

P&S SDR: Sat-Tracker Dokumentation

Sean Helle, Charpoan Kong

Ziel

Das Ziel des Projektes ist einen Satelliten-Tracker zu bauen mit dem Ziel Amateurfunk- und Wettersatelliten zu empfangen u.a auch QSO's dadurch zu führen.

Systemflow

Um den Antennenrotor via gPredict zu steuern. Lies ein Pythonscript die Sat. Daten von gPredict via TCP aus und gibt diese per serieller Schnittstelle zum Pico weiter. Der Pico liest den Winkel des Rotors via des Rotorcontrollers aus und steuert die einzelnen Motoren via des Rotorcontrollers. Auch steuert gPredict das Radio und kompensiert die Frequenz zum Satelliten wegen des Dopplereffektes.



Figure 1: Datenflow

Der Programflow sieht wie folgt aus:

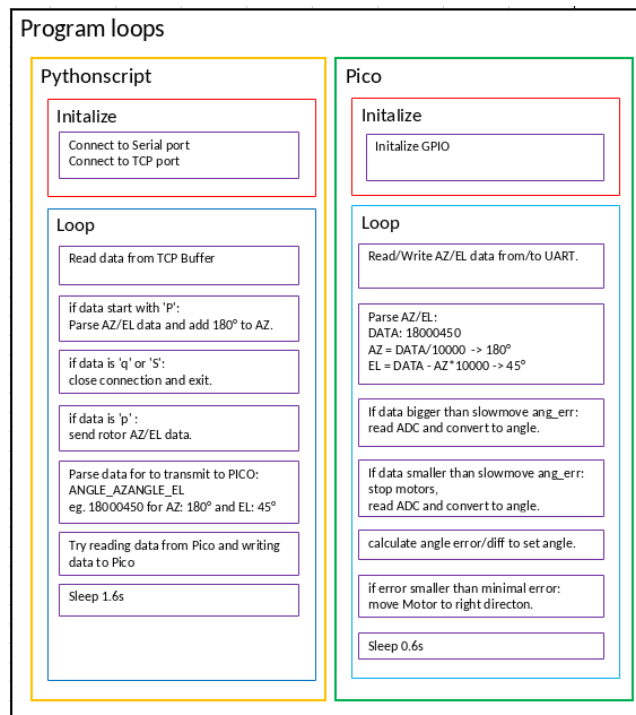


Figure 2: Programmflow

Antennenrotor-Controller

Picocontroller

Der Pico steuert den Rotorcontroller durch den DIN-8 Stecker. Durch die [Dokumentation](#) des Controllers wurde herausgefunden welcher Pin welche Funktion hat. Zudem konnte herausgefunden werden, dass um die einzelnen Motoren anzusteuern, die "Steuer"-Pins auf GND kurzgeschlossen werden müssen.

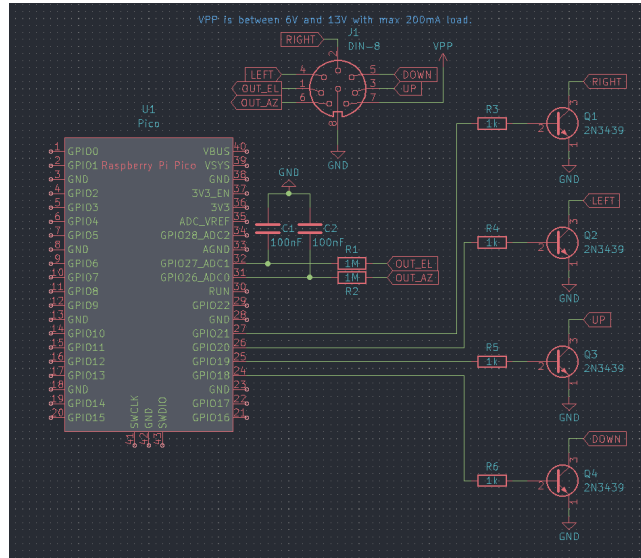


Figure 3: Schaltplan Pico zu Rotor-Controller

Problem mit dem Spannungsabfall

Durch Anmerkung von Herr Lerjen vom Projekt Stratos, konnte ein Spannungsabfall beobachtet werden, wenn die Motoren stoppen. Dies ist ein Problem, wenn der Motor den Endwinkel erreicht, da die Werte nicht mehr stimmen. Deswegen stoppen alle Motoren beim auslesen der Spannung, wenn der Motor in der Nähe des Soll-Winkels ist. Jedoch gab es dann Probleme, da der Motor "oszillierte". Dieses wurde so gelöst, dass ein minimaler Abweichungswinkel vom Soll-Winkel eingestellt wurde. Und dass der Motor sich um genug grosse Winkel bewegt.

