Diesen Beitrag gibt es auch als:
PDF in deutsch
This episode is also available as:
PDF in English

Today it's about sending text messages over the cellular network via SMS. ESP32 and SIM808 form a kind of server or relay station to which text messages can be sent for control purposes, but which is also able to send messages itself, time-controlled or event-controlled. The cell phone or the PC, the Raspi ... is then the switching and receiving center. And because the distance between the SIM808 and the mobile phone is irrelevant, there is no distance limit, provided that there is sufficient network coverage. The LCD keypad is actually superfluous because direct control of the remote ESP32 unit is out of the question anyway. Nevertheless, a display does a good job when starting the relay unit and for further status messages when debugging. Perhaps more interesting than an LCD is a small OLED display, purely for maintenance purposes, but this can also be done via SMS. So welcome to the third part of

# GPS and GSM with MicroPython on the ESP32

## Hardware growth - the SIM card

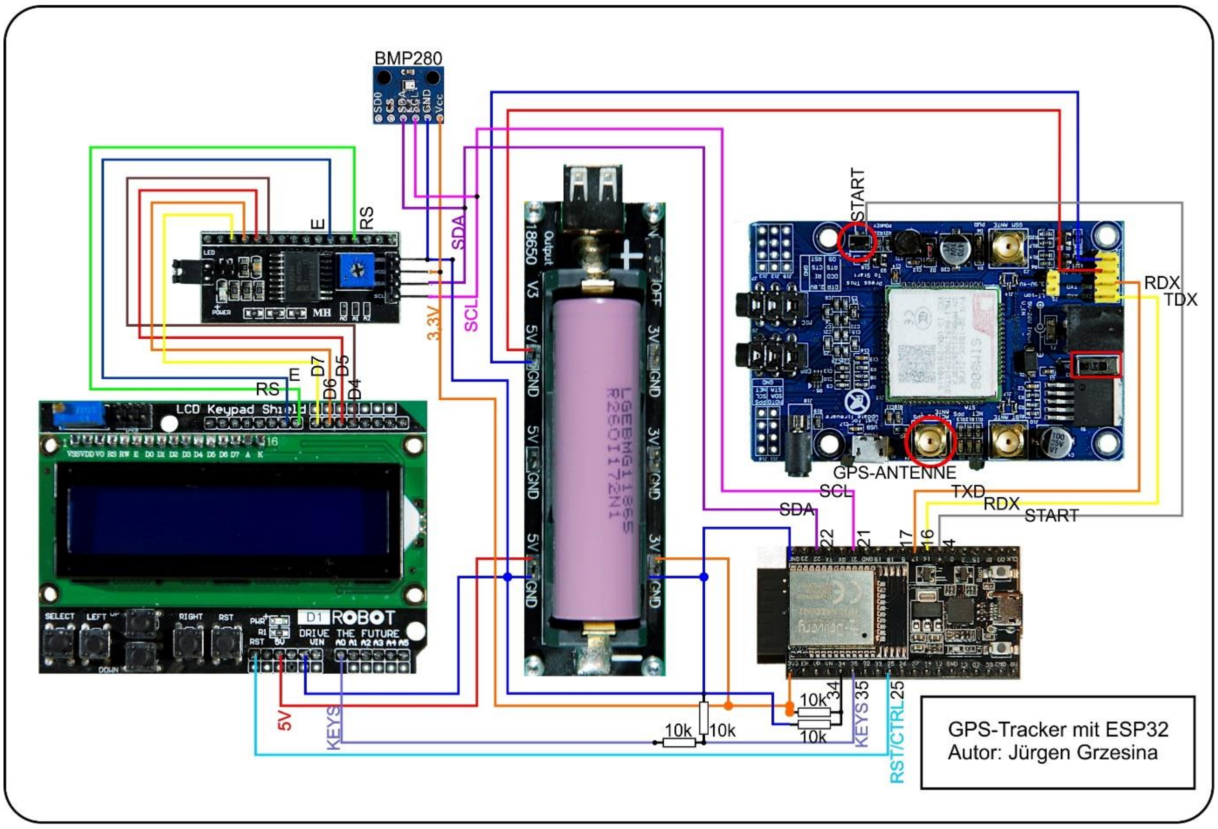
There is little in terms of hardware compared to Tei2. How? You have not read Parts 1 and 2 and are new here? All right, persuaded, in the following list you will find all the parts for the current project. Almost all of it has already been used in Part 1 and

Part 2 and of course also described in detail there. Of course, we reuse these components. A new addition is a SIM card, because without it you will not be able to send or receive SMS messages. Of course, it is possible that not all sensors and / or actuators can be connected directly to the relay station because of the length of the cables. Then it wouldn't be bad if there was a radio link for it. Therefore, in the next episode, I will show you how something like this can be implemented very easily with the UDP protocol via WLAN connections. For this purpose, I'll use an ESP8266 with an LDR resistor as a wireless sensor as an example. Other sensors such as DS18B20 (temperature), DHT22 AM2302 (humidity and temperature), GY-302 BH1750 (light sensor), etc. can be used just as well thanks to the various MicroPython modules that are available for this. But now first about the GSM connection, we want to text a little with our ESP32.

