Entwicklung einer Antennensteuerung für einen Stratosphärenballon

Maturaarbeit 2015 im Bereich Technik an der Kantonsschule Sursee

Verfasser:
Lars Horvath
Starenweg 1
6210 Sursee

Betreuer:
Dr. C. Wildfeuer
Moosgasse 9a
6210 Sursee

Danksagungen

Herrn Dr. C. Wildfeuer danke ich recht herzlich für die angenehme Zusammenarbeit und hervorragende Betreuung.

Ich danke dem Radio Club Sursee und speziell Kari Künzli (HB9DSE), Casimir Schmid (HB9WBU), Thomas Fischer (HB9FPQ) und Dr. Jürg Regli (HB9BIN) für ihre Unterstützung.

Marco Oberholzer (HB9ZCF) möchte ich für die hilfreiche Unterstützung während des Ballonstarts und für die Freigabe der Fotos herzlich danken.

Herrn Leander Gutzwiller (HB9RMW) danke ich herzlich für die Bereitstellung der Rotoren und den Steuergeräten.

Mein Dank gilt auch meinem Vater, der mich stets unterstützt hat und immer für fachliche Diskussionen bereit war.

Ferner gilt mein Dank allen, die an meiner Arbeit mitgewirkt haben und mich unterstützt haben.

1 Abstract

Die vorliegende Maturarbeit soll zeigen, ob die Entwicklung und Konstruktion einer automatischen Antennensteuerung, mit den mir zur Verfügung stehenden Mitteln möglich ist. Diese Antennensteuerung soll automatisch einem Stratosphärenballon folgen und so den Sichtkontakt bzw. die unterbruchsfreie Übertragung von Daten sicherstellen.

Die folgende Dokumentation wird mein Vorgehen wissenschaftlich darlegen, erläutern und mein Produkt vorstellen.

Inhaltsverzeichnis

2 Einleitung

Die Faszination des Gedankens, unsere Erde einmal aus der Stratosphäre zu sehen, hat mich dazu bewegt, diese Arbeit zu machen. Ist es nicht phänomenal, dass eine 40cm grosse ‚Schachtel', ausgestattet mit Technik, aus 36 km Höhe uns live ein Videosignal zur Erde sendet? Ich finde ja.
Ich habe schon immer mit Elektronik gebastelt und diese Leidenschaft wohl von meinem Vater geerbt und auch gelernt. Das Ziel dieser Arbeit ist eine automatische Antennensteuerung zu entwickeln, die eine Richtantenne sowohl in Azimut als auch in Elevation auf einen fliegenden Statosphärenballon permanent ausrichtet.
Die Idee dazu ist aus einem Projekt entstanden, welches mein Mathematiklehrer Dr. Christoph Wildfeuer auf die Beine gestellt hat. Er hatte das visionäre Ziel, ein HD-Live-Video von einem Stratosphärenballon mittels Digitalfunk an die Bodenstation zu übertragen. Eines Tages ist Herr Wildfeuer auf mich zugekommen und hat mich gefragt, ob ich interessiert wäre, an der Entwicklung einer Bodenstation für sein Projekt mitzuwirken. Ich sagte sofort ja und startete meine Arbeit. Ein solches Projekt wurde mit der verwendeten Technik und der grossen Distanz, von über 100 km Luftlinie zwischen Ballon und Bodenstation, noch nie realisiert (oder zumindest nicht dokumentiert). Man kann hier also von einem innovativen Projekt **–** ja sogar von einer wissenschaftlichen Weltpremiere **–** sprechen.
Mein Beitrag zu diesem grossen Projekt ist die Bodenstation. Meine Aufgabe bestand darin, mit Hilfe von Richtantennen, die Übertragung des Videosignals sicherzustellen. Dabei musste ich vorhandene, manuell gesteuerte Azimut- und Elevationsrotoren mit einem Arduino-Mikrokontroller verbinden und eine komplette Software zur automatischen Steuerung der Rotoren entwickeln. Diese Software muss die aktuellen Positionsdaten des Ballons aus dem Internet herunterladen, die relevanten Daten herausfiltern, die Steuerimpulse für die Motoren berechnen und diese an die Motoren senden.

In diesem schriftlichen Teil meiner praktischen Arbeit erkläre ich als Erstes die Theorie, zeige wie ich die Aufgabe realisiert habe, diskutiere die Ergebnisse und reflektiere nochmals das Ergebnis.

3 Theorie:

3.1 Antenne und Funktechnik

3.1.1 Elektromagnetische Wellen

„Die von einer Sendeantenne abgestrahlte Energie pflanzt sich in Form von elektromagnetischen Wellen im Raum fort. Dieser Vorgang lässt sich an einer unbewegten Wasseroberfläche, die durch einen eintauchenden Gegenstand zur Wellenbildung angeregt wird, veranschaulichen. Die fortschreitende Wellenbewegung entsteht, ohne dass eine Strömung im Wasser nachweisbar ist. […] Der Wellenzug pflanzt sich kreisförmig fort, ohne dass die Wasseroberfläche weiterbewegt wird.“ [1]

3.1.2 Verwendete Frequenzen

Definition: „Frequenz f = Anzahl der Wellenbewegungen (Wellenlängen), die sich in einer Sekunde ausbilden.“ [1]

- Für die Übertragung des Videosignals wird die Frequenz 436.500MHz mit 2MHz Bandbreite verwendet. Dies liegt im 70-Zentimeter-Band[1]; Englisch UHF (Ultra High Frequencies)
- Das APRS Signal wird auf der Frequenz 144.800 MHz übertragen. Diese Frequenz liegt im 2-Meter-Band[2]; Englisch VHF (Very High Frequencies).

3.1.3 Verwendete Antennen

Für die Übertragung des Video-Signals wurden zwei Antennen verwendet. Eine 14-Element-Kreuz-Jagi und eine selbst gebaute Helix Antenne. Die Kreuz-Jagi habe ich selbst – mit Hilfe von Herrn Wildfeuer – zusammengebaut. Die Helix Antenne hat Casimir Schmid, zusammen mit Herr Wildfeuer entworfen und gebaut. Sie hat einen Vorteil, da sie das, sich in alle Richtungen ausbreitende und bewegende Signal, besser empfangen kann.

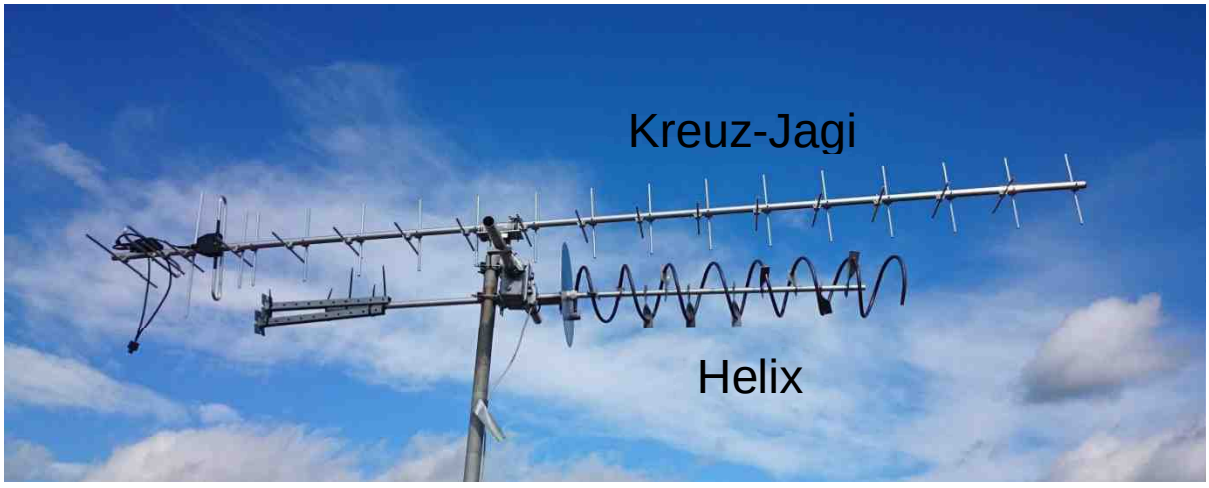


Abbildung 1: Kreuz-Jagi und Helix Antenne

[1] Als 70-Zentimeter-Band bezeichnet man den Frequenzbereich von 400 MHz bis 460 MHz. [8]
[2] Das 2-Meter-Band ist der Frequenzbereich von 144 MHz bis 146 MHz. [9]

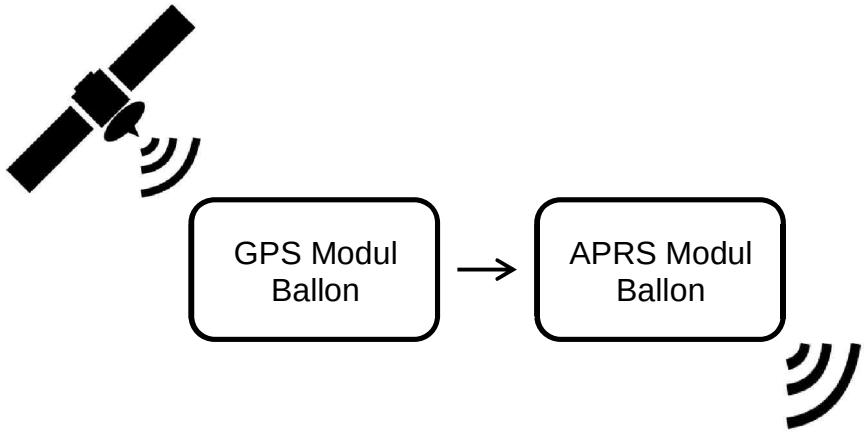
3.2 GPS

GPS (Global Positioning System) ist ein globales Navigationssatellitensystem zur Positionsbestimmung. [2]

Seit 1993 kreisen 24 Satelliten in 20000 km Höhe um die Erde und bilden zusammen mit Bodenstationen das Navigationssystem GPS. Jeweils vier Satelliten haben eine gemeinsame Umlaufbahn, sodass es sechs verschiedene Umlaufbahnen gibt. An Bord eines jeden Satelliten befinden sich hochpräzise Atomuhren, deren Synchronisation ständig per Funk überwacht wird. Alle 24 Satelliten senden nach jeweils einer Millisekunde codierte Funksignale auf der Frequenz 1,6 GHz. Neben der Kennung des jeweiligen Satelliten enthalten die Signale die aktuellen Bahnparameter und die exakte Zeit der Signalaussendung. Das Satellitensystem ist so aufgebaut, dass bei freier Sicht zu jeder Zeit und an jedem Ort der Erde die Signale von mindestens vier Satelliten empfangen werden können. Notwendig wäre zur Ortsbestimmung eigentlich nur der Empfang von drei Satelliten. Dann müsste der Empfänger aber selbst eine hochpräzise Atomuhr besitzen. Das ist jedoch nicht notwendig, denn das vierte Signal liefert die genaue Zeitkoordinate, mit der der Empfänger seine Zeit synchronisiert. Ein Empfänger kann aus der Laufzeit der vier Signale die momentane Position nach Länge, Breite und Höhe auf 7m genau berechnen. [3]

3.2.1 GPS Empfänger

Zur Nachführung der Antenne wird die aktuelle Position des Ballons benötigt. Dafür hat der Ballon ein GPS Empfänger, welcher aktuelle Position und Höhe des Ballons an das APRS Modul (siehe Grafik 1) weiter gibt. Dieses sendet dann die GPS Daten an die Bodenstation.



Grafik 1: GPS im Ballon

3.3 APRS

3.3.1 Was ist APRS?

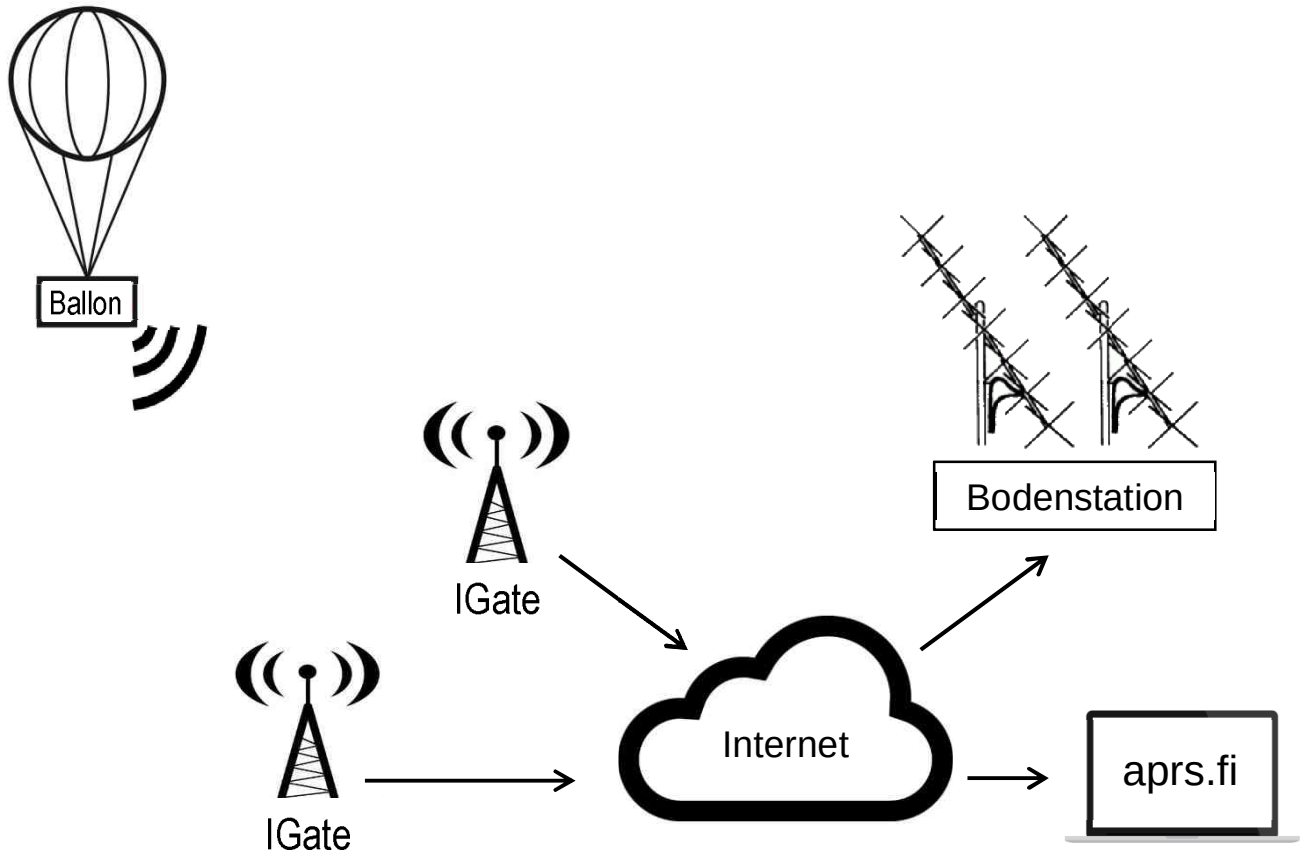
Vereinfacht aus: [4]

APRS (Automatic Packet Reporting System) wurde in den 80er Jahren vom Amerikaner Bob Bruninga WB4APR entwickelt. Es ist eine auf dem Packet Radio Standard basierende Betriebsart von Amateurfunk. Ziel ist eine digitale Kommunikation, die Kurznachrichten, Objektinformationen (Telemetrie Daten), Positionsdaten (GPS), Wetterdaten und vieles mehr einem breiten Publikum zur Verfügung stellt. APRS kann ausserdem Kurznachrichten verschicken und Notrufe mit GPS-Position absetzen.