Pythonskript

Gpredict steuert die kommerziellen via TCP an und es konnte schon eine Analyse des Protokolls mit Beispiel [Python-Code](#) gefunden werden. Dieser wurde angepasst und der Code zum steuern des Pico wurde hinzugefügt.

Pico code

```
1 #include <stdio.h>
2 #include <math.h>
3 #include "pico/stdlib.h"
4 #include "pico/util/queue.h"
5 #include "pico/multicore.h"
6 #include "hardware/adc.h"
7
8 const uint8_t AZ_R = 21, AZ_L = 20, EL_U = 19, EL_D = 18; //AZ/EL Pins
9
10 void MotorControl(){
11     uint32_t buffer = 1; //UART buffer
12
13     float azel[2] = {10, 10}; //true AZ/EL angle
14
15     float tgt_azel[2] = {1800, 900}; //target AZ/EL angle
16
17     float err_azel[2] = {0, 0}; //error AZ/EL angel
18
19     const float minERR = 50; //minimal tolerated error angle: 5deg
20     const float slowERR = 100; //error angle when to compensate for
    voltage drop: 10deg
21     const float maxaz = 3360, maxel = 3520; //max ADC value for max angle ->
    calibration
22
23     while(true){
24         // write data to UART
25         printf("%f/%f/%u\n", azel[0], azel[1], buffer);
26         fflush(NULL); //flush buffer
27         scanf("%u\r\n", &buffer); //read data from UART
28
29         //Parse AZEL
30         //Eg. DATA = 18000450
31         //AZ = DATA/10000 -> 180 because is integer and everything after the
    comma is "cut out".
32         //EL = DATA - AZ*10000 -> 45
33         tgt_azel[0] = (buffer/10000);
34         tgt_azel[1] = (buffer - (tgt_azel[0] * 10000));
35
36         //calculate error AZ/EL angle
37         err_azel[0] = tgt_azel[0] - azel[0];
38         err_azel[1] = tgt_azel[1] - azel[1];
39
40         if(std::abs(err_azel[0]) > slowERR || std::abs(err_azel[1]) >
    slowERR){ // if error bigger the slowERR -> because of voltage drop
41             //read ADC and convert to angle
42             adc_select_input(0);
43             azel[1] = (((float)adc_read()/maxel)*1800);
44             adc_select_input(1);
45             azel[0] = (((float)adc_read()/maxaz)*3600);
46         }else{ //if smaller than slowERR
47             //shutoff all Motors to stop voltage drop
48             gpio_put(AZ_R, 0);
49             gpio_put(AZ_L, 0);
50             gpio_put(EL_U, 0);
51             gpio_put(EL_D, 0);
52
53             sleep_ms(400); //wait for motor halt and voltage settling
54
55             //read ADC and convert to error
```

```

56         adc_select_input(0);
57         azel[1] = (((float)adc_read()/maxel)*1800);
58         adc_select_input(1);
59         azel[0] = (((float)adc_read()/maxaz)*3600);
60     }
61
62     //if error is bigger than the minimal error move motors
63     if(std::abs(err_azel[0]) > minERR){
64         if (err_azel[0] <= 0){
65             gpio_put(AZ_R, 0);
66             gpio_put(AZ_L, 1);
67         }else if(err_azel[0] > 0){
68             gpio_put(AZ_R, 1);
69             gpio_put(AZ_L, 0);
70         }
71     }else{
72         gpio_put(AZ_R, 0);
73         gpio_put(AZ_L, 0);
74     }
75
76     if(std::abs(err_azel[1]) > minERR){
77         if (err_azel[1] <= 0){
78             gpio_put(EL_U, 1);
79             gpio_put(EL_D, 0);
80         }else if(err_azel[1] > 0){
81             gpio_put(EL_U, 0);
82             gpio_put(EL_D, 1);
83         }
84     }else{
85         gpio_put(EL_U, 0);
86         gpio_put(EL_D, 0);
87     }
88
89     sleep_ms(600); //sleep for 0.6s -> enough movement that motor does no
90     oscillate with the error resolution.s
91 }
92
93
94
95 int main() {
96     //inititalize stdlib
97     stdio_init_all();
98     //inititalize adc
99     adc_init();
100
101     //inititalize ADC_GPIO
102     adc_gpio_init(26);
103     adc_gpio_init(27);
104
105     //inititalize GPIO
106     gpio_init(AZ_R);
107     gpio_init(AZ_L);
108     gpio_init(EL_U);
109     gpio_init(EL_D);
110
111     //set GPIO function
112     gpio_set_dir(AZ_R, GPIO_OUT);
113     gpio_set_dir(AZ_L, GPIO_OUT);
114     gpio_set_dir(EL_U, GPIO_OUT);
115     gpio_set_dir(EL_D, GPIO_OUT);