Here is the list of ingredients, the recipe for the menu comes later.

| | |
|---|---|
| 1 | ESP32 Dev Kit C V4 unverlötet oder ähnlich |
| 1 | LCD1602 Display Keypad Shield HD44780 1602 Modul mit 2x16 Zeichen |
| 1 | SIM 808 GPRS/GSM Shield mit GPS Antenne für Arduino |
| 1 | Battery Expansion Shield 18650 V3 inkl. USB Kabel |
| 1 | Li-Akku Typ 18650 |
| 1 | I2C IIC Adapter serielle Schnittstelle für LCD Display 1602 und 2004 |
| 4 | Widerstand 10kΩ |
| 1 | GY-BMP280 Barometrischer Sensor für Luftdruckmessung |
| 1 | SIM-Karte (beliebiger Anbieter) |

The circuit for the project is first taken over 1: 1 from part 2. Later you decide for yourself which parts you want to leave out, replace or add new ones. You can find the program options for implementation in this article.



You will receive a more legible copy of the illustration in DIN A4 with the Download of the PDF-File .

## The Software

**Used Software:**
For flashing and programing the ESP:
Thonny oder
µPyCraft

**Used Firmware:**
MicropythonFirmware

**MicroPython-Module und Programme**
GPS-Modul für SIM808 und GPS6MV2(U-Blocks)
LCD-Standard-Modul
HD44780U-I2C-Expansion for LCD-Modul
Keypad-Modul
Button Modul
BMP208-Modul
i2cbus-Modul for standardized access to the bus
Das Hauptprogramm relais.py
testkeypad.py for testing the key decoding of the LCD keypad

## Tricks and information on MicroPython

The interpreter language MicroPython is used in this project. The main difference to the Arduino IDE is that you have to flash the MicroPython firmware onto the ESP32 before the controller understands MicroPython instructions. You can use Thonny, µPyCraft or esptool.py for this. For Thonny, I described the process in the first part of the blog on this topic.

After the firmware has been flashed, you can have a casual conversation with your controller, test individual commands and immediately see the answer without first having to compile and transfer an entire program. I made ample use of it when developing the software for this blog. The spectrum ranges from simple tests of the syntax to trying out and refining functions and entire program parts. For this purpose, as in the previous episodes, I will create small test programs. They form a kind of macro because they combine recurring commands.

Such programs are started from the current editor window in the Thonny IDE using the F5 key, which is faster than clicking the start button or using the Run menu. I also described the installation of Thonny in detail in the first part.