Die APRS Pakete werden in Europa v.a. auf der Frequenz 144.800 MHz verbreitet und können von anderen Stationen direkt oder via Digipeater empfangen werden. Weiter besteht die Möglichkeit, dass die Pakete über einen I-Gate (Internet Gateway) ins Internet gelangen, wo man sie u.a. auf Karten (z.B. aprs.fi) betrachten kann. APRS-Betrieb ist auch über die Internationale Raumstation (ISS) und gewisse Amateurfunksatelliten möglich.

3.3.2 Verwendung

Für unser Projekt haben wir APRS – wie in der folgenden Grafik 2 ersichtlich – verwendet. Der Ballon sendet auf der Frequenz 144.800 MHz sein Kennzeichen und Position. Das Signal wird von IGates ins Internet gespiesen. Von dort ist es dann zugänglich, um z.B. die Position auf einer Karte anzuzeigen oder auch für meine Bodenstation.



Grafik 2: Funktion APRS für Ballonstart

3.4 Koordinaten Berechnung

Mit Koordinaten Berechnung ist die Berechnung der Azimut- und Elevationswinkel gemeint. Diese Soll-Winkel werden für die Nachführung der Antenne benötigt.

3.4.1 Umrechnung von ellipsoidischen WGS84-Koordinaten in Schweizer CH1903 Projektionskoordinaten (Näherungsformeln)

Vereinfacht aus: [5]

Bezeichnungen
φ, λ = geogr. Breite und Länge bezüglich Greenwich
Y, X = Zivilkoordinaten im Schweiz. Projektionssystem
y, x = Landeskoordinaten (Militärkoordinaten) LV03

1. Breite φ und Länge λ sind in Sexagesimalsekunden ["] umzuwandeln

2. Hilfsgrössen (Breiten- und Längendifferenz gegenüber Bern in der Einheit [10000"]) berechnen:
$$\varphi' = (\varphi - 169028.66'')/10000$$
$$\lambda' = (\lambda - 26782.5'')/10000$$

3.
$$y\,[m] = 600072.37 + 211455.93\,\lambda' - 10938.51\,\lambda'\varphi' - 0.36\,\lambda'\varphi'^2 - 44.54\,\lambda'^3$$

$$x\,[m] = 200147.07 + 308807.95\,\varphi' + 3745.25\,\lambda'^2 + 76.63\,\varphi'^2 - 194.56\,\lambda'^2\varphi' + 119.79\,\varphi'^3$$

$$h'\,[m] = h - 49.55 + 2.73\,\lambda' + 6.94\,\varphi'$$

4. Zahlenbeispiel

gegeben: φ = 46° 2' 38.87" λ = 8° 43' 49.79" h = 650.60 m

⇒ φ' = -0.326979 λ' = 0.464729

⇒ y = 699 999.76 m x = 99 999.97 m h' = 600.05 m

aus NAVREF: y = 700 000.0 m x = 100 000.0 m h' = 600 m

Diese Näherungen sind für die ganze Schweiz besser als 1 Meter in der Lage und 0.5 Meter in der Höhe.

Aufgrund der Öffnungswinkel der Antenne (ca. 35 Grad) sind die Fehler bei der Umrechnung und Formatierung der Koordinaten zu vernachlässigen, da sich diese kaum mit einem relevanten Betrag auf den Elevation und Azimut Winkel auswirken.

3.5 Arduino

Vereinfacht aus [6]

Arduino ist eine Open Source Plattform, welche es ermöglicht, einfache Steuerungen zu entwickeln.

Neben der Hardware gehört zu Arduino auch Software, wie z.B. die eigene Entwicklungsumgebung (basiert auf Processing) und Beispiel-Programme für die Mikrocontroller.

3.5.1 Arduino Yun

Der Arduino Yun unterscheidet sich von den anderen Arduinos, in dem er über einen Linux Teil verfügt, mit welchem er über Wlan, USB und Ethernet kommunizierten kann.

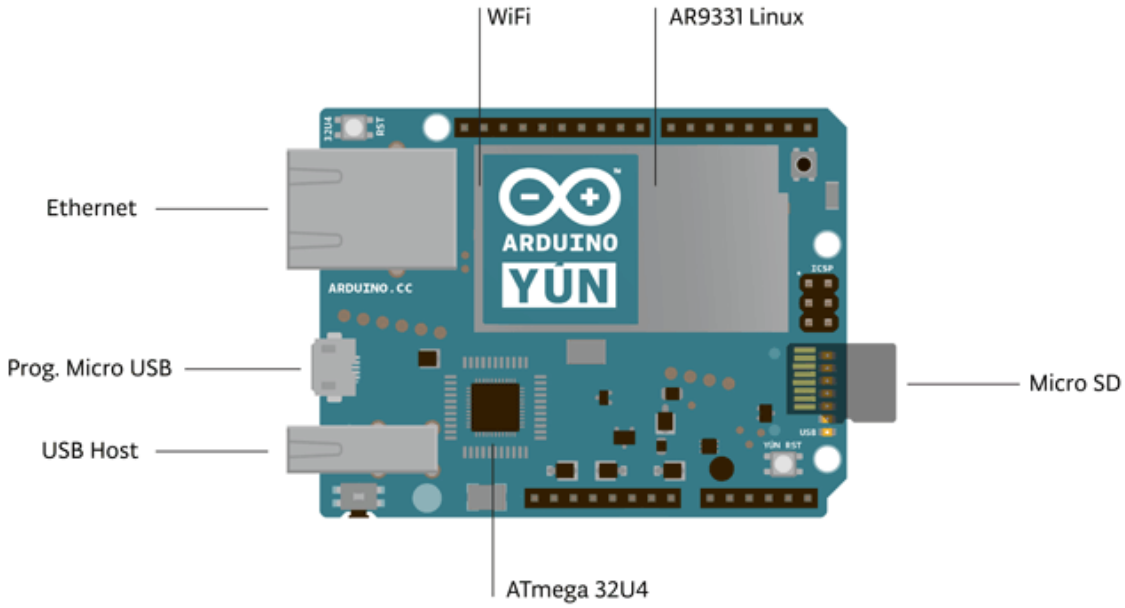


Abbildung 2: Arduino Yun Aufbau

Somit hat der Arduino Yun zwei Teile, die durch eine sogenannte Bridge verbunden sind:

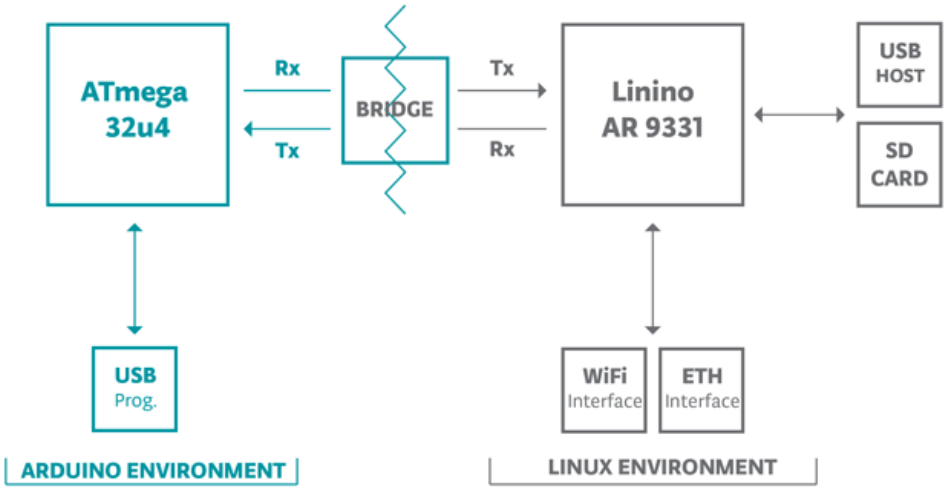


Abbildung 3: Arduino Yun Bridge

Der Arduino Yun hat folgende Spezifikationen:

AVR Arduino Microcontroller

Microcontroller	ATmega32u4
Operating Voltage	5V
Input Voltage	5V
Digital I/O Pins	20
PWM Channels	7
Analog Input Channels	12
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (of which 4 KB used by bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

Linux Microprocessor

Processor	Atheros AR9331
Architecture	MIPS @400MHz
Operating Voltage	3.3V
Ethernet	IEEE 802.3 10/100Mbit/s
WiFi	IEEE 802.11b/g/n
USB Type-A	2.0 Host
Card Reader	Micro-SD only
RAM	64 MB DDR2
Flash Memory	16 MB
Length	73 mm
Width	53 mm
Weight	32 g

4 Realisierung

Im folgenden Kapitel wird die Realisierung und Entwicklung dargestellt.

4.1 Erste Ideen, Prototypen

Das Ziel war klar: eine automatisch nachführende Antennensteuerung. Doch wie sollte ich das umsetzten?

Ich bekam netterweise zwei professionelle Antennenmotoren von Leander Gutzwiller (HB9RMW) zur Verfügung gestellt. Mit den Motoren kamen zwei analoge Steuergeräte, die je einen Motor per Taster steuern und die Position der Antenne anzeigen konnten. Die Motoren funktionierten mit den vorhandenen Steuergeräten einwandfrei, doch konnte ich daraus keine autonome Steuerung bauen. Die Motoren haben ein Potentiometer (siehe Schema 1), welches mit einer Referenzspannung die aktuelle Position ‚messen' kann.

Die ersten Skizzen machte ich mit einem Beaglebone als Controller. Aufgrund seiner niedrigen Spannungen auf den I/O-Pins wäre eine externe Referenzspannung für die Potentiometer notwendig gewesen. Ausserdem ist der Beaglebone nicht fähig, grössere Spannungen zu schalten, weshalb ich eine Relaiskarte gebraucht hätte oder C-Control Steuerungen mit Relais. (siehe Schemas im Anhang)

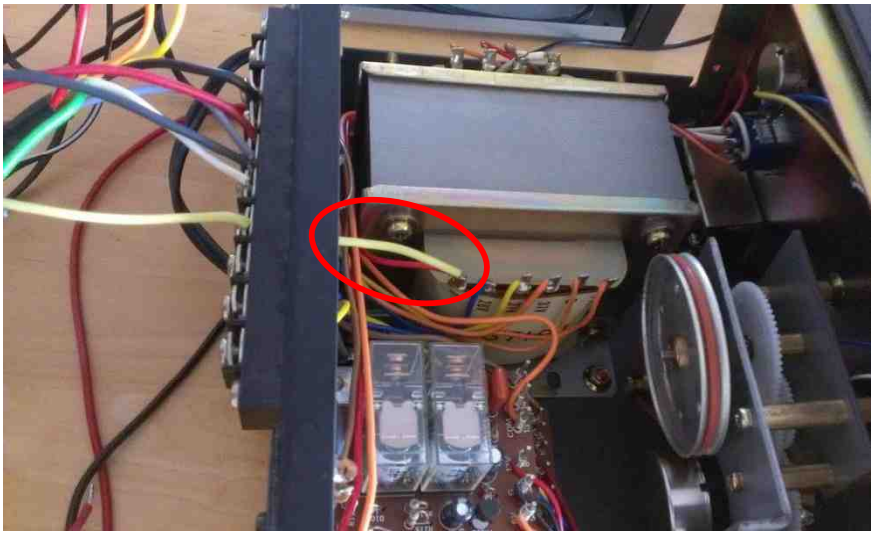
Zum Glück kam Herr Wildfeuer auf den Arduino Yun, welcher sich aufgrund seiner einfachen Programmierung und Spezifikationen perfekt für meine Arbeit eignete.

4.2 Umbau der AC-Netzgeräte

Da die Motoren mit den vorhandenen Netzgräten schon funktionierten, machte ich mir nicht die Mühe, selbst neue Netzgräte zu fertigen, sondern baute die vorhandenen einfach um. Nach langem Studieren der Schemas der Netzgeräte (siehe Schema 2 und Schema 3 im Anhang) konnte ich dann die notwenigen Änderungen vornehmen.

