen der Daten vom GPS Shield erfolgt in der Endlosschleife while(1) {} . Die Daten werden zeilenweise in einen Pufferspeicher buffer[] gelesen, wobei das Zeilenende an Hand des Carriage Return Zeichens ‚\r’ erkannt wird. Solange kein Carriage Return auftritt, wird der Pufferspeicher gefüllt, wobei die nicht lesbaren Zeichen einfach überlesen werden. Wenn das Zeilenende erreicht ist, wird geprüft ob es sich um die NMEA-Message GLL handelt. Diese Message ist kürzer als die RMC Message und enthält Position, Zeit und den GPS Status. Wenn der Pufferspeicherinhalt die GLL Message enthält, wird er auf die SD-Karte geschrieben. Die anderen Messages werden ignoriert. Zu Debugging-Zwecken werden die Daten danach mit Stringroutinen aus der Message extrahiert und separat auf den Serial Monitor (USB zum PC) geschrieben. Diesen Block könnte man auch weglasen oder nur den Pufferspeicherinhalt auf den Serial Monitor schreiben. Aber zu Anschauungszwecken wurde dieser Teil im Code belassen.

Die Interrupt Service Routine wurde so kurz wie möglich gehalten um das Lesen vom GPS-Shield so wenig wie möglich zu behindern. In der augenblicklichen Version wird die Zeit für 100 Zählimpulse gemessen. Dazu merkt sich das System die Systemzeit wann der Zählerstand 100 erreicht wurde und bildet beim nächsten mal die Zeitdifferenz zu dieser Zeit. Das bewirkt zunächst, dass die Statistik der Zählimpulse gleich bleibt und die gemessenen Zeitintervalle, welche abgespeichert werden umso länger werden, je geringer die Zählrate ist. Das gemessene Zeitintervall wird zur Unterscheidung von der GLL Message mit einem vorangestellten %-Zeichen markiert und ist in der Einheit Millisekunden angegeben.

```
#include "Arduino.h"
#include "SoftwareSerial.h"
#include "string.h"
#include "SD.h"
#define LINEBUFSZ 90
#define MAXCNT 100

volatile int counter = 0;
volatile unsigned long time;
volatile unsigned long oldTime = 0;
volatile unsigned long dt = 0;

SoftwareSerial gpsSerial = SoftwareSerial(3,4);
char fileName[15] = "datalog.txt";
File myFile;

void setup()
{
  char c;
  int i;
  int cCnt, prmCnt;
  char buffer[LINEBUFSZ];
  char *prm;
  char nord[11];
  char east[11];
  char tim[11];
  char fix;

  Serial.begin(9600);
  gpsSerial.begin(9600);
  pinMode(10, OUTPUT);
  if (!SD.begin(10)) {
    Serial.println("SDcard not ready\n");
    return;
  }
  if (!SD.exists(fileName)) {
    myFile = SD.open(fileName, FILE_WRITE);
    myFile.println("###");
```

```
  myFile.flush();
}
else {
  myFile = SD.open(fileName, FILE_WRITE);
  myFile.println("-----");
  myFile.flush();
}

attachInterrupt(0, count, FALLING);


cCnt=0;
while (1) {
  while (gpsSerial.available()) {
    c=gpsSerial.read();
    if (c == '\r')
    {
      buffer[cCnt]=0;
      cCnt=0;
      //Serial.print(buffer[4]);
      if (buffer[4]=='L') // then it's a GLL message
      {
        //Serial.println(buffer);
        if (buffer[7]==',') { //no valid coord data
          Serial.println("***");
        }
        else {
          myFile.println(buffer);
          myFile.flush();

          prm = strtok (buffer,",");
          prmCnt=1;
          while (prm != NULL) {
           if (prmCnt==2) strcpy(nord,prm);
           if (prmCnt==4) strcpy(east,prm);
           if (prmCnt==6) strcpy(tim,prm);
           if (prmCnt==7) fix=prm[0];
           prm = strtok (NULL, ",");
           prmCnt++;
            }

            Serial.print(nord);
            Serial.print(" ");
            Serial.print(east);
            Serial.print(" ");
            Serial.print(tim);
            Serial.print(" ");
            Serial.println(fix);
```

```
        }
      }
    }
    else
    {
      if (c>' ') {
        buffer[cCnt]=c;
        cCnt++;
      }
    }
  }
 // while(1) loop
 }
}

void loop()
{
}

void count()
{
  counter++;
  if (counter == MAXCNT) {
    oldTime = time;
    time = millis();
    dt = time-oldTime;
    Serial.print('%');
    Serial.println(dt);
    myFile.print('%');
    myFile.println(dt);
    counter = 0;
  }
}
```