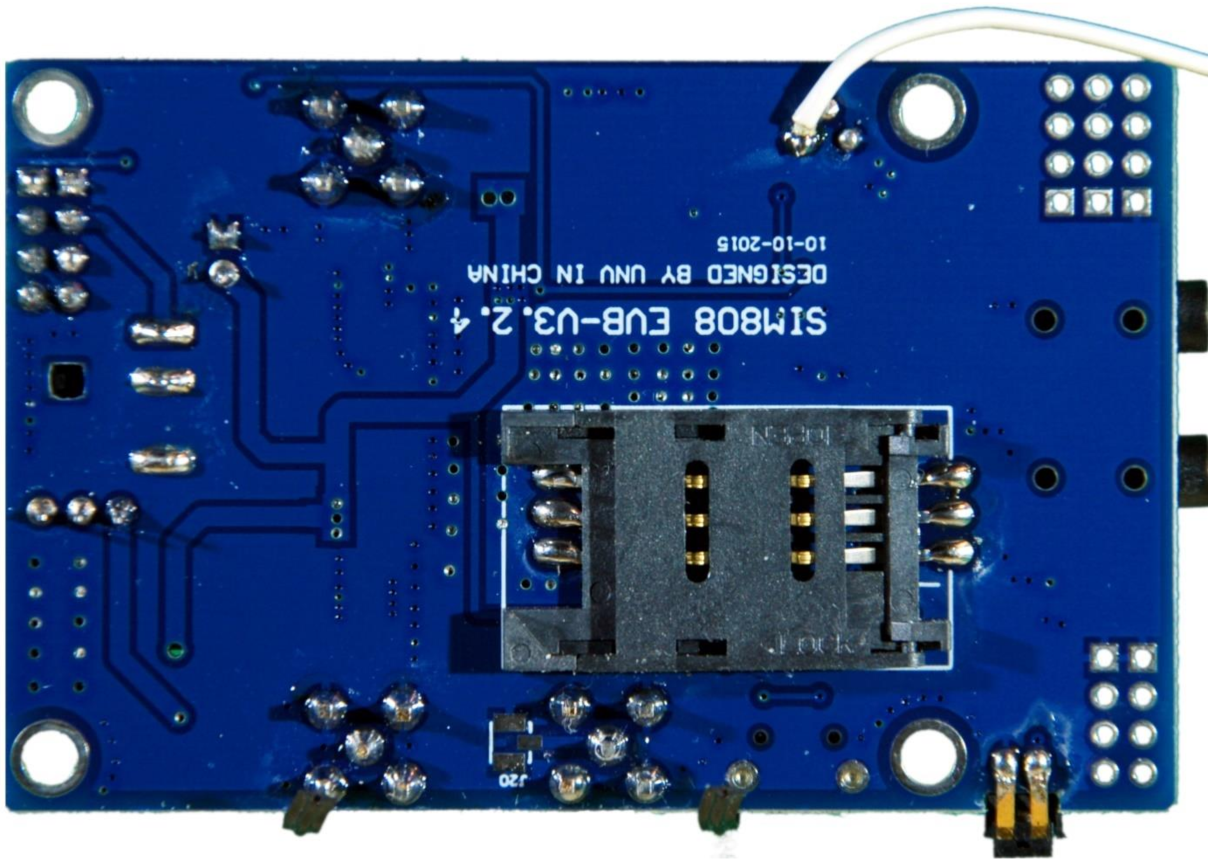
## The classes GPS, SIM808 and GSM

Well, the most important thing for a GPS application is: How do I address the GPS services of the SIM808? Oh yes, right - this article should not (only) be about GPS but primarily about GSM. So the question must be different. Maybe: How can you text with the SIM808 and the ESP32? This is exactly what we want to look at now. The AT commands allow us to easily handle the diverse properties of the SIM808 module. I used a very small part to build the GSM class for my project. Together with the GPS class for localization and the SIM808 class for hardware control, you will find GSM peacefully united in the gps.py module.

For newly added readers, put the SIM808 module into operation. You have to slide the small slide switch right next to the pipe socket for the power supply, 5V to 12V, towards the SIM808 chip. A red LED lights up next to the GSM antenna socket. You will surely find that very easy, because all solder points and interfaces are well documented on the board.

A little further to the left of the GSM antenna socket is the start button. You can also use the circuit diagram above for orientation. Press the start button for approx. 1 second, then two more LEDs light up between the other two antenna sockets, the right one flashes every second. The active GPS antenna should already be connected to the left screw socket. It is best to place them near a window.

So that you don't have to open the housing of your GPS receiver every time to start the SIM808, I recommend that you do the same and solder a cable to the hot connection of the start button. Viewed from above, it is the right one if the pipe socket also points to the right. You can now start the SIM808 by defining a GPIO pin of the ESP32 as an output and switching from high to low and back to high for one second. I intended pin 4 for this.



When the constructor for the GPS object is called, the number of the pin is transferred together with the display object as a parameter. Before you send the following commands to the ESP32, please upload the modules linked at the beginning to the controller's flash memory. The commands are entered via the command line in the terminal area.

```
>>> from gps import GPS,SIM808,GSM
>>> from lcd import LCD
>>> from machine import ADC, Pin, I2C
>>> from keypad import KEYPAD
>>> i2c=I2C(-1,Pin(21),Pin(22))
>>> d=LCD(i2c,0x27,cols=16,lines=2)
>>> g=GSM(4,d)
>>> k=KEYPAD(35)
```

If no display object (d) is passed, there is of course no output on the LCD or OLED. However, there is no error message, the key control works normally. Almost all important results are output in the terminal window.

The GPS class does most of the work. As mentioned, the constructor expects a display object that must be defined in the calling program or must already be known. A serial channel to the SIM808 is opened at 9600 baud, 8,0,1, then the instance variables are set up to record the GPS data.

Here is an overview of the most important methods of the GPS class.

The waitForLine () method does what its name says, it waits for a NMEA sentence from the SIM808. The type of NMEA sentence that is expected is given as a parameter. If the record is complete and free of errors, it is returned to the calling program. In the current version of the program, $ GPRMC and $ GPGGA records can be received. They contain all relevant data such as validity, date, time, geographical latitude (latitude, from the equator to the poles in degrees) and longitude (longitude from the zero meridian +/- 180 °) as well as height above sea level in meters. Similar to the existing code, further data records can easily be recorded and decoded by the SIM808.

The method decodeLine () takes the received record and tries to parse it. This method contains a local function that converts the angle specifications into the formats degrees, minutes, seconds and fractions, degrees and fractions, or degrees, minutes and fractions, according to the specification of the mode attribute.