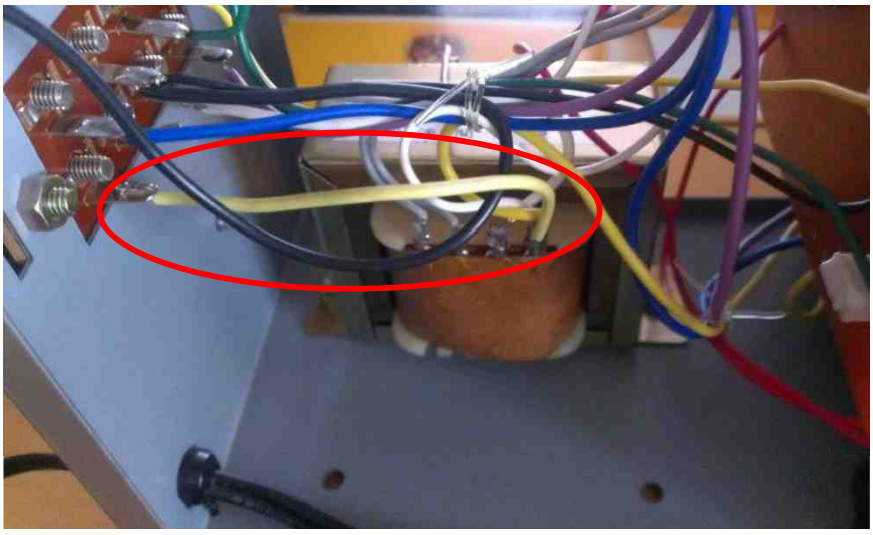
Abbildung 4: Mein Arbeitsplatz

Ich habe an den Steuereinheiten DAIWA MR-750 und DAIWA KR-500 folgende Änderungen vorgenommen:



Vom Trafo habe ich eine neue Verbindung an einen freien Pin (Ausgang) gelötet. Diese geht zur Relais-Karte und kommt wieder zurück, an den ‚normalen' Pin. Somit kann ich den Rotor mit dem Arduino, sowie auch manuell bedienen.

Abbildung 5: Umbau der MR-750 Steuereinheit



Auch bei der KR-500 Steuereinheit habe ich eine neue Verbindung zwischen Trafo und einem freien Pin gelötet.

Abbildung 6: Umbau der KR-500 Steuereinheit

4.3 Aufbau der Bodenstation

Für den Aufbau der Bodenstation haben wir uns an einem Samstag in den Ferien getroffen. Einige Amateurfunker vom Amateurfunkclub Sursee, Herr Wildfeuer, mein Vater und ich bauten die Antennen, Rotoren und Steuereinheiten der Bodenstation auf. Auch hier war wieder Innovationsgeist und Improvisation gefragt.

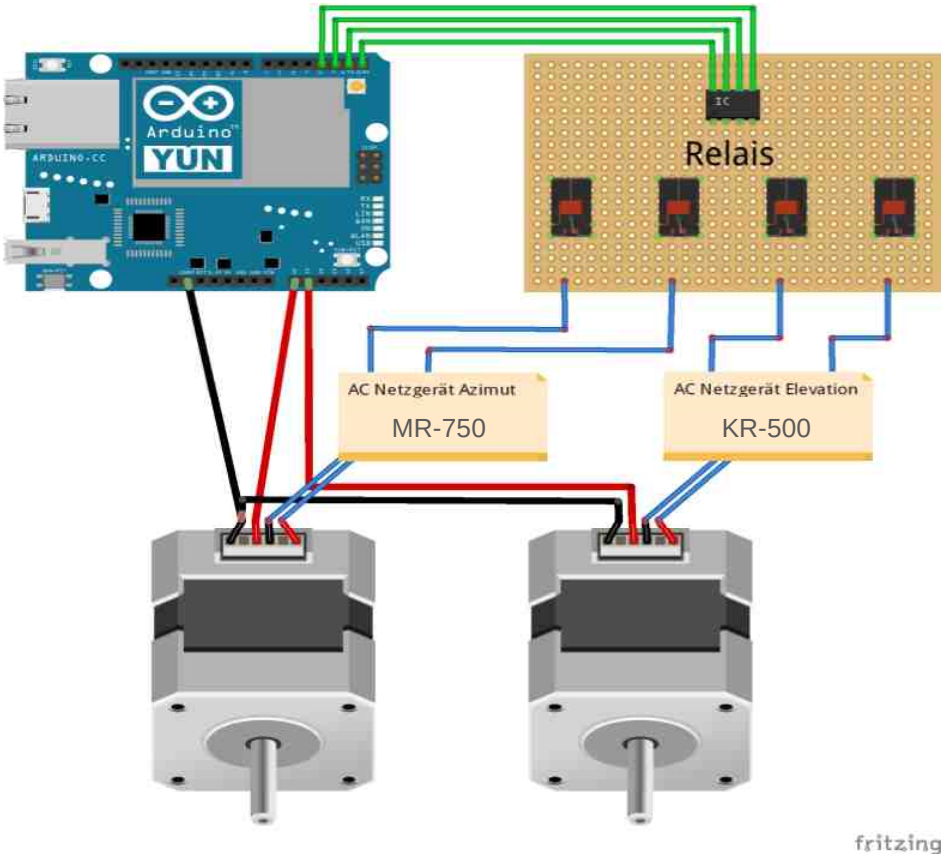
Abbildung 7: Casimir Schmid und ich bei der Montage der Rotoren und deren Verkabelung

Probleme zeigten sich bei der Ausrichtig nach Norden und der Verkabelung der Rotoren. Doch nach einigen Diskussionen konnten auch diese Probleme gelöst werden. Weiter stellte uns die Kabelführung vor weitere Herausforderungen, denn die hochsensiblen HF-Kabel durften nicht geknickt werden.

Abbildung 8: Die Antennen stehen und funktionieren

4.4 Schema und Aufbau

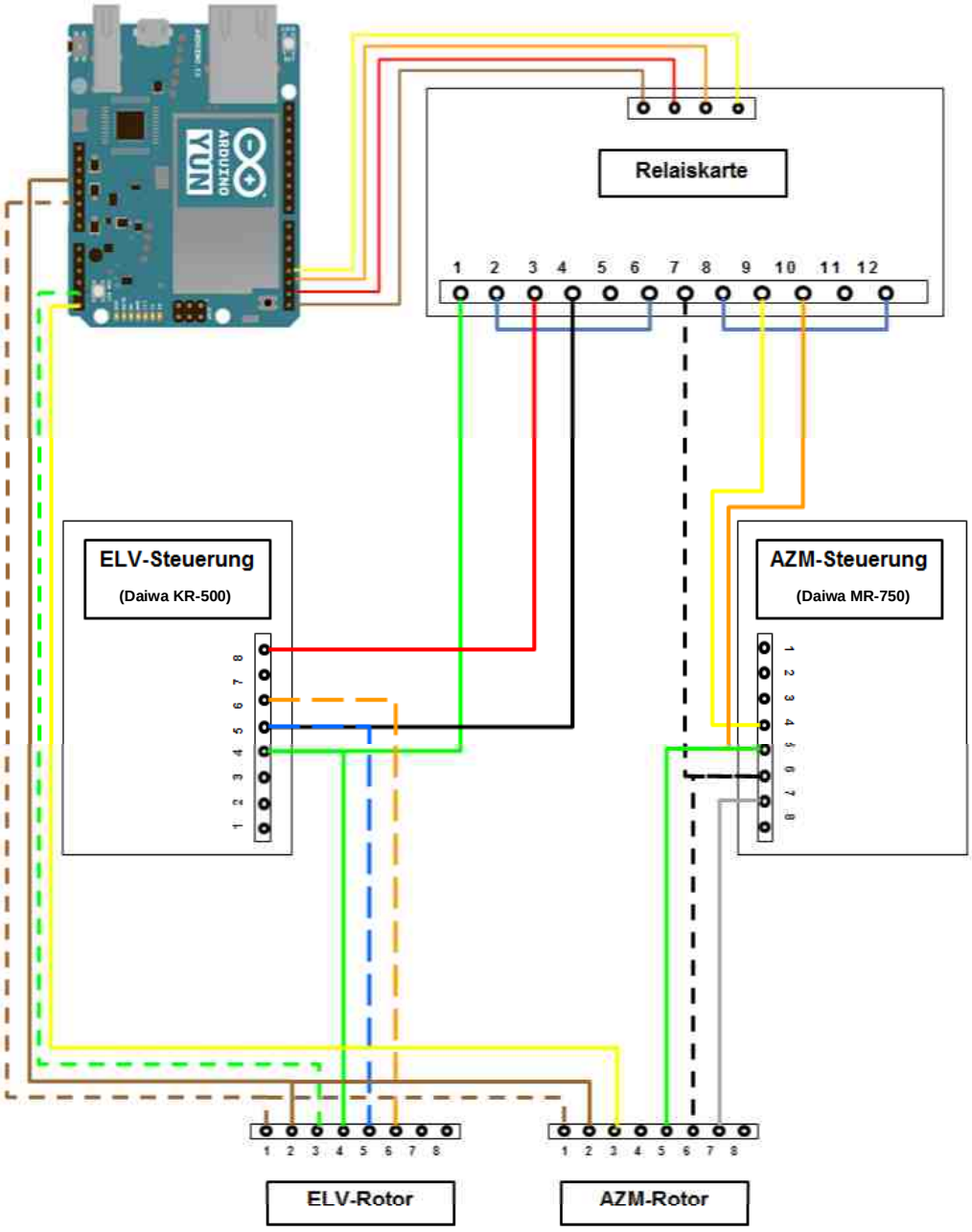
Der Arduino Yun ist durch Wlan ans Internet angebunden. Von dort bezieht er die Koordinaten vom Ballon. Vier digitale Outputs sind mit einer Relaiskarte verbunden, welche ich freundlicherweise von Casimir Schmid (HB9WBU) bekommen habe. Diese Relais schalten die abgegriffene AC-Spannung aus den beiden Steuergeräten. Für jeden Motor gibt es ein ‚Vor' und ‚Zurück' Signal. Die Relaiskarte wird mit 12V aus einem externen Netzgerät gespiesen, um die Schaltleistung für die Relais zu erreichen.



Grafik 3: Übersicht

4.4.1 Verkabelung

Die Verkabelung ist ein wenig kompliziert, und ich muss zugeben, dass ich leider, infolge fehlender Kabel, nicht immer nach dem richtigen Farbcode gearbeitet habe. Die vorhandenen Elevations- und Azimut Steuerungen (ELV- und AZM-Steuerung) dienen als Speisegeräte. Der Arduino schaltet über die Relaiskarte deren Ausgangspins und misst die Position der Rotoren.



Grafik 4: Verkabelung detailliert

Bemerkung:

Die Relaiskarte bekommt vom ELV-Netzgerät (Pin 8) und vom AZM-Netzgerät (Pin 4) die abgegriffene Spannung auf Pin 3 (rot) bzw. Pin 9 (gelb).

4.5 Arduino

Technische Spezifikationen siehe Kapitel 3.5. Da der Arduino ohne Gehäuse geliefert wird, habe ich eine Halterung mit dem 3D-Drucker ausgedruckt.

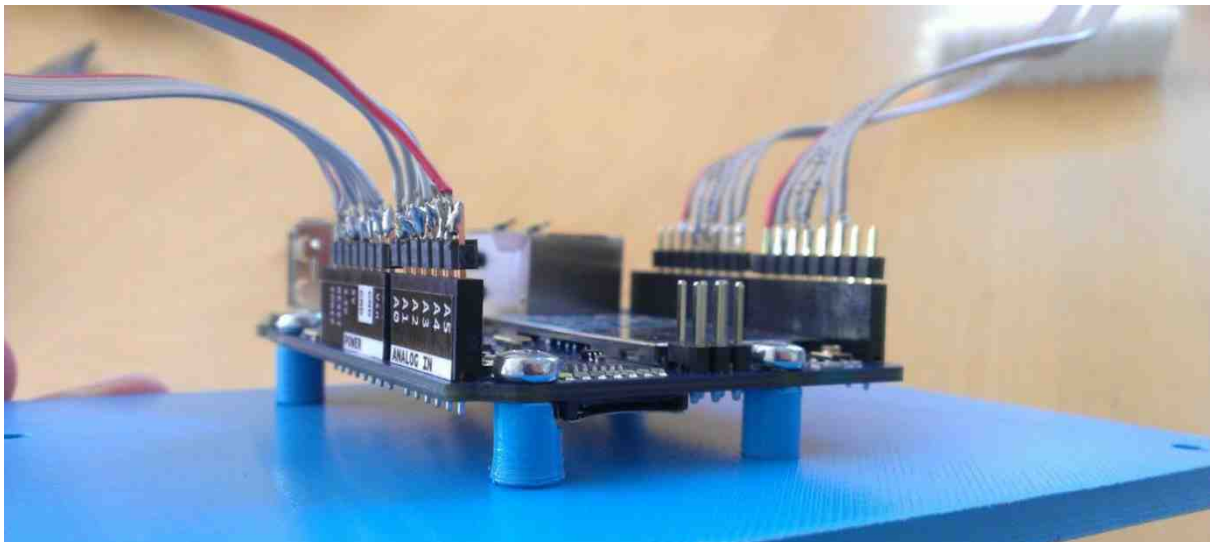


Abbildung 9: Arduino Yún mit Verkabelung

4.5.1 Warum Arduino?

Ich bin erst sehr spät auf Arduino gestossen und habe im Arduino Yún den perfekten Controller gefunden, da er eine 5V Referenzspannung für die Potentiometer in den Motoren hat. Zudem ist im Arduino Yún ein Linux Teil mit Internetanbindung über WLAN oder Ethernet vorhanden, welcher für die automatische Aktualisierung der GPS Koordinaten des Ballons notwendig ist. Ein weiterer sehr grosser Vorteil liegt in der vergleichsweise einfachen Programmierung und der guten Website mit vielen Beispielen und Anleitungen.

4.5.2 Internetanbindung

Man kann beim Arduino Yun über sein eigenes Wlan auf ein Web Panel zugreifen, auf welchem man ihn dann mit einem anderen Wlan Netzwerk verbinden kann.