Abb. 7: Listing des Arduino-Programms für das GeoRex-System

```
$GPGLL,4839.4751,N,00931.8695,E,092029.000,A*3C
$GPGLL,4839.4749,N,00931.8684,E,092030.000,A*3D
$GPGLL,4839.4747,N,00931.8672,E,092031.000,A*3B
$GPGLL,4839.4744,N,00931.8661,E,092032.000,A*39
$GPGLL,4839.4740,N,00931.8650,E,092033.000,A*3E
$GPGLL,4839.4737,N,00931.8638,E,092034.000,A*37
$GPGLL,4839.4735,N,00931.8627,E,092035.000,A*3A
%163109
$GPGLL,4839.4733,N,00931.8615,E,092036.000,A*3E
$GPGLL,4839.4730,N,00931.8604,E,092037.000,A*3C
$GPGLL,4839.4730,N,00931.8604,E,092038.000,A*33
```

```
$GPGLL,4839.4730,N,00931.8604,E,092039.000,A*32
$GPGLL,4839.4730,N,00931.8604,E,092040.000,A*3C
$GPGLL,4839.4730,N,00931.8604,E,092041.000,A*3D
$GPGLL,4839.4719,N,00931.8548,E,092042.000,A*3E
```

Abb. 8: Ausschnitt aus einem Daten-Log (%163109 ist die Dauer eines Zählintervalls für 100 Zählimpulse in Millisekunden)

Das Auswerteprogramm zum GeoRex System

Das Auswerteprogramm wurde unter Matlab von Mathworks entwickelt, welches den großen Vorteil hat, dass unzählige Bibliotheksroutinen für aufwändige Analyse und Grafische Visualisierung vorhanden sind. Ein vergleichbares Freeware-Program ist scilab (www.scilab.org) das ursprünglich von der INRIA Forschungsgesellschaft in Frankreich entwickelt wurde. Im Prinzip kann das Programm aber in jeder anderen Programmiersprache geschrieben werden. Eine Umsetzung ist leicht möglich, da die Syntax von Matlab sehr intuitiv ist. Ein spezielles Element der Sprache ist allerdings die Tatsache, dass alle Variablen auch Matrizen und Vektoren (eindimensionale Matrizen sein können, was sehr zur Lesbarkeit beiträgt da viele Programmschleifen wegfallen, wenn eine Operation auf alle Elemente eines Arrays angewendet wird und dazu nur eine Programmzeile benötigt wird.

Die Aufgabe des Auswerteprogramms ist die Visualisierung der Messdaten entlang des vom GPS aufgezeichneten Wegs (track). Als Karten-Anzeigeprogramm wird das MapSource Programm von Garmin verwendet (der Nachfolger heißt BaseCamp) das man kostenlos von Garmins Webseite (www.garmin.com) herunterladen kann. Als Kartendaten werden Nop's Wanderreitkarte auf Open Street Map Basis (www.wanderreitkarte.de) benutzt. Um die GPS Daten im Anzeigeprogramm darstellen zu können, muss ein gpx-Datenformat erzeugt werden, das den Garmin Vorgaben entspricht. Das bedeutet, dass ein spezieller Header verwendet wird und nur gewisse gpx-spezifische XML-Tags verwendet werden dürfen.

Die Visualisierung der Messdaten wird nun einerseits dadurch erreicht, dass das Garmin-Anzeigeprogramm verschiedene Farben zur Darstellung der GPS-Tracks zulässt. Das heißt, die Farbe wird durch Schwellwerte für die gemessene Zählrate ausgewählt. Zusätzlich werden Wegpunkte angelegt deren Namen aus einem Index und der gemessene Zählrate zusammengesetzt ist. Diese Wegpunkte können für die Anzeige auch ausgeblendet werden.

Das Programm liest zunächst die Datalog Datei ein. Dabei werden die Messdaten des Geigerzählers und die GPS-Daten in Elemente einer Struktur eingelesen. Die Daten des Geigerzählers werden durch das vorangestellte %-Zeichen erkannt und die Daten des GPS-Moduls durch das jeder Message vorangestellte $-Zeichen.