```

```
116
117 //call MotorControl
118 MotorControl();
119 }
```

Python code

```
1 import socket
2 import serial
3 from time import sleep
4
5 #inititalize Serial port
6 ser = serial.Serial('/dev/ttyACM2', 9600, timeout=1, parity=serial.
    PARITY_EVEN)
7
8
9 TCP_IP = '127.0.0.1'
10 TCP_PORT = 4533
11 BUFFER_SIZE = 100
12
13 #inititalize/connect to TCP Port // data parsing form https://adventurist.me/
    posts/0136
14 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
15 s.bind((TCP_IP, TCP_PORT))
16 s.listen(1)
17
18 conn, addr = s.accept()
19 print('Connection address:', addr)
20
21 az = 0.0 #AZ from Pico
22 el = 0.0 #EL from Pico
23
24 mmaz = 0 #AZ from gPredict
25 mmel = 0 #EL from gPredict
26
27 maz = '9999' #AZ to Pico
28 mel = '9999' #EL to Pico
29
30 response = " " # data to gPredict
31 while 1:
32     data = conn.recv(BUFFER_SIZE) #read data fromm gPredict
33     response = "{}\n{}\n".format(float(f'{az:.2f}'), float(f'{el:.2f}')) #
        parse data to gPredict
34
35     if (data.startswith(b'P')):
36         values = data.split(b' ')
37         #print(values)
38         mmaz = int(float(values[1])*10)+1800 #convert to integer and add 180deg
        // 1800 because 0.1deg resolution -> unnecessary...
39         mmel = int(float(values[2])*10) #convert to interger
40
41         conn.send(bytes(response, 'utf-8')) #send data from Pico to gPredict
42     elif data == b'q\n' or data == b'S\n':
43         print("close command, shutting down") #closes Program
44         conn.close()
45         exit()
46     elif data == b'p\n':
47         conn.send(bytes(response, 'utf-8')) # send data from Pico to gPredict
48
49     print(response)
50     print("moving to az:{} el: {}".format( mmaz, mmel));
51
52     # Parse data for Pico:
53     # AZ: 180deg, EL: 45deg -> 18000450
54
55     if(mmaz == 0):
```

```

56     maz = '000' + str(mmaz)
57 if(mmaz < 100):
58     maz = '00' + str(mmaz)
59 elif(mmaz < 1000):
60     maz = '0' + str(mmaz)
61 else:
62     maz = str(mmaz)
63
64 if(mmel == 0):
65     mel = '000' + str(mmel)
66 elif(mmel < 100):
67     mel = '00' + str(mmel)
68 elif(mmel < 1000):
69     mel = '0' + str(mmel)
70 else:
71     mel = str(mmel)
72
73 #Try to write to the Pico and clear buffer if not print nosend
74 try:
75     print("az: " + str(mmaz/10) + " el: " + str(mmel/10))
76     ser.reset_output_buffer()
77     ser.write(bytes(maz + mel + '\r\n', 'ascii'))
78     ser.reset_output_buffer()
79
80 except:
81     print('nosend')
82     ser.reset_output_buffer()
83     print(str(maz + mel))
84
85 #Try to read to the Pico and clear buffer if not print nor
86 try:
87     if(ser.in_waiting > 0 ):
88         adc = ser.readline()
89         ser.reset_input_buffer()
90         ser.read_all()
91         adc = adc.split(b'/')
92         print(adc)
93         print("az: " + str(float(adc[0])/10) + " el: " + str(float(adc[1])/10)
94 + " RECEV:" + str(adc[2]))
94         az = int(float(adc[0])/10)
95         el = int(float(adc[1])/10)
96         sleep(0.5)
97 except:
98     print('nor')
99
100 sleep(1.6) #sleep for clearing buffer.

```