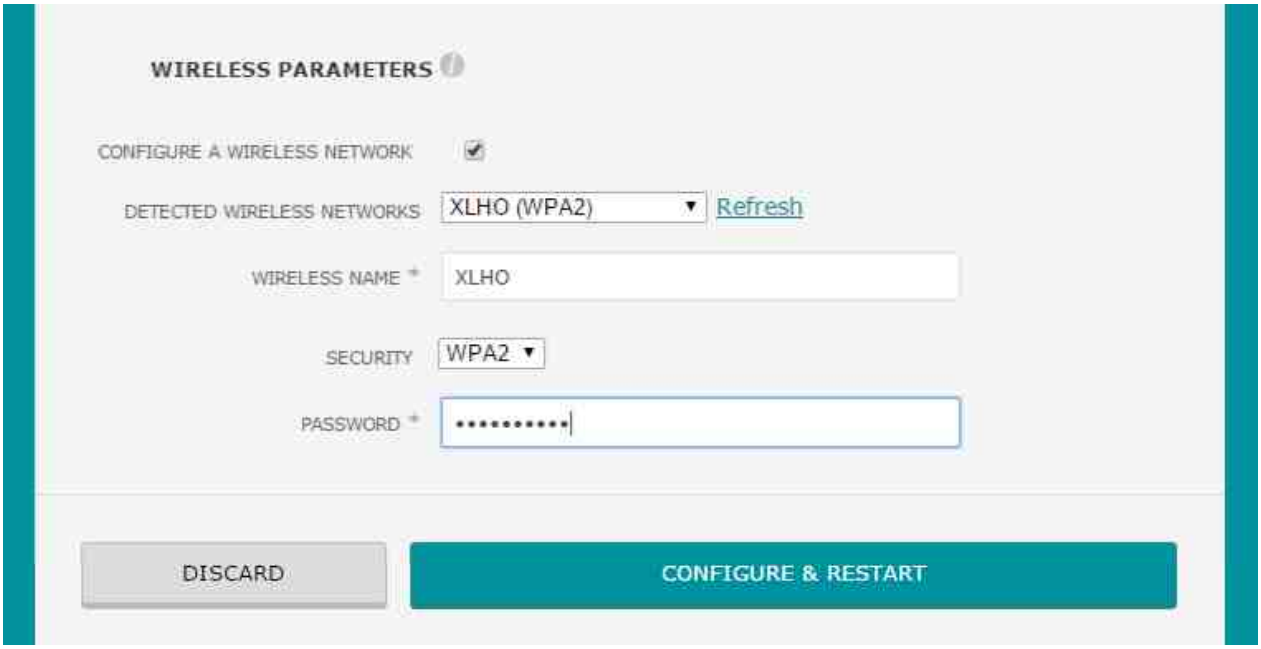


Abbildung 10: Wifi Konfiguration im Web Panel

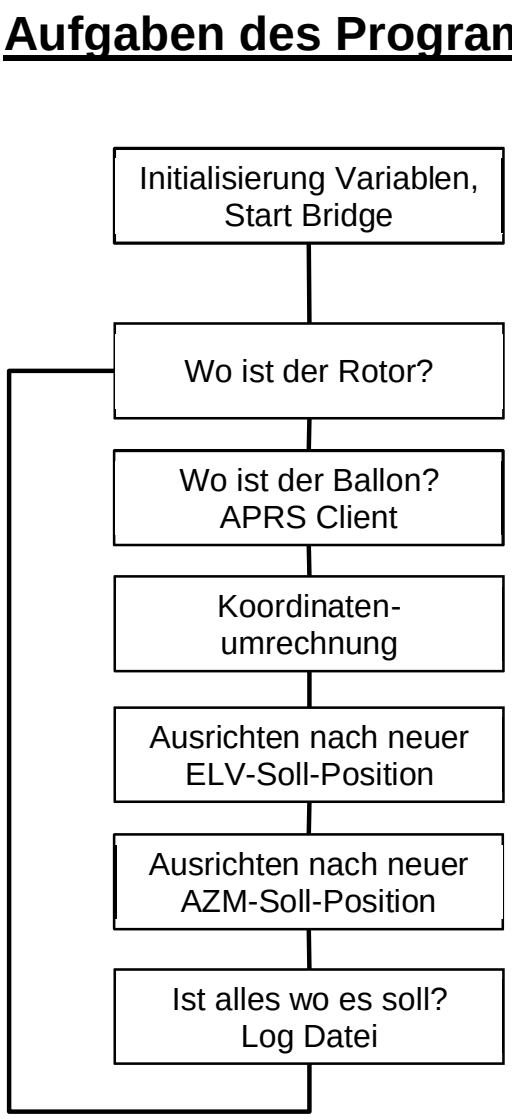
Ich habe mich dazu entschieden, den Arduino über einen mobilen Hotspot meines Handys mit dem Internet zu verbinden, da ich so sehr mobil bin und Eventualitäten wie zum Beispiel geschlossene Ports in Netzwerken (bei Schulen…) umgehen kann. Mein Handy stellt also einen WLAN-Hotspot zur Verfügung, der mit WPA2-Personal gesichert ist.

```
Current WiFi configuration
SSID: XLHO
Mode: Client
Signal: 74%
Encryption method: WPA2 PSK (CCMP)
Interface name: wlan0
Active for: 3 minutes
IP address: 192.168.43.162/255.255.255.0
MAC address: 90:A2:DA:F7:04:98
RX/TX: 56/75 KBs
```

Abbildung 11: Wifi Status

4.6 Programmierung

Die Programmierung war ziemlich nervenaufreibend und zeitintensiv, da ich noch so gut wie keine C-Kenntnisse hatte und mir die ganze Programmiersprache über Foren und der Arduino Referenz Website selbst beibringen musste.
Als erstes habe ich mir Gedanken darüber gemacht, was mein Programm alles können muss. So bin ich zur folgenden Grafik gekommen:



Grafik 5: Aufgaben des Programms

Nachfolgend beschreibe ich die einzelnen Programmteile zur Erklärung der Aufgaben des Programms.

4.6.1 Messung der IST-Position der Motoren

Die Messung der IST-Position wird durch eine Referenzspannung, welche vom Potentiometer im Motor linear zur Position bereitgestellt wird, gemessen. Der Arduino liefert eine Referenzspannung von 5V an das Potentiometer. Den Spannungswerten wird eine Position zugeordnet.
Folgender Programmteil misst die aktuelle Position der Antenne:

```
#define ELV90Value 84   //ELV90Value (winkel bei 700)
#define ELVValueLow 280
#define ELVValueHigh 700

#define AZMStopValueRef 228   // (Referenzsystem Winkel bei Stopp)
#define AZMStopValue 138   // (Reeller Winkel bei Stopp)
#define AZMValueLow 350    //Spannung im Westen
#define AZMValueHigh 912   //Spannung bei Stopp


// Azm "left" Relais (gegen 0 Volt)

//Inputs
#define sensorELV A2  // select the input pin for the potentiometer
#define sensorAzm A1

//Variablen Ist-Winkel
int ElvValue = 0;  // variable to store the value coming from the Poti
int AzmValue = 0;
int degElvN = 0;   // Ist-Winkel in Grad
int degAzmN = 0;

/* read the value from the sensor/Poti */
ElvValue = analogRead(sensorELV);
AzmValue = analogRead(sensorAzm);
degElvN = map(ElvValue, ELVValueLow, ELVValueHigh, 0, ELV90Value);
degAzmN = map(AzmValue, AZMValueLow, AZMValueHigh, 0, AZMStopValueRef);
if (degAzmN < 90) {
  degAzmN = degAzmN + 270;
}
else {
  degAzmN = degAzmN - 90;
}
Serial.println("\nMomentane Position der ROTOREN:  " );
Serial.print(" Elevation =  ");
Serial.print(degElvN);
if (DebugFlag == true) {
  Serial.print("  /  Spannung =  ");
  Serial.println(ElvValue);
}
Serial.print("   Azimut =  ");
Serial.print(degAzmN);
if (DebugFlag == true) {
  Serial.print("  /  Spannung =  ");
  Serial.print(AzmValue);
}
Serial.println("\n");
```

4.6.2 APRS Client

Das Programm, welches über das APRS Internet Protokoll die Koordinaten des Ballons empfängt, hat mich sehr herausgefordert. Im TCP/IP Bereich hatte ich bisher keine Kenntnisse, weshalb mir dieses Programm schwer gefallen ist. Zuerst habe ich mich über das APRS Protokoll informiert. Es gibt verschiedene Server, auf welchen die von I-Gates[3] eingespeisten Daten verfügbar gemacht werden. Ich gehe über den Server von HB9SDB in Zug, http://zug.aprs2.net:14501/ da dieser ziemlich zuverlässig ist.

Zuerst muss das Programm eine TCP-Verbindung mit dem Server auf Port*:14580* herstellen. Dann muss es ein Login senden, denn es können nur registrierte Nutzer sich direkt mit dem Server verbinden. Mit einem Filter kann man filtern, was man alles zugesendet haben möchte. Ich benutze den Filter *b/HB9AW-11,* so bekomme ich nur die Daten vom Ballon.

```
#include <Bridge.h>
#include <Process.h>
#include <YunClient.h>
boolean lastConnected = false;    // state of the connection last time through the
main loop
YunClient APRS;
IPAddress APRServer(91, 201, 57, 230);                    //zug.aprs2.net
#define PORT 14580                                                          //
Definiert Port bei zug.aprs2.net
                  "user HB9AW-12 pass 21985 vers KSRotor A0.2 filter b/HB9AW-11"
/* Definiert Login mit Benutzernamen, Passwort und Filter */

void Loop (){
/* Verbindung mit APRS server aufbauen. */
  if (!lastConnected) {
    Serial.println("Bitte warten...Verbindung wird aufgebaut.");
    if (APRS.connect(APRServer, PORT)) {
      lastConnected = true;
      Serial.println("APRS connected!...\n");
      delay(1000);
      while (APRS.available() > 0) {
        char c = APRS.read();
        Serial.print(c);
      }
      APRS.write((uint8_t *)LoginAPRS, sizeof(LoginAPRS));
    }
    else{Serial.println("Verbindung konnte nicht hergestellt werden!");}
    //!lastConected

  /* Read TCP Messages */
  char StringAPRS[512];
  readAPRS(StringAPRS); //Aufruf von Funktion readAPRS
  Serial.println(StringAPRS);
  }

int readAPRS(char * result){ //Funktion readAPRS
  int i = 0;
  result[i] = '\0';
  while (APRS.available() > 0 ) {
    char inChar = APRS.read();
    if (inChar == '\n'){
      result[i] = '\0';
      APRS.flush();
```

[3] I-Gate (Internet Gateway) = Empfangsstation mit Einspeisung der empfangenen Daten ins Internet

```
      result[i] = '\0';
      return 0;
    }
    if (inChar != '\r'){
      result[i] = inChar;
      i++;
      if (i == 333) {return 0;}
    }
  } //APRS.available
} //readAPRS
```

4.6.3 Filter / Formatierung

Der Filter hat mich ebenfalls gefordert, da ich so ein Programm noch nie zuvor gemacht hatte. Zum Glück hat mir Kevin etwas auf die Sprünge geholfen.

Für den Filter habe ich eine .h Datei gewählt, da ich den Filter so in normalem C schreiben konnte. Ein solches .h Modul kann dann per Aufruf von definierten Funktionen im normalen Programm integriert werden.

```
//Info.h
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct info {
    float north, east;
};

struct info loadData(char input[]) {
    struct info output;
    int stringLength = strlen(input);
    int it = 0;
    while(it < stringLength) {
        if(input[it] == 'h') {
            ++it;
            int jt = 0;
            while(input[++jt + it] != 'N') {}

            char northStr[jt+1];
            strncpy(northStr, input+it, jt );
            northStr[jt ] = '\0';
            output.north = atof(northStr);

            it += jt + 2; //Skip D
            jt = 0;
            while(input[++jt + it] != 'E') {}

            char eastStr[jt+1];
            strncpy(eastStr, input+it, jt);
            eastStr[jt ] = '\0';
            output.east = atof(eastStr);
        }
        ++it;
    }

    return output;
}
```

```
//Info2.h
#include <stdio.h>
#include <string.h>
#include <stdlib.h>


struct info2 {
    float north, east, height;
};

struct info2 loadData2(char input[]) {
    struct info2 output;
    int stringLength = strlen(input);
    int it = 0;
    while(it < stringLength) {
        if(input[it] == '=') {
            ++it;
            int jt = 0;
            while(input[++jt + it] != 'H') {}


            char heightStr[jt];
            strncpy(heightStr, input+it, jt - 1);
            heightStr[jt - 1] = '\0';
            output.height = atof(heightStr);

        }
        ++it;

    }

    return output;
}
```

4.6.4 Koordinatenumrechnung

Für die Koordinatenumrechnung von WGS84 zu Schweizer Projektionskoordinaten habe ich ebenfalls eine .h Datei geschrieben.

```
//  WGStoCH.h
//  Umrechnung WGS84 zu CH1903

#include <math.h>

#define t1  169028.66
#define t2  26782.5

struct CH {
    float north, east;
};


struct CH loadData3(float WGSlat, float WGSlong) {
    struct CH output;

    // 1. in [''] umwandleln
    WGSlat = WGSlat * 3600;
    WGSlong = WGSlong * 3600;

    // 2. Hilfsgrössen [φ', λ'] ausrechnen
```

```
    float d1 = WGSlat - t1;
    float d2 = WGSlong - t2;

    float WGSlat2 = d1 / 10000;
    float WGSlong2 = d2 / 10000;

    // 3. CH1903 Koordinaten in Meter ausrechnen
    float X = 600072.37 + 211455.93  * WGSlong2 -  10938.51  * WGSlong2  * WGSlat2
-  0.36  * WGSlong2  * pow(WGSlat2, 2)  -  44.54  * pow(WGSlong2, 3);
    float Y = 200147.07 + 308807.95 * WGSlat2 + 3745.25 * pow(WGSlong2, 2) + 76.63
* pow(WGSlat2, 2) - 194.56 * pow(WGSlong2, 2) * WGSlat2 + 119.79 * pow(WGSlat2,
3);

    output.east = X;
    output.north = Y;

    return output;
}
```