Nach dem Einlesen der Daten und einer rudimentären Fehlerkontrolle wird die Ausgabe Datei im gpx-Format erzeugt (XML-basiert). Das nötige Format wurde durch Analyse von Beispiel-Dateien der Garmin GPS-Geräte entwickelt. Der Header wurde einfach kopiert und wird automatisch der erzeugten gpx-Ausgabedatei vorangestellt.

Die gpx-Ausgabedatei enthält einmal die Wegpunkte (Waypoints) mit den numerischen Angaben der gemessenen Zählrate sowie Track in verschiedenen Farben, welche die Höhe der Zählrate darstellen.

Die Zählrate für einen einer Messung zugeordneten Wegpunkt wird aus der Messzeit für das Erreichen des vorgegebenen maximalen Zählerstand berechnet. Die Zählrate für ein Wegstück (Tracksegment) wird aus dem Mittel zwischen beiden begrenzenden Wegpunkten berechnet. Für die numerische Angabe bei den Wegpunkten wird der Name aus dem Punkteindex und der Zählrate in „counts per minute“ (cpm) des Wegpunkts zusammengesetzt.

Ein gpx-Trackfile besteht immer aus einem oder mehreren Tracks, die als Track-Segmente in der gpx-Datei stehen. Einem Trackpunkt-Segment kann als Garmin spezifische Extension eine Farbe zugewiesen werden. Diese Farbe wird vorab anhand von Schellwerten bestimmt. Da das Verändern dieser Werte im Source Code sehr einfach ist, wurde dafür in der Prototypen Variante der Software keine Benutzerschnittstelle eingebaut. Allerdings wurde zur Überprüfung der Auswahl der Schwellwerte eine grafische Ausgabe der Strahlungsmesswerte und die Berechnung und Anzeige eines Messwert-Histogramms eingebaut.

Die Trackpunkte werden aus den Koordinatendaten des GPS-Moduls erzeugt und mit den entsprechenen XML-Tags versehen. Solange die Trackpunkte dieselbe Farbe zugeordnet haben wie der vorige Trackpunkt wird der Trackpunkt dem laufenden Segment zugeordnet. Sobald die Farbe sich die einem Trackpunkt zugeordnete Farbe ändert, wird das bisherige Segment abgeschlossen und ein neues Tracksegment mit der neuen Farbe begonnen.

Bei der Wahl der Schwellen sollte man darauf achten, dass nicht zu viele Tracksegmente entstehen, sonst lässt die Aussagekraft nach und das Anzeigeprogramm bekommt unter Umständen Probleme. Um das einigermaßen kontrollieren zu können wird am Ende die Zahl der erzeugten Tracks ausgegeben.

```
<wpt lat="49.606488" lon="7.172250">
<name>#002:43</name>
<sym>Flag, Blue</sym>
</wpt>

<name>Track 007</name>
<extensions>
        <gpxx:TrackExtension
xmlns:gpxx="http://www.garmin.com/xmlschemas/GpxExtensions/v3"
><gpxx:DisplayColor>Cyan</gpxx:DisplayColor>
</gpxx:TrackExtension>
</extensions>
<trkseg>
<trkpt lat="49.612262" lon="7.168798"/>
<trkpt lat="49.6122650" lon="7.1687800"/>
<trkpt lat="49.6122667" lon="7.1687583"/>
<trkpt lat="49.6122717" lon="7.1687367"/>
</trkseg>
</trk>
```