4.6.5 Motorensteuerung

Die Ansteuerung der Motoren per Relais habe ich als void-Funktion programmiert.

```
/* Modul: ELV Motor auf Position bringen */
void elvMOVE(int SOLL)
{
    while(degElvN < SOLL){
      digitalWrite(elvUP, HIGH);
      ElvValue = analogRead(sensorELV);
      degElvN = map(ElvValue, ELVValueLow, ELVValueHigh, 0, ELV90Value);

      if (DebugFlag==true) { Serial.println(degElvN);}
      delay(AnalogREAD);
    }
    digitalWrite(elvUP, LOW);
    while(degElvN > SOLL){
      digitalWrite(elvDOWN, HIGH);
      ElvValue = analogRead(sensorELV);
      degElvN = map(ElvValue, ELVValueLow, ELVValueHigh, 0, ELV90Value);
      delay(AnalogREAD);
    }
    digitalWrite(elvDOWN, LOW);
} //finish elvMOVE

/* Modul: AZM Motor auf Position bringen */
void azmMOVE(int SOLL)
{
  if((SOLL > 0) && (SOLL < 135)){
    while(degAzmN < SOLL){
      digitalWrite(azmR, HIGH);
      AzmValue = analogRead(sensorAzm);
      degAzmN = map(AzmValue, AZMValueLow, AZMValueHigh, 0, AZMStopValueRef);
  if (degAzmN < 90){
    degAzmN = degAzmN + 270;
    }
    else{
      degAzmN = degAzmN - 90;
    }
            delay(AnalogREAD);
    }
```

```
    digitalWrite(azmR, LOW);

    while(degAzmN > SOLL){
      digitalWrite(azmL, HIGH);
      AzmValue = analogRead(sensorAzm);
      degAzmN = map(AzmValue, AZMValueLow, AZMValueHigh, 0, AZMStopValueRef);
  if (degAzmN < 90){
    degAzmN = degAzmN + 270;
    }
    else{
      degAzmN = degAzmN - 90;
    }
      delay(AnalogREAD);
    }
    digitalWrite(azmL, LOW);
  }
  if((SOLL > 270)&&(SOLL < 360)){
        while(degAzmN < SOLL){
      digitalWrite(azmR, HIGH);
      AzmValue = analogRead(sensorAzm);
      degAzmN = map(AzmValue, AZMValueLow, AZMValueHigh, 0, AZMStopValueRef);
  if (degAzmN < 90){
    degAzmN = degAzmN + 270;
    }
    else{
      degAzmN = degAzmN - 90;
    }
            delay(AnalogREAD);
    }
     digitalWrite(azmR, LOW);

    while(degAzmN > SOLL){
      digitalWrite(azmL, HIGH);
      AzmValue = analogRead(sensorAzm);
      degAzmN = map(AzmValue, AZMValueLow, AZMValueHigh, 0, AZMStopValueRef);
  if (degAzmN < 90){
    degAzmN = degAzmN + 270;
    }
    else{
      degAzmN = degAzmN - 90;
    }
      delay(AnalogREAD);
    }
    digitalWrite(azmL, LOW);
  }
} //finish azmMOVE
```

4.6.6 Log Datei / Fehlerprotokoll

Die Log Datei war nicht so schwierig zu programmieren, da es dazu ein gutes Beispiel aus dem Arduino Forum gibt. Dieses Programm soll alle Messdaten auf eine SD-Karte aufgezeichnen.

```
/* Data logger */
    // make a string that start with a timestamp for assembling the data to log:
    String dataString;
    dataString += getTimeStamp();
    dataString += " =  ";
    dataString += String(infoData1.north);
    dataString += ",";
    dataString += String(infoData1.east);
    dataString += ",";
    dataString += String(infoData2.height);
    dataString += ",";
    dataString += String(CHdata.east);
    dataString += ",";
    dataString += String(CHdata.north);
    dataString += ",";
    dataString += String(elvSOLL);
    dataString += ",";
    dataString += String(azmSOLL);
    dataString += " : ";
    dataString += String(degElvN);
    dataString += ",";
    dataString += String(degAzmN);
    dataString += " : ";


    // open the file. note that only one file can be open at a time,
    // so you have to close this one before opening another.
    // The FileSystem card is mounted at the following "/mnt/FileSystema1"
    File dataFile = FileSystem.open("/mnt/sd/datalog.txt", FILE_APPEND);

    // if the file is available, write to it:
    if (dataFile) {
      dataFile.println(dataString);
      dataFile.close();
      // print to the serial port too:
      Serial.println(dataString);
    }
    // if the file isn't open, pop up an error:
    else {
      Serial.println("error opening datalog.txt");
    }
```

5 Ergebnisse und Diskussion

5.1 Ballon Start am 15. Oktober 2014

Am 15. Oktober 2014 liessen wir einen Stratosphärenballon starten, ausgestattet mit einem Arduino, der Luftdruck, Temperatur und Luftfeuchtigkeit messen konnte, einer GPS-Einheit, die ein APRS-Signal sendet und einem Rasberry Pi, der per DVBT-Protokoll ein HD-Live-Videosignal auf der Frequenz 436.5 MHz an die Bodenstation senden sollte.

Der Start wurde zuerst wegen schlechtem Wetter um einen Tag verlegt. Doch zum Glück war dies ein richtiger Entscheid, denn am nächsten Tag war das Wetter wunderschön. Die Simulation für die Flugbahn des Ballons versprach eine günstige Flugbahn.

Meine Steuerung war zu diesem Zeitpunkt noch nicht ganz fertig. Ich sollte als Operator der Bodenstation die aktuellen Koordinaten des Ballons per Seriellem Monitor selbst eingeben. Der automatische Download der APRS Daten funktionierte leider noch nicht. Die Interaktion mit den Rotoren lief einwandfrei.

Die GPS-Einheit des Ballons fiel kurz vor dem Start aus, doch Herr Wildfeuer entschied sich, den Ballon trotzdem zu starten, glücklicherweise, denn sonst hätten wir viele schöne Bilder aus der Stratosphäre verpasst. Ohne GPS konnte ich mit meiner Antenne natürlich nicht mehr so genau die Verbindung aufrechthalten. Ich musste also improvisieren. Die ersten paar hundert Meter konnte ich das Videosignal noch per Rundstrahlantenne empfangen, doch dann wurde das Signal schwächer und ich schaltete auf die Helix-Antenne um. Am Anfang konnte ich die Antenne noch manuell per analogem Steuergerät auf Sichtkontakt ausrichten, aber irgendwann war der Ballon zu hoch und ich konnte ihn nicht mehr sehen. Ab da musste ich mit Hilfe (1) der letzten gesehenen Position, (2) der Zeit seit dem Start, (3) aktueller Windrichtung, (4) Geschwindigkeit des Aufstiegs und (5) der simulierten Flugbahn etwas improvisieren. Notizblock, Bleistift und Taschenrechner mussten schnell her! Zum Glück konnte ich auf verschiendene Programme zurückgreifen, mit welchen ich z.B. nur die Elevations und Azimut Winkel eingeben konnte, um so die Rotoren nachzuführen. Ein schwächer werdendes Signal war leider nur über die empfangenen Packets der Videoübertragung auszumachen. Das Ganze ging ziemlich gut, und ich konnte die Übertragung erstaunlich lange mit der simulierten Flugbahn und der verstrichenen Zeit aufrechhalten – zum Glück – denn die SD-Karte im Ballon, welche das Video aufnehmen sollte, war nach wenigen Minuten schon voll (aufgrund falsch berechneter Partitionierung).

Ich konnte am Fernseher bei meiner Bodenstation über 1:52 Stunden Live und in HD (!) verfolgen, was die Kamera im Ballon aufzeichnete – bis ich das Signal verlor.

Das phänomenale Bild der Erde: von Bodenoberfläche, über Troposphäre, bis Stratosphäre, und sogar der Blick ins dunkle Weltall, war sehr erstaunlich, im wahrsten Sinne des Wortes *der Höhepunkt meiner Arbeit*.

Abbildung 12: Screenshot Live-Video

5.2 Bodenstation beim Start

Obwohl der automatische Download am 15. Oktober 2014 noch nicht funktionierte, bin ich sehr zufrieden mit dem damaligen Stand meiner Arbeit. Der ganze Aufbau der Station und der Antennen wie auch der Betrieb hat mir viel Spass gemacht.

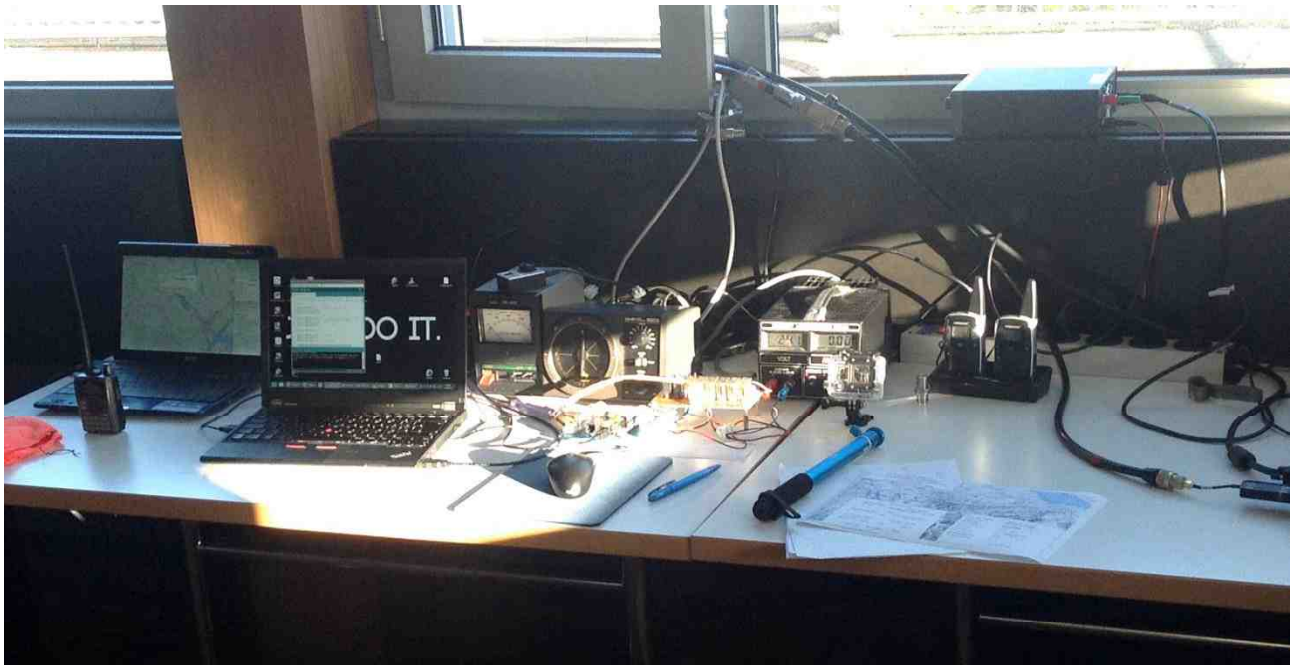


Abbildung 13: Bodenstation / Kontrollzentrum

5.3 Medienpräsenz

Der Start des Stratosphärenballons sorgte in der Region Sursee für Aufmerksamkeit. Sogar das Fernsehen Tele M1 war vor Ort und berichtete am selben Abend über das Projekt. Weiter gab es Artikel in der Neuen Luzerner Zeitung und der Surseer Woche, welche das Geschehen und das Engagement von Physiklehrer C. Wildfeuer porträtierten.

Auch auf YouTube ist der Flug des Ballons zu finden. Es gibt dort verschiedene Videos vom Start und die ganze Aufzeichnung der Bordkamera. Der Link zum Video ist im Anhang.

Abbildung 14: Interview mit Tele M1

6 Reflexion

6.1 Die Arbeit

Die Arbeit behandelt ein sehr interessantes, breites und spannendes Thema! Dies hat mir sehr geholfen immer wieder am Ball zu bleiben. Die Herausforderungen beim Löten und Basteln und dann nochmal beim Programmieren und Rechnen, haben mich sehr gefordert.

Ich habe sehr früh angefangen mit meiner Arbeit, da Herr Wildfeuer bereits vor den Sommerferien 2014 auf mich zugekommen ist und mich gefragt hat, ob ich für seinen Ballon-Start im Herbst eine Steuerung für die Antenne machen könne. Ich fand die Idee super und hab mich gleich an den Notizblock gesetzt. In der Zeit von Sommer bis Herbst habe ich dann sehr intensiv nach Lösungen gesucht und habe dann in den Herbstferien eine funktionierende Steuerung auf die Beine gestellt. Zu diesem Zeitpunkt musste ich die Koordinaten des Ballons noch selbst eingeben, per Seriellem Monitor. Das Ziel war ja, dass der Arduino die Koordinaten irgendwann mal selbst herunterlädt, doch nach dem mehr oder weniger geglückten Start machte ich eine grössere Pause. Ich würde dies nicht unbedingt weiter empfehlen, da man es so viel schwerer hat, wieder anzufangen. Ausserdem hatte ich in der Zeit vor dem Ballon-Start – wo ich die ganze Hardware gemacht habe – noch nicht so viele Kenntnisse über die Vorgehensweise mit einer solchen Arbeit. Einen Notizblock mit vielen Skizzen und Ideen hatte ich zwar immer dabei, doch ein Laborbuch wie ich es mir später zulegte wäre besser gewesen und hätte alles an einen Ort zusammen geführt. So musste ich aus einem Stapel Papier alle Verkabelungen und Einstellungen zusammensuchen. Dies ist ein weiterer Punkt, den ich allen empfehlen würde, die schon früh mit ihrer Maturaarbeit anfangen: Benutzt von Anfang an ein Laborbuch!

Das Ergebnis meiner Arbeit, sind die vielen Programme, welche jetzt auch automatisch die Rotoren nachführen können und die Hardware, welche das Ganze technisch ermöglicht.

Insgesamt bin ich sehr zufrieden mit meiner Arbeit, ich hatte interessante und abwechslungsreiche Probleme zu lösen und habe meine Ziele erreicht. Die Arbeit hat mich zudem motiviert, eine HB9 Amateur Lizenz zu machen und hat auch Einfluss bei meiner zukünftigen Studienwahl zum Wirtschaftsingenieur oder Elektroingenieur.☺

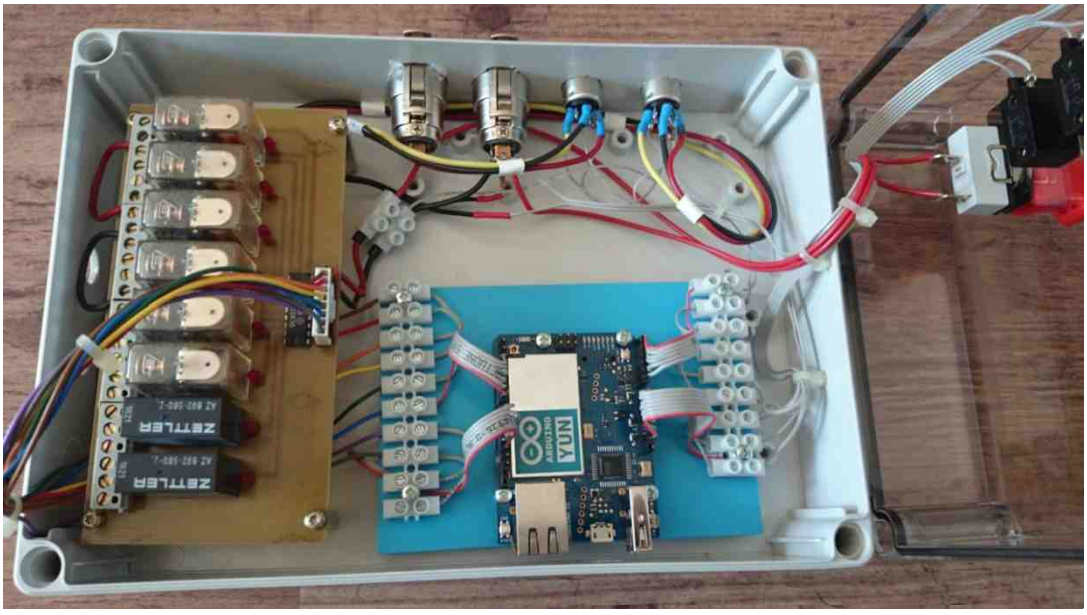


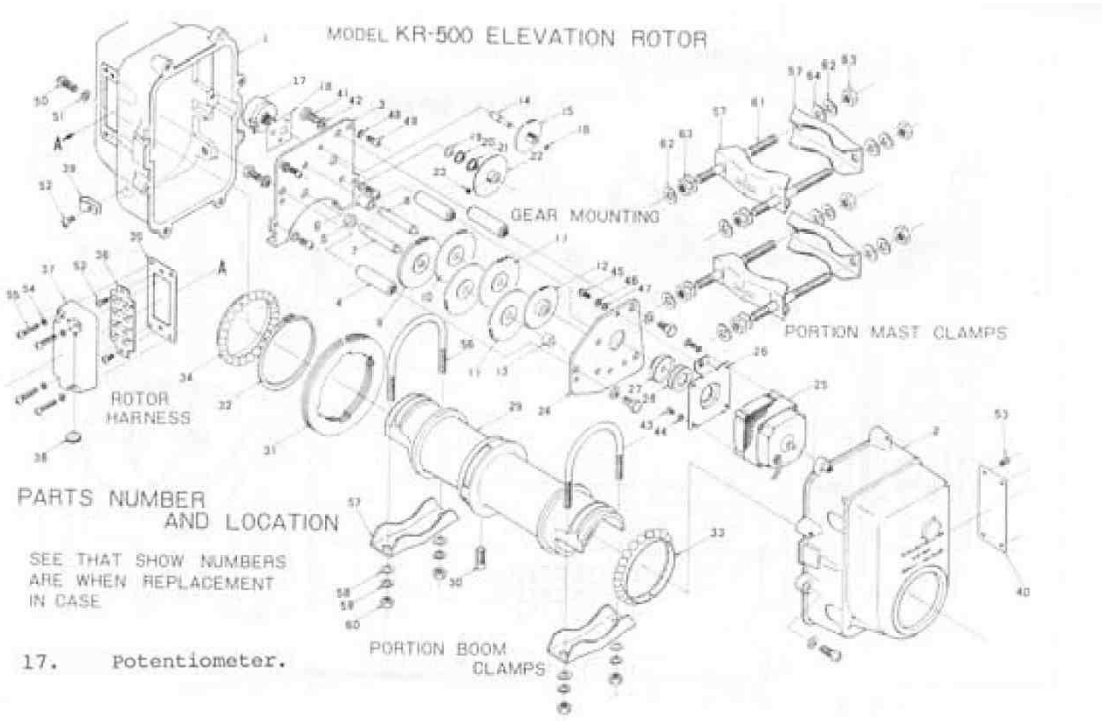
Abbildung 15: Endprodukt Gehäuse

7 Anhang

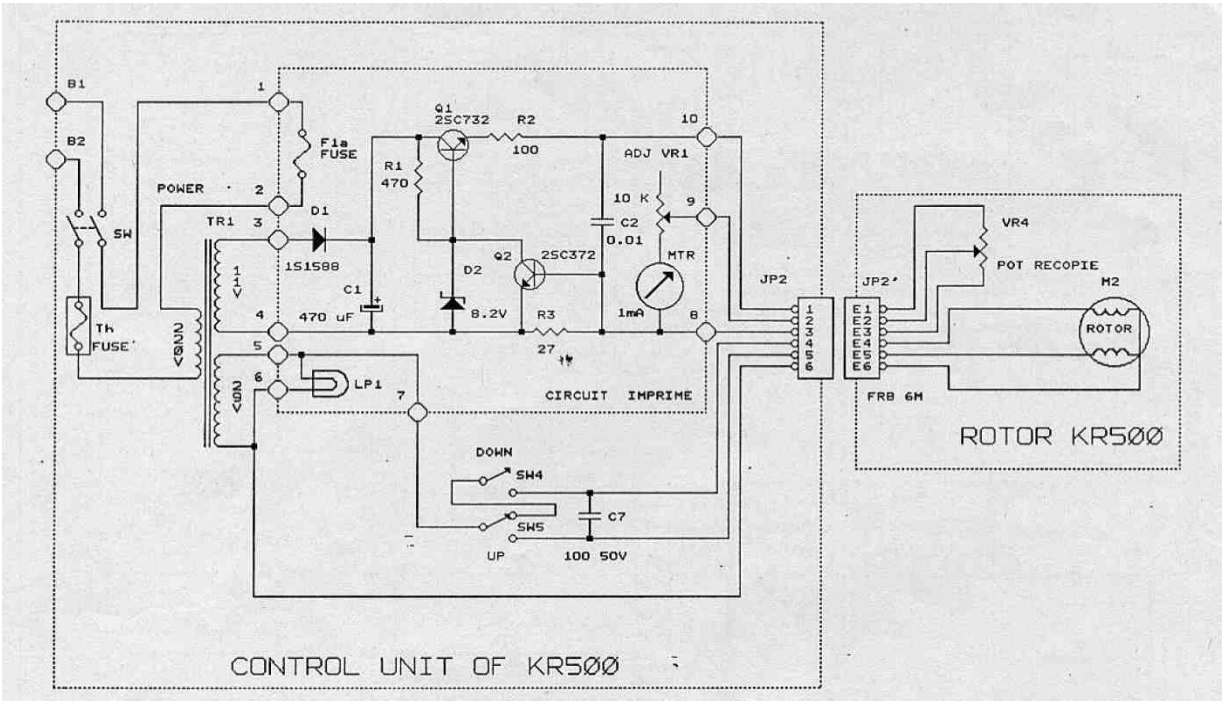
Hier werden alle gebrauchten Schemas und der finale Programm-Code aufgelistet.