Abb. 9 : Wegpunkt und (gekürztes) Tracksegment der gpx Ausgabedatei

```
%GeoRex Analysis Software V3.0
clear;
gpsTimeOffset=+1*3600; %trkData are in MEZ
infile='DATALOG.TXT';
infid=fopen(infile,'r');

fdata=textread(infile,'%s','delimiter','\n');
fdata=fdata(1:length(fdata));
fclose(infid);

nrLines=length(fdata);
nrRadVals=1;
nrTrkPts=0;
radData(1).val=200000; % the first val
lineBuffer='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
for k=1:nrLines
   %fprintf(1,'%s\n',char(fdata(k)));
   line=fdata{k};
   if (line(1)=='%')  %RadData
      nrRadVals=nrRadVals+1;
      radData(nrRadVals).val=sscanf(line(2:length(line)),'%d');
      radData(nrRadVals).north=geoData(nrTrkPts).north;
      radData(nrRadVals).east=geoData(nrTrkPts).east;
      %fprintf(1,'%d\n', radData(nrRadVals));
   elseif (line(1)=='$')
       if (~strcmp(line(1:30),lineBuffer(1:30)))
```

```
        nrTrkPts = nrTrkPts + 1;
        if nrTrkPts == 699
           debug = 1;
        end
        %fprintf(1,'%s\n',char(fdata(k)));
        [str, remain] = strtok(line, ',');
        [str, remain] = strtok(remain, ',');
        geoData(nrTrkPts).north = sscanf(str(1:2),'%f')+sscanf(str(3:9),'%f')/60;
        [str, remain] = strtok(remain, ',');
        [str, remain] = strtok(remain, ',');
        geoData(nrTrkPts).east = sscanf(str(1:3),'%f')+sscanf(str(4:10),'%f')/60;
        [str, remain] = strtok(remain, ',');
        [str, remain] = strtok(remain, ',');
        geoData(nrTrkPts).timestr = sscanf(str,'%f');
        geoData(nrTrkPts).radIdx=nrRadVals;

        if (isempty(geoData(nrTrkPts).north) || isempty(geoData(nrTrkPts).north))
           fprintf(1,'Coord Error in line %d north: "%.7f" east: "%.7f"\n', k,
geoData(nrTrkPts).north, geoData(nrTrkPts).east);
        end

        if ~((geoData(nrTrkPts).north > 46.0) && (geoData(nrTrkPts).north < 50.0) &&
(geoData(nrTrkPts).east > 7.0) && (geoData(nrTrkPts).east < 10.0))
           fprintf(1,'Coord Error in line %10.0f north: "%.7f" east: "%.7f"\n', k,
geoData(nrTrkPts).north, geoData(nrTrkPts).east);
        end
        lineBuffer=line;
      end
   end
end
nrRadVals=nrRadVals+1;
radData(nrRadVals).val=200000; % the last val


gpxHeader='<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <gpx
xmlns="http://www.topografix.com/GPX/1/1" creator="MapSource 6.11.6" version="1.1" >';
trkExtHeader=' <gpxx:TrackExtension
xmlns:gpxx="http://www.garmin.com/xmlschemas/GpxExtensions/v3" >';
trkFileName='GeoRex.gpx';
outfid=fopen(trkFileName, 'w');

%cals doseRate and assign color
for k=1:nrTrkPts-1

geoData(k).doseRate=(radData(geoData(k).radIdx).val+radData(geoData(k).radIdx+1).val)/2;
%arithMean between end points
      if geoData(k).doseRate < 100000
        geoData(k).color='Red';
```

```
        elseif geoData(k).doseRate < 110000
           geoData(k).color='Magenta';
        elseif geoData(k).doseRate < 120000
           geoData(k).color='Yellow';
        elseif geoData(k).doseRate < 130000
           geoData(k).color='Cyan';
        elseif geoData(k).doseRate > 1E6
           geoData(k).color='Blue';
        elseif geoData(k).doseRate > 1E6
           geoData(k).color='White';
        elseif geoData(k).doseRate > 1E6
           geoData(k).color='LightGray';
        elseif geoData(k).doseRate > 1E6
           geoData(k).color='DarkGray';
        else
           geoData(k).color='Black';
        end
end
nrGeoData=nrTrkPts;
% calc doserate histogram just for info
hist([geoData.doseRate]);
figure;plot([geoData.doseRate],'*-');

%create gpxFile
fprintf(outfid, gpxHeader);
%create waypoints
for k=2:nrRadVals-1
  fprintf(outfid, '<wpt lat="%.6f" lon="%.6f">\n',radData(k).north, radData(k).east);
  fprintf(outfid,     '<name>#%03.0f:%.0f</name>\n',     k-1,     100*60/(radData(k).val/1000));
%Anzeige in ipm
  fprintf(outfid, '<sym>Flag, Blue</sym>\n');
  fprintf(outfid, '</wpt>\n');
end
%create track
m=1;
trkColor=geoData(1).color; %color is determined from first trkpt
nrTracks=1;
while m<nrGeoData
    fprintf(outfid, '<trk>\n');
    fprintf(outfid, '<name>Track %03.0f</name>\n',nrTracks);
    fprintf(outfid, '<extensions>\n');
    fprintf(outfid, trkExtHeader);
    fprintf(outfid, '<gpxx:DisplayColor>%s</gpxx:DisplayColor>\n',trkColor);
    fprintf(outfid, '</gpxx:TrackExtension>\n');
    fprintf(outfid, '</extensions>\n');
    fprintf(outfid, '<trkseg>\n');
    fprintf(outfid, '<trkpt lat="%.6f" lon="%.6f"/>\n', geoData(m).north, geoData(m).east);
    m=m+1;
```

```
   while strcmp(geoData(m).color,trkColor)
      fprintf(outfid, '<trkpt lat="%.7f" lon="%.7f"/>\n', geoData(m).north, geoData(m).east);
      m=m+1;
   end
   fprintf(outfid, '</trkseg>\n');
   fprintf(outfid, '</trk>\n');
   if m>=nrGeoData
      break;
   end
   trkColor=geoData(m).color;
   nrTracks=nrTracks+1;
   m=m-1;              %start next track with last coord of last track
end

fprintf(outfid, '</gpx>\n');
fclose(outfid);
fprintf(1,'Final Nr Tracks %d\n',nrTracks);
```