7.1 Schemas

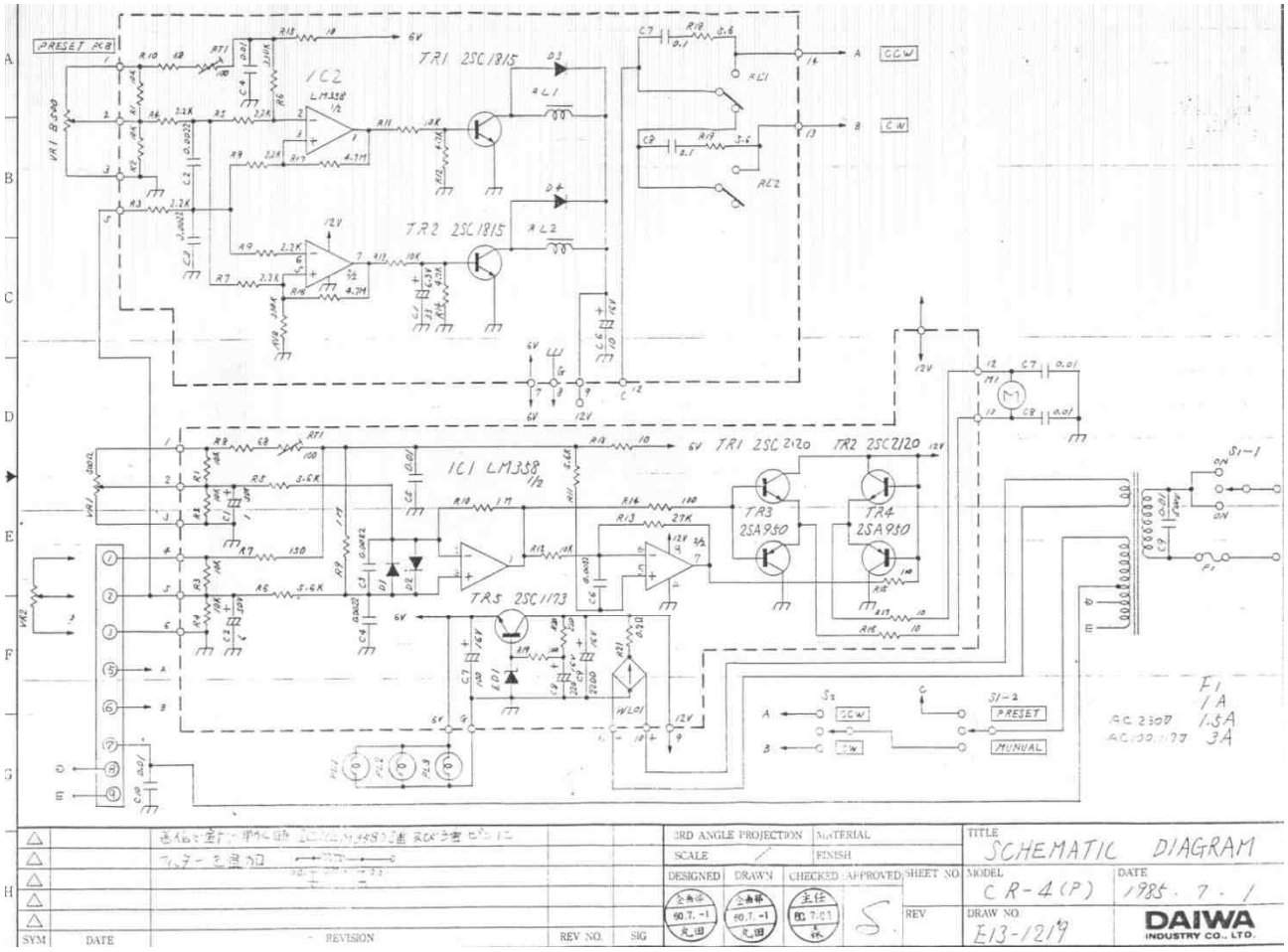
7.1.1 Motoren Dokument



Schema 1: DAIWA KR-500 Rotor Aufbau

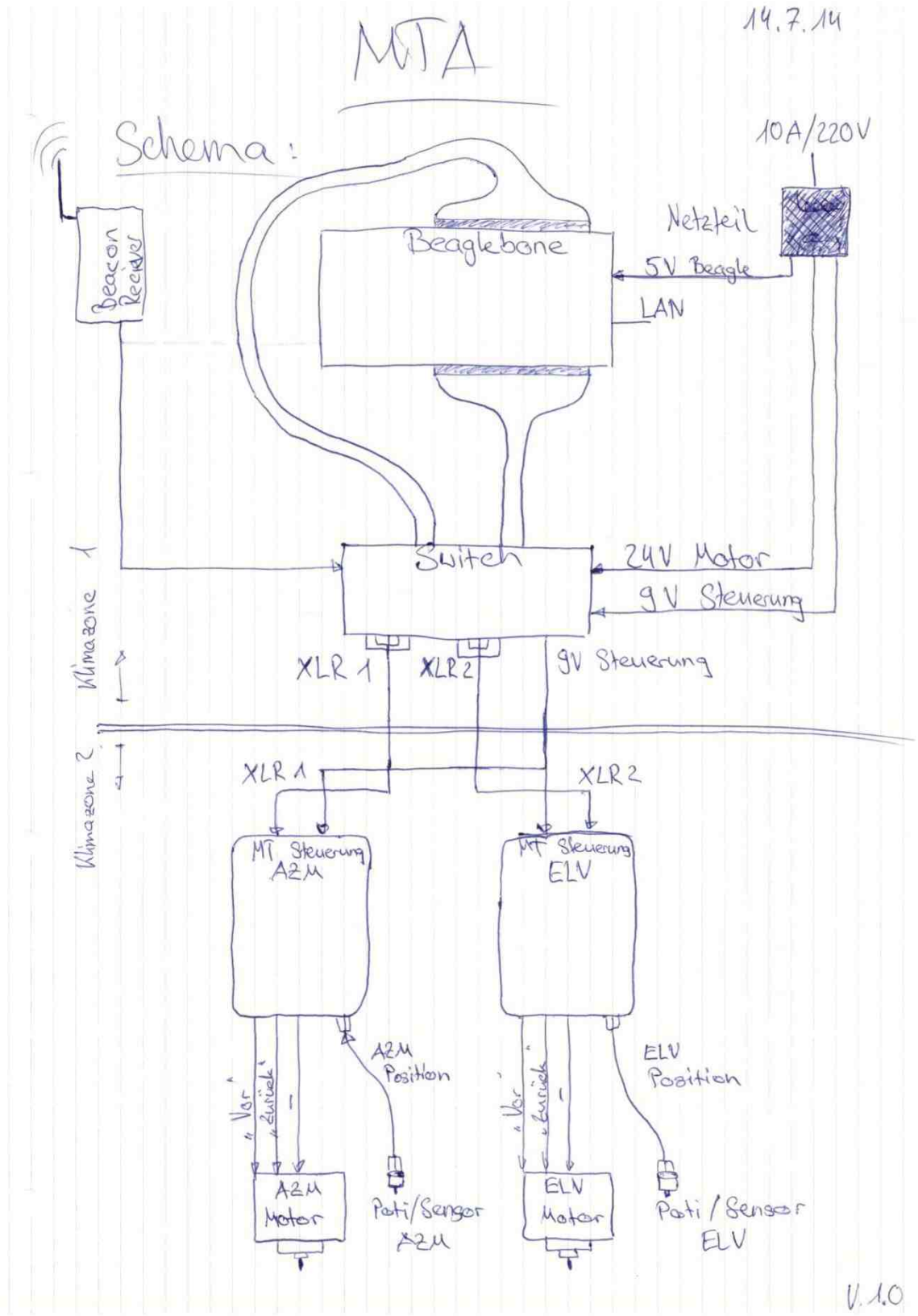


Schema 2: DAIWA KR-500 Schema

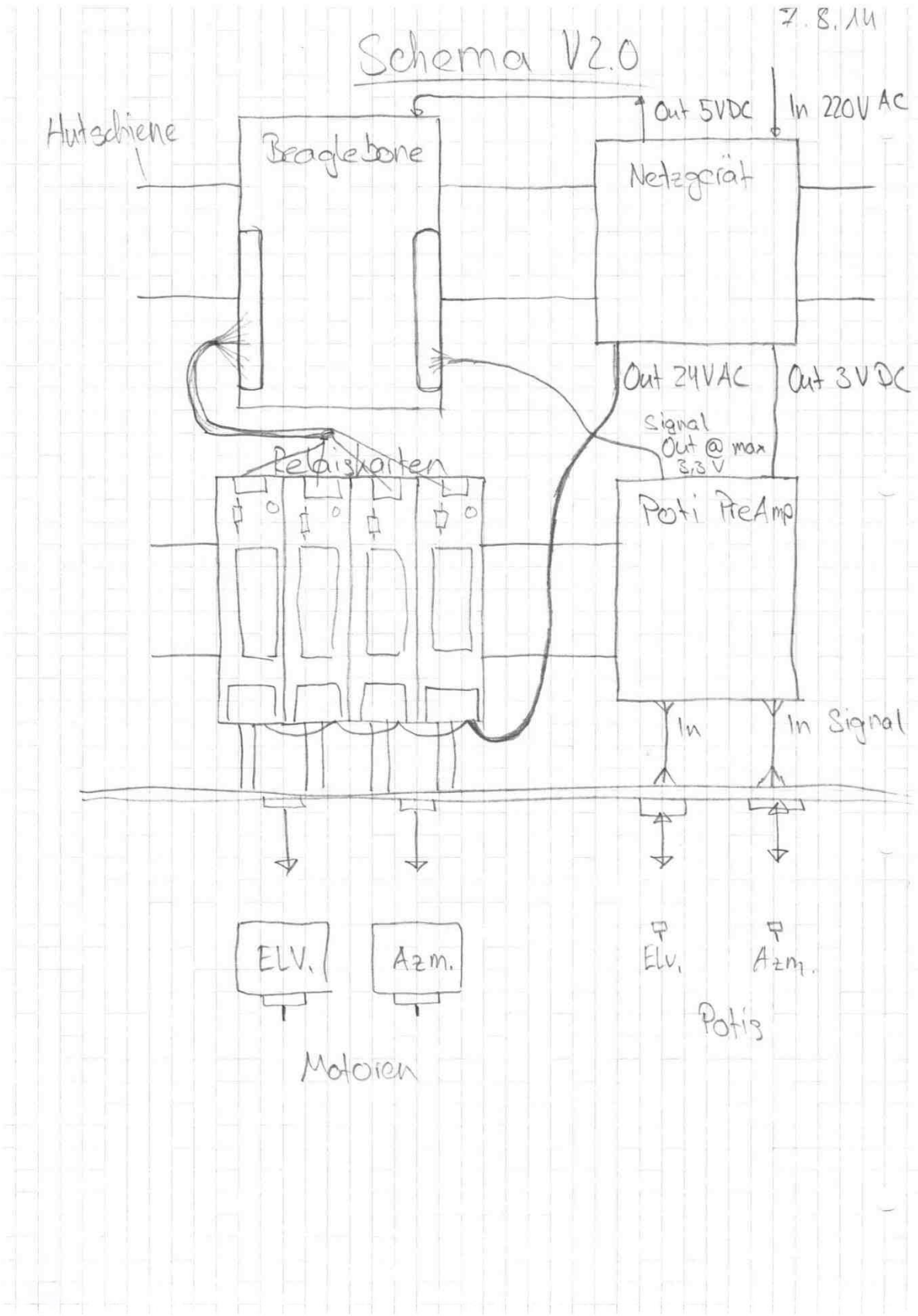


Schema 3: DAIWA MR-750e

7.1.2 Ideen und Prototypen



Schema 4: Idee mit C-Control



Schema 5: Idee mit Beaglebone

7.2 Links

Webseiten, die ich viel verwendet habe oder relevant für meine Arbeit waren, werden hier aufgelistet:

- http://aprs.fi/
  APRS Tracking Website. (Stand 29.09.15)
- https://www.arduino.cc/en/Reference/HomePage
  Arduino Referenz-Webseite für die Programmiersprache. (Stand 29.09.15)
- http://www.swisstopo.admin.ch/internet/swisstopo/de/home/apps/calc/navref.html
  Umrechnung von verschiedenen Koordinaten-Formaten. (Stand 29.09.15)
- https://map.geo.admin.ch/
  Online Karte der Schweiz. Zur Bestimmung von Standorten. (Stand 29.09.15)
- https://youtu.be/zID856nL-Vo
  Aufzeichnung des übertragenen Videos vom 15.Oktober 14. (Stand 29.09.15)
- https://youtu.be/4-UffhkyQ4U
  Beitrag von Tele M1 über den Ballonstart vom 15.Oktober 14. (Stand 29.09.15)

7.3 Source Code Masterprogramm

Das Masterprogramm ist die Finale Edition mit allen vorhergehenden Komponenten.

Es besitzt drei zusätzliche .h Dateien. Diese sind am Schluss aufgeführt.

```
/* Programm ist eine Final Edition zur Steuerung der Motoren.
-Get APRS String
-Filter APRS String
-Formatiere Koordinaten
-Umrechnung Koordinaten
-Umrechnung Koordinaten in Winkel
-Fahren der Rotoren mit SOLL Wert
*/

#include <Bridge.h>
#include <Process.h>
#include <YunClient.h>
#include "Info.h"
#include "Info2.h"
#include "WGStoCH.h"
#include <math.h>
#include <FileIO.h>

//Referenzsystem Initialisierung / Definitionen

#define ELV90Value 84   //ELV90Value (winkel bei 700)
#define ELVValueLow 280
#define ELVValueHigh 700

#define AZMStopValueRef 228   // (Referenzsystem Winkel bei Stopp)
#define AZMStopValue 138   // (Reeller Winkel bei Stopp)
#define AZMValueLow 350
#define AZMValueHigh 912

//Variablen IST-Winkel
  int ElvValue = 0;  // variable to store the value coming from the Poti
  int AzmValue = 0;
  int degElvN = 0;   // Ist-Winkel in Grad
  int degAzmN = 0;

//Variablen SOLL-Winkel
int degElvS = 0; //Soll-Winkel in Grad
int degAzmS = 0;
float HighS = 0; //relle Höhe (Differenz)

// Höhen
#define myHeight 518 // Meine Höhe in Meter über Meer 498 + 20m Gebäude

//Meine Koordinaten
//CH1903 / LV03650'646.9, 225'317.2
#define myLat 47.17688 //North
#define myLong 8.10675 //East
#define myEast 650646.9 //[m]
#define myNorth 225317.2 //[m]


//Berechnungs Konstanten
#define pi 3.14159265359  //Das ist Pi :)
#define ft 0.3048         //Fuss in Meter
#define t1  169028.66
#define t2  26782.5
```

```
#define INTERVAL  500 // Loop wait for 1000 = 1 sec
#define AnalogREAD 200
#define DebugFlag true // true or false
#define pollingInterval 5000  // TCP Polling
boolean lastConnected = false;    // state of the connection last time through the
main loop
unsigned long lastConnectionTime = 0;   // last time you connected to the server,
in milliseconds

//Outputs
#define ledPin 13   // select the pin for the LED
#define elvUP 2     // Elv Up Relais (gegen 5 Volt)
#define elvDOWN 3   // Elv Down Relais (gegen 0 Volt)
#define azmR 5      // Azm "right" Relais (gegen 5 Volt)
#define azmL 4      // Azm "left" Relais (gegen 0 Volt)

//Inputs
#define sensorELV A2  // select the input pin for the potentiometer
#define sensorAzm A1

/*Verbindung mit APRS Server definieren*/
YunClient APRS;
IPAddress APRServer(91, 201, 57, 230); //zug.aprs2.net
//IPAddress APRServer(10, 10, 43, 227);
#define PORT 14580
//Anmeldungsstring mit Benutzername, Passwort und Filter
                  "user HB9AW-12 pass 21985 vers KSRotor A0.2 filter b/HB9AW-11\r"
//#define LoginAPRS "user HB9AW-12 pass 21985 vers KSRotor A0.2 filter b/HB9AW-
11\r"

/*  Funktionen deklaration  */
float formatnorth (float);
float formateast (float);
float formatheight (float);



void setup() {
  //Definition der Outputs
  pinMode(ledPin, OUTPUT);
  pinMode(elvUP, OUTPUT);     //Elv Up
  pinMode(elvDOWN, OUTPUT);   //Elv Down
  pinMode(azmR, OUTPUT);      //Azm "right"
  pinMode(azmL, OUTPUT);      //Azm "left"

  //Initialisierung der seriellen Schnittstelle
  Serial.begin(9600);
  Bridge.begin();
  while (!Serial);  // Wait until a Serial Monitor is connected.
  digitalWrite(ledPin, HIGH);
  Serial.println("Programm startet... WIFI Check...");

  Process wifiCheck;  // Check Wifi Connection
  delay(pollingInterval);
  wifiCheck.runShellCommand("/usr/bin/pretty-wifi-info.lua");
  // while there's any characters coming back from the
  // process, print them to the serial monitor:
  while (wifiCheck.available() > 0) {
    char c = wifiCheck.read();
```

```
    Serial.print(c);
  }
  Serial.println();
  Serial.println("Setup abgeschlossen. Programm gestartet...\n");
}

void loop() {

  /* Messung der momentanen Position der Rotoren */
  ElvValue = analogRead(sensorELV);
  AzmValue = analogRead(sensorAzm);
  degElvN = map(ElvValue, ELVValueLow, ELVValueHigh, 0, ELV90Value);
  degAzmN = map(AzmValue, AZMValueLow, AZMValueHigh, 0, AZMStopValueRef);

 // Ausgleich für Norden
 if (degAzmN < 90){
   degAzmN = degAzmN + 270;}

  else{
   degAzmN = degAzmN - 90;
   }

  Serial.println("\nMomentane Position der ROTOREN:  " );
  Serial.print(" Elevation =  ");
  Serial.print(degElvN);
  Serial.print("   Azimut =  ");
  Serial.print(degAzmN);



  /* Verbindung mit APRS server aufbauen. */
  if (!lastConnected) {
    Serial.println("Bitte warten...Verbindung wird aufgebaut.");
    if (APRS.connect(APRServer, PORT)) {
      lastConnected = true;
      Serial.println("APRS connected!...\n");
      delay(1000);
      while (APRS.available() > 0) {
        char c = APRS.read();
        Serial.print(c);
      }
      APRS.write((uint8_t *)LoginAPRS, sizeof(LoginAPRS));
      if (DebugFlag == true) {
        Serial.println(LoginAPRS);
      }
    }
    else
    {
      Serial.println("Verbindung konnte nicht hergestellt werden!");
    }
    //!lastConected

  /* Read TCP Messages */
  char StringAPRS[512];
  readAPRS(StringAPRS); //Aufruf von Funktion readAPRS
  Serial.println(StringAPRS);

  // Filter der Koordinaten aus dem String
  info infoData1 = loadData(StringAPRS);
  info2 infoData2 = loadData2(StringAPRS);
```

```
/*Wenn Ballon sendet fürhre Formatierung etc aus*/

if (infoData1.north > 1) {
  // Ausgabe der ausgelesenen Koordinaten
  Serial.print("ausgelesene Koordinaten: ");
  Serial.println(infoData1.north);
  Serial.print("ausgelesene Koordinaten: ");
  Serial.println(infoData1.east);
  Serial.print("ausgelesene Hohe: ");
  Serial.println(infoData2.height);

  /*Umrechnung der Formatierten Koordinaten zu Winkel*/

  /*Formatierung Koordinaten und Höhe*/

  float WGSlat = formatnorth(infoData1.north);
  float WGSlong = formateast(infoData1.east);
  int long Height = formatheight(infoData2.height);

  Serial.print("WGS lat: ");
  Serial.println(WGSlat, 5);
  Serial.print("WGS long: ");
  Serial.println(WGSlong, 5);
  Serial.print("Hohe: ");
  Serial.println(Height);

  /*  Umrechnung WGS84 zu CH1903  */

  CH CHdata = loadData3(WGSlat, WGSlong);

  // Ausgabe CH1903 Koordinaten in Meter
  Serial.print("Ost Koordinate:  ");
  Serial.println(CHdata.east);
  Serial.print("Nord Koordinate:  ");
  Serial.println(CHdata.north);


  /* ELV Soll Winkel ausrechnen */

  float xE = CHdata.east - myEast;
  float yN = CHdata.north - myNorth;
  float distance = sqrt(pow(xE, 2) + pow(yN, 2)); //Distance on the ground
  int elvSOLL = atan(Height / distance) * (180 / pi); //ELVSOLL winkel


  /* AZM Soll Winkel ausrechnen */

  int azmSOLL1 = atan(xE / yN) * (180 / pi);
  int azmSOLL11 = atan(abs(xE) / abs(yN)) * (180 / pi);
  int azmSOLL2 = 180 - azmSOLL11;
  int azmSOLL3 = 180 + azmSOLL11;
  int azmSOLL4 = 360 - azmSOLL11;
  int azmSOLL = 0;

  if (xE > 0 && yN > 0) {
    azmSOLL = azmSOLL1;
  }
  if (xE > 0 && yN < 0) {
    azmSOLL = azmSOLL2;
  }
```

```
    if (xE < 0 && yN < 0) {
      azmSOLL = azmSOLL3;
    }
    if (xE < 0 && yN > 0) {
      azmSOLL = azmSOLL4;
    }

    // Ausgabe AZM ELV Soll Winkel in Grad

    Serial.print("ELV Soll:    ");
    Serial.println(elvSOLL);
    Serial.print("AZM Soll:    ");
    Serial.println(azmSOLL);

    elvMOVE(elvSOLL);  //Fahre mit elvSOLL!
    azmMOVE(azmSOLL);  //Schicke Wert an Ausführungsfunktion

    /* Data logger */
    // make a string that start with a timestamp for assembling the data to log:
    String dataString;
    dataString += getTimeStamp();
    dataString += " =  ";
    dataString += String(infoData1.north);
    dataString += ",";
    dataString += String(infoData1.east);
    dataString += ",";
    dataString += String(infoData2.height);
    dataString += ",";
    dataString += String(CHdata.east);
    dataString += ",";
    dataString += String(CHdata.north);
    dataString += ",";
    dataString += String(elvSOLL);
    dataString += ",";
    dataString += String(azmSOLL);
    dataString += " : ";

    // open the file. note that only one file can be open at a time,
    // so you have to close this one before opening another.
    // The FileSystem card is mounted at the following "/mnt/FileSystema1"
    File dataFile = FileSystem.open("/mnt/sd/datalog.txt", FILE_APPEND);

    // if the file is available, write to it:
    if (dataFile) {
      dataFile.println(dataString);
      dataFile.close();
        print to the serial port too:
      Serial.println(dataString);
    }
    // if the file isn't open, pop up an error:
    else {
      Serial.println("error opening datalog.txt");
    }

    delay(15000);


}//End Big If
```

```
/*--------------Stopp, Conect Again----------------*/

if (Serial.available()) {       // got anything from Serial
  char c = Serial.read();
  if (c == 's') {
    Serial.println("GESTOPPT...");
    delay(3000);
    Serial.println("Program wartet auf neuen Befehl. g = weiter, c = connect
again, l = led leuchtet...");
    int a = 0;
    while (a != 1) {
      if (Serial.available()) {
        c = Serial.read();
        if (c == 'c') {
          Serial.println("connect again...");
          lastConnected = false;
          a = 1;
        }
        if (c == 'g') {
          Serial.println("weiter gehts!...");
          lastConnected = true;
          a = 1;

        }
        if (c == 'l') {
          digitalWrite(ledPin, HIGH);
        }
      }
    }//End While

  }//End if s
}

delay(INTERVAL);
delay(1000);
digitalWrite(ledPin, HIGH);


}//FINISH void Loop

int readAPRS(char * result)
{
  int i = 0;
  int q = 0;
  result[i] = '\0';
  while (APRS.available() > 0 ) {
    char inChar = APRS.read();
    if (inChar == '\n')
    {
      result[i] = '\0';

      APRS.flush();
      result[i] = '\0';
      return 0;
    }
    if (inChar != '\r')
    {
      result[i] = inChar;
      i++;
      if (i == 333) {
        return 0;
```

```
      }
    }
    ;
  } //APRS.available

} //readAPRS

float formatnorth (float North) {

  /* Formatierung der Ausgelesenen Koordinaten*/
  int degnorth1;
  North = North / 100;
  degnorth1 = (int) North; // float to int
  int mnorth1;
  North = (North - degnorth1) * 100;
  mnorth1 = (int) North;
  int snorth1;
  North = (North - mnorth1) * 100;
  snorth1 = (int) (North + .5);
  float deznorth = 0;
  float asdf = 0;
  asdf = snorth1;
  asdf = asdf / 60;
  asdf = asdf + mnorth1;
  asdf = asdf / 60;
  deznorth = deznorth + degnorth1 + asdf;
  Serial.print("nord Koordinate in Dezimalgrad: ");
  Serial.println(deznorth, 5);
  return deznorth;

}//Finish format function

float formateast (float East) {

  /* Formatierung der Ausgelesenen Koordinaten*/
  int degeast;
  East = East / 100;
  degeast = (int) East; // float to int
  int meast;
  East = (East - degeast) * 100;
  meast = (int)  East;
  int seast;
  East = (East - meast) * 100;
  seast = (int) (East + .5);
  float dezeast = 0;
  float w = 0;
  w = seast;
  w = w / 60;
  w = w + meast;
  w = w / 60;
  dezeast = dezeast + degeast + w;
  Serial.print("ost Koordinate in Dezimalgrad: ");
  Serial.println(dezeast, 5);
  return dezeast;

}//Finish format function

float formatheight (float hohe) {
  /* Formatierung der Ausgelesenen Koordinaten*/
  hohe = hohe * ft;
  Serial.println(hohe);
```

```
  return hohe;
}//Finish format function

// This function return a string with the time stamp
String getTimeStamp() {
  String result;
  Process time;
  // date is a command line utility to get the date and the time
  // in different formats depending on the additional parameter
  time.begin("date");
  time.addParameter("+%D-%T");  // parameters: D for the complete date mm/dd/yy
  //             T for the time hh:mm:ss
  time.run();  // run the command

  // read the output of the command
  while (time.available() > 0) {
    char c = time.read();
    if (c != '\n')
      result += c;
  }

  return result;
}//Finish Time Stamp


/* Modul: ELV Motor auf Position bringen */
void elvMOVE(int SOLL)
{
    while(degElvN < SOLL){
      digitalWrite(elvUP, HIGH);
      ElvValue = analogRead(sensorELV);
      degElvN = map(ElvValue, ELVValueLow, ELVValueHigh, 0, ELV90Value);
      if (DebugFlag==true) { Serial.println(degElvN);}
      delay(AnalogREAD);
    }
    digitalWrite(elvUP, LOW);
    while(degElvN > SOLL){
      digitalWrite(elvDOWN, HIGH);
      ElvValue = analogRead(sensorELV);
      degElvN = map(ElvValue, ELVValueLow, ELVValueHigh, 0, ELV90Value);
      if (DebugFlag==true) { Serial.println(degElvN);}
      delay(AnalogREAD);
    }
    digitalWrite(elvDOWN, LOW);
} //finish elvMOVE

/* Modul: AZM Motor auf Position bringen */
void azmMOVE(int SOLL)
{
    while(degAzmN < SOLL){
      digitalWrite(azmR, HIGH);
      AzmValue = analogRead(sensorAzm);
      degAzmN = map(AzmValue, AZMValueLow, AZMValueHigh, 0, AZMStopValueRef);
      if (degAzmN < 90){degAzmN = degAzmN + 270;}
      if (degAzmN >= 90){degAzmN = degAzmN - 90;}
      if (DebugFlag==true) { Serial.println(degAzmN);}
      delay(AnalogREAD);
    }

    digitalWrite(azmR, LOW);
```

```
    while(degAzmN > SOLL){
      digitalWrite(azmL, HIGH);
      AzmValue = analogRead(sensorAzm);
      degAzmN = map(AzmValue, AZMValueLow, AZMValueHigh, 0, AZMStopValueRef);
      if (degAzmN < 90){degAzmN = degAzmN + 270;}
      if (degAzmN >= 90){degAzmN = degAzmN - 90;}
      if (DebugFlag==true) { Serial.println(degAzmN);}
      delay(AnalogREAD);
    }
    digitalWrite(azmL, LOW);
} //finish azmMOVE
```