Abb. 10: Listing der GeoRex Auswerte-Software implementiert in Matlab (Prototypen Version)

Anwendungsbeispiele für das GeoRex System

Es ist natürlich klar, dass wenn es in der eigenen Landeshauptstadt eine amtlich vermessene Referenzstrecke gibt, dass diese als erster Test herangezogen wird.

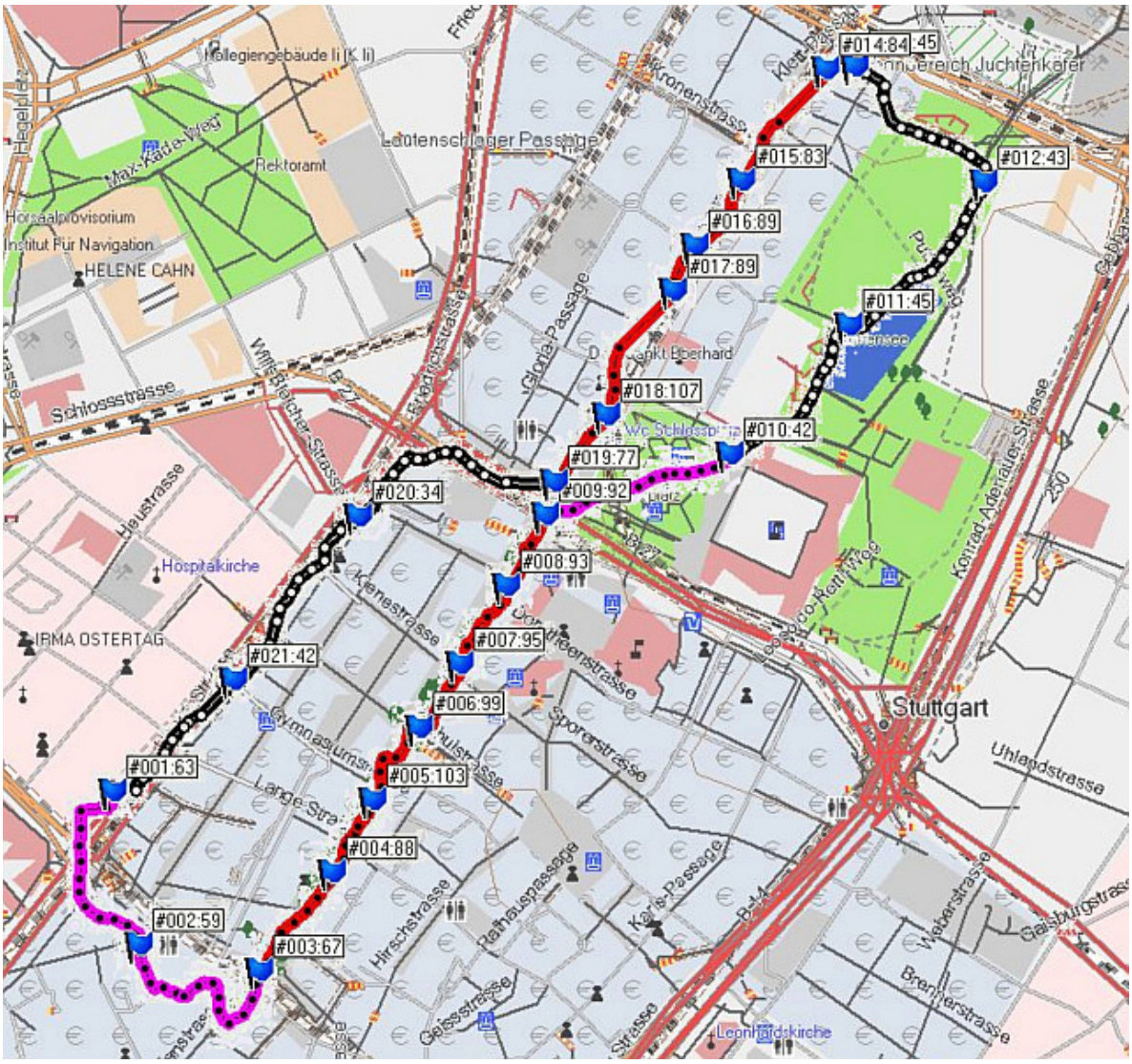

Abb. 11: Die Stuttgarter Referenzstrecke für Strahlungsmessungen: Der „Radioaktivitäts-Achter“ über die Königsstraße und die Nachbarstraßen. Hier gemessen mit dem GeoRex System. Die Messwerte (#idx:Zählrate in cpm) sind als Wegpunkte dargestellt.

Abb. 11: zeigt also den berühmten „Königstraßen-Achter“, ein Weg, der auf der oberen Königstraße beginnt, bis zum Schlossplatz führt, dann durch den Schlossgarten bis zum Hauptbahnhof, von da aus zurück auf der unteren Königstraße bis wieder zum Schlossplatz und von dort aus zurück auf der Theodor-Heuss-Strasse zum Rotebühlplatz. Ganz deutlich kann man die fast 3-mal höheren Zählraten auf der Königstraße erkennen, was an dem dort verlegten Flossenbürger Granit liegt. Dieser enthält erhebliche Mengen Uran, im Gegensatz zum Asphalt des restlichen Wegs. Laut Messung des LUBW, zeigt der Granit auf der Königstraße eine Dosisrate in Gonadenhöhe von bis zu 0.3uSv/h, während die Dosisrate auf dem Asphalt etwa 0.1uSv/h beträgt. Man kann also sagen, dass die Zählrate von 100 cpm des GeoRex-Systems einer Dosisrate von etwa 0.3uSv/h entspricht.

Ein weiteres, weitaus anspruchsvolleres Anwendungsbeispiel ist das Vermessen der Schiefervorkommen in Holzmaden. Schiefergestein ist ebenfalls für einen gewissen Urangehalt bekannt. Abb. 12 zeigt einen Ausschnitt eines Wegs in der Nähe des Urweltmuseums von Holzmaden am Rande der schwäbischen Alb. Man kann deutlich erkennen, dass sich auf dem Gelände eines Schiefersteinbruchs die Zählrate etwa verdoppelt. Man kann aber auch auf dem freien Gelände erkennen, dass die Zählrate etwas höher ist, als im Wald. In der Natur sieht man auf dem freien Gelände immer wieder Stellen mit offenliegendem Schiefer. Im Wald ist der Boden dagegen durch eine dicke Humusschicht überdeckt. Die Schwellwerte für die Farbgebung der Tracks wurden entsprechend angepasst.