7.3.1 Info.h Programm Modul

```
//  Info.h
//  Filter für die Koordinaten

#include <stdio.h>
#include <string.h>
#include <stdlib.h>


struct info {
    float north, east;
};


struct info loadData(char input[]) {
    struct info output;
    int stringLength = strlen(input);
    int it = 0;
    while(it < stringLength) {
        if(input[it] == 'h') {
            ++it;
            int jt = 0;
            while(input[++jt + it] != 'N') {}


            char northStr[jt+1];
            strncpy(northStr, input+it, jt );
            northStr[jt ] = '\0';
            output.north = atof(northStr);


            it += jt + 2; //Skip D
            jt = 0;
            while(input[++jt + it] != 'E') {}


            char eastStr[jt+1];
            strncpy(eastStr, input+it, jt);
            eastStr[jt ] = '\0';
            output.east = atof(eastStr);
        }
        ++it;
    }

    return output;
}
```

7.3.2 Info2.h Programm Modul

Dieses Script Filter die Höhe aus dem heruntergeladenem String.

```
// Info2.h
// Filter für die Höhe
//

#include <stdio.h>
#include <string.h>
#include <stdlib.h>


struct info2 {
    float north, east, height;
};


struct info2 loadData2(char input[]) {
    struct info2 output;
    int stringLength = strlen(input);
    int it = 0;
    while(it < stringLength) {
        if(input[it] == '=') {
            ++it;
            int jt = 0;
            while(input[++jt + it] != 'H') {}


            char heightStr[jt];
            strncpy(heightStr, input+it, jt - 1);
            heightStr[jt - 1] = '\0';
            output.height = atof(heightStr);

        }
        ++it;

    }


    return output;
}
```

7.3.3 WGStoCH.h Programm Modul

Diese .h Datei rechnet die GPS Koordinaten in Schweizer Projektionskoordinaten um.

```
//  lars
//  Umrechnung WGS84 zu CH1903
//

#include <math.h>

#define t1  169028.66
#define t2  26782.5

struct CH {
    float north, east;
};


struct CH loadData3(float WGSlat, float WGSlong) {
    struct CH output;

    // 1. in [''] umwandleln
    WGSlat = WGSlat * 3600;
    WGSlong = WGSlong * 3600;

    // 2. Hilfsgrössen [φ', λ'] ausrechnen
    float d1 = WGSlat - t1;
    float d2 = WGSlong - t2;

    float WGSlat2 = d1 / 10000;
    float WGSlong2 = d2 / 10000;

    // 3. CH1903 Koordinaten in Meter ausrechnen
    float X = 600072.37 + 211455.93  * WGSlong2 -  10938.51  * WGSlong2  * WGSlat2
-  0.36  * WGSlong2  * pow(WGSlat2, 2)  -  44.54  * pow(WGSlong2, 3);
    float Y = 200147.07 + 308807.95 * WGSlat2 + 3745.25 * pow(WGSlong2, 2) + 76.63
* pow(WGSlat2, 2) - 194.56 * pow(WGSlong2, 2) * WGSlat2 + 119.79 * pow(WGSlat2,
3);

    output.east = X;
    output.north = Y;

    return output;
}
```

8 Abbildungsverzeichnis

Die folgenden Bilder wurden nicht von mir gemacht. Ich besitze aber – in Absprache mit den Eigentümern – das Recht, sie in meiner Arbeit zu verwenden.

Abbildung 2, Foto: arduino.cc (Creative Commons Attribution ShareAlike 3.0)
Quelle: https://www.arduino.cc/en/uploads/Main/YunParts.png

Abbildung 3, Foto: arduino.cc (Creative Commons Attribution ShareAlike 3.0)
Quelle: https://www.arduino.cc/en/uploads/Main/BridgeInShort.png

Abbildung 7, Foto: C. Wildfeuer

Abbildung 8, Foto: C. Wildfeuer

Abbildung 13, Foto: Casimir Schmid HB9WBU

Abbildung 14, Foto: Marco Oberholzer HB9ZCF,
Quelle: http://sys-tec.ch/marco/wordpress/?page_id=4389 (Stand 28.9.15)

Alle anderen Fotos sind mein Eigentum und unterliegen den Copyright © Bestimmungen.

8.1 Grafikverzeichnis

Die verwendeten Grafiken wurden von mir selbst erstellt.

8.2 Schemaverzeichnis

9 Literaturverzeichnis

[1] K. Rothammel, Antennenbuch, Stuttgart: Franckh-Kosmos, 1991.

[2] „Wikipedia," 2015. [Online]. Available: http://de.wikipedia.org/wiki/Automatic_Packet_Reporting_System. [Zugriff am 20 Januar 2015].

[3] J. K. Joachim Grehn, Metzler Physik, Braunschweig: Schroedel, 2007.

[4] USKA Schweiz - Union Schweizerischer Kurzwellen-Amateure, „USKA.ch," 2015. [Online]. Available: http://uska.ch/amateurfunkpraxis/funkbetrieb/aprs/. [Zugriff am 17 Juli 2015].

[5] swisstopo, Bundesamt für Landestopografie, «Formeln und Konstanten für die Berechnung der Schweizerischen schiefachsigen Zylinderprokjektion und der Transformation zwischen Koordinatensystemen,» 2008.

[6] Arduino, „arduino.cc," 2015. [Online]. Available: https://www.arduino.cc/. [Zugriff am 17 Juli 2015].

[7] „Wikipedia," [Online]. Available: http://de.wikipedia.org/wiki/Automatic_Packet_Reporting_System. [Zugriff am 20 Januar 2015].

[8] „Wikipedia," 2015. [Online]. Available: https://de.wikipedia.org/wiki/70-Zentimeter-Band. [Zugriff am 5 September 2015].

[9] „Wikipedia," 2015. [Online]. Available: https://de.wikipedia.org/wiki/2-Meter-Band. [Zugriff am 5 September 2015].

10 Legende:

Open Source = quelloffen, d.h. alle Schemas, Programme und Code sind einsehbar.

Digipeater = Ein Digipeater (von engl. digital repeater) ist eine automatisch arbeitende Sende- und Empfangsstation zur Weiterleitung digital codierter Information zwischen zwei Funkstationen. Analog zum Digipeater im Datenfunk spricht man bei Sprechfunk von einer Relaisstation. [2]

I-Gate (Internet Gateway) = Empfangsstation mit Interneteinspeisung der empfangenen Daten

ELV = Elevation

AZM = Azimut

GHz = Giga Herz

MHz = Mega Herz

.h Datei = Unterprogramm

Void-Funktion = Eine Funktion der Programmsprache, welche keine Information zurückgibt.

Helix-Antenne = Richtantennenart, welche spiralförmig ist.

11 Deklaration

„Ich erkläre hiermit, dass ich die vorliegende Maturaarbeit selbständig und ohne unerlaubte fremde Hilfe erstellt habe und dass alle Quellen, Hilfsmittel und Internetseiten wahrheitsgetreu verwendet wurden und belegt sind.“

Unterschrift, Datum und Ort

_____

Lars Horvath