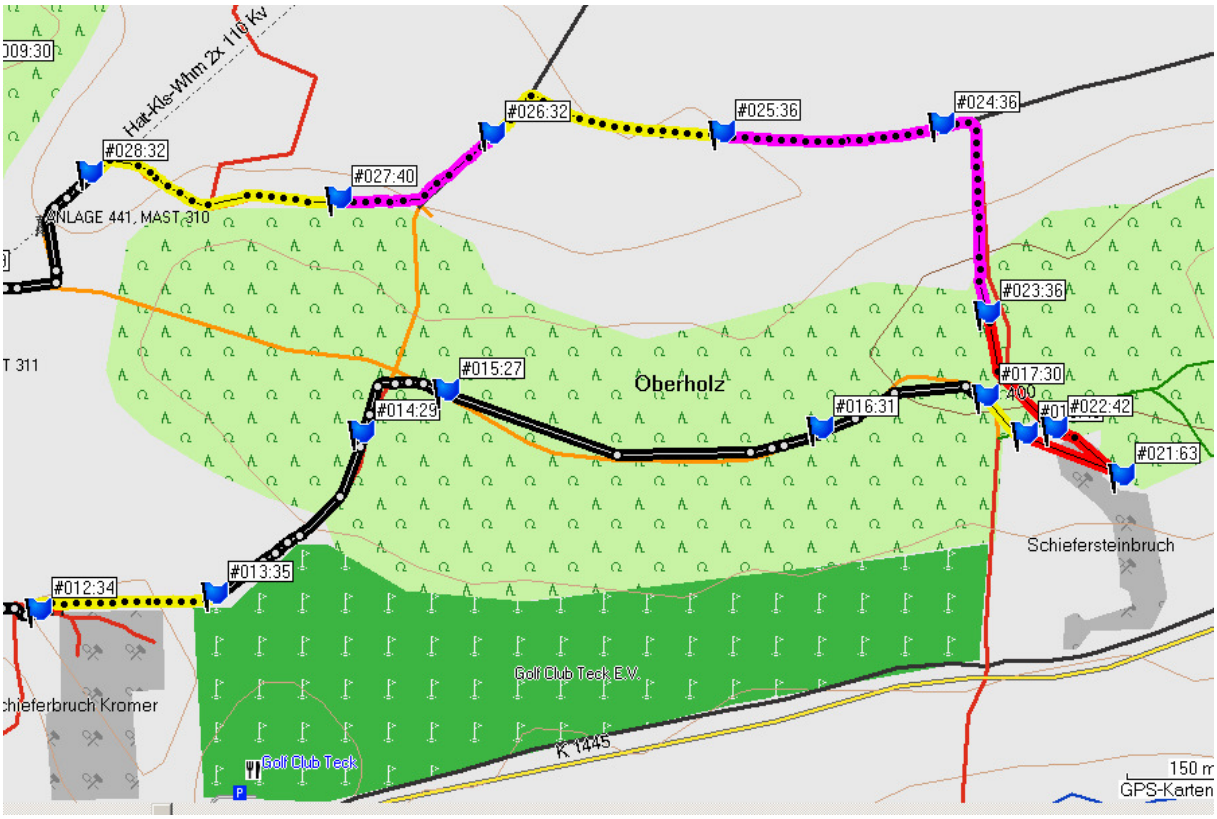

Abb. 12: Geo-referenzierte Radioaktivitätsmessungen an einem Steinbruch bei Holzmaden mit dem GeoRex-System

Insgesamt kann man deutlich erkennen, dass geo-referenzierte Messungen der lokalen Radioaktivität mit Hilfe eines sehr kostengünstigen Arduino-Mikrcontrollers sehr einfach möglich sind. Wenn man ein schnellzählendes Zählrohr als Geigerzähler benutzt, kann man auch sehr schwache Kontaminationen von Böden relativ schnell erfassen und in einer Karte darstellen. Das aber ist genau das, was nach einem Nuklear-Unfall oder einer Nuklear-Katastrophe sofort gemacht werden müsste um die Zivilbevölkerung rechtzeitig und richtig vor den Kontaminationen mit radioaktivem Fallout zu schützen. Das sollte man auch besser

vorher schon ein paar mal geübt haben und das Wissen auf möglichst viele potentielle Helfer verteilt haben, denn ein einzelner Experte am BfS reicht in solchen Situationen kaum. Ein fest installiertes Ortsdosisleistungs-Mess-Netzwerk, wie es in der BRD nach Tschernobyl aufgebaut wurde, ist zwar durchaus eine gewisse grobe Hilfe, aber das geographische Raster, welches damit bisher erreicht wurde, ist noch viel zu groß angesichts der enormen Tragweite einer daraus abgeleiteten großflächigen Evakuierung der Bevölkerung.

Abschließend ist hier noch eine Danksagung nötig: Wesentliche Teile des Konzepts und die grundlegenden Anstöße zur Verwendung des Arduino gehen auf eine Studienarbeit meines Studenten Jakob an der Dualen Hochschule in Stuttgart zurück, der nach diesem Prinzip unter Verwendung eines Gas-Sensors ein System für geo-referenzierte Messungen der Luftqualität entwickelt hat.