The method printData () outputs a data record in the terminal window. showData () returns the result to the display. Because only a two-line display is used with the LCD keypad, the display must be divided into several sections. The keys on the keypad take control.

Because the UART0 interface is reserved for REPL, a second interface must be available for communication with the SIM808. The ESP32 provides such a UART2. The connections for RXD (reception) and TXD (transmission) can even be freely selected. For full duplex operation (send and receive simultaneously) the RXD and TXD connections from the ESP32 to the SIM808 must be crossed. You can understand this on the circuit diagram. The default values on the ESP32 are RXD = 16 and TXD = 17. The connection is organized by the gps.GPS class.

This begins when the SIM808 is switched on. If you followed my recommendation and soldered a cable to the power button, you can now switch on the SIM808 with the following command, provided that this cable is connected to pin 4 of the ESP32.

```
>>> g.SIMOn ()
```

Commands to the SIM808 are transmitted in AT format. There is a huge variety of commands in a PDF-File where they can be studied in detail. But don't worry, a few commands are enough for our project. Two of them are summarized in the methods init808 () and deinit808 (), a few more are presented in the GSM chapter.

```
    def init808(self):
        self.u.write("AT+CGNSPWR=1\r\n")
        self.u.write("AT+CGNSTST=1\r\n")

    def deinit808(self):
        self.u.write("AT+CGNSPWR=0\r\n")
        self.u.write("AT+CGNSTST=0\r\n")
```

AT + CGNSPWR = 1 activates the GPS module and AT + CGNSTST = 1 activates the transmission of the NMEA sentences to the ESP32 via the serial interface UART2. The controller receives the information from the SIM808 and provides it in the manner described above via the terminal and LCD.

In addition to the hardware control of the SIM808, the gps.py module also contains the necessary commands for the smaller GPS system GPS6MV2 with the Neo 6M chip from UBLOX. This module is not controlled via AT commands, but via its own syntax. The SIM808 and GPS6MV2 classes are not interchangeable because they have different APIs.

The listing now follows to study the gps module in more detail. The three included classes GPS, SIM808 (GPS) and GSM (SIM808) build a uniform namespace through inheritance. Therefore, all methods are available in an object of the GSM class. If GSM is not needed (no SMS transfer), you can also enter via the SIM808 class, as was done in Part 2.

The SIM808 class takes care of the hardware control and the data transfer to the ESP32. The GPS class contains methods for decoding the NMEA sentences from the SIM808, for displaying them on the display and for calculating the course. Finally, the GSM class provides the methods for SMS transfer and message management. These messages need not relate solely to GPS data. Rather, I kept the gps.GSM class neutral so that it can also be used in other projects. In the relais.py program you will not only find a GPS approach, but also the implementation of a BMP280 query via GSM as an example for the integration of further sensors. The BMP280 and I2CBus classes required for this have already been presented in Part 2.

```
"""
File: gps.py
Author: J. Grzesina
Rev. 1.0: AVR-Assembler
Rev. 2.0: Adaption auf Micropython
------------------
Die enthaltenen Klassen sprechen einen ESP32 als Controller
an.
Dieses Modul beherbergt die Klassen GPS, GPS6MV2 und SIM808
GPS stellt Methoden zur Decodierung und Verarbeitung der NMEA-
Saetze
$GPGAA und $GPRMC bereit, welche die wesentlichen Infos zur
Position, Hoehe und Zeit einer Position liefern. Sie werden
dann
angezeigt, wenn die Datensaetze als "gueltig" gemeldet werden.
Eine Skalierung auf weitere NMEA-Sätze ist jederzeit möglich.
GPS6MV2 und SIM808 beziehen sich auf die entsprechende
Hardware.
"""
from machine import UART,I2C,Pin
import sys
from time import sleep, time, ticks_ms
from math import *

# ********************** Beginn GSM
****************************PS
class GPS:
    #
    gDeg=const(0)
    gFdeg=const(1)
    gMin=const(1)
    gFmin=const(2)
    gSec=const(2)
    gFsec=const(3)
    gHemi=const(4)

    #DEFAULT_TIMEOUT=500
    #CHAR_TIMEOUT=200

    def __init__(self,disp=None,key=None):  # display mit
OLED-API
        self.u=UART(2,9600)
        # u=UART(2,9600,tx=19,rx=18)  # mit alternativen Pins
        self.display=disp
        self.key=key
        self.timecorr=2
        self.Latitude=""
        self.Longitude=""
        self.Time=""
        self.Date=""
        self.Height=""
        self.Valid=""
        self.Mode="DMF" # default
        self.AngleModes=["DDF","DMS","DMF"]
        self.displayModes=["time","height","pos"]
        self.DMode="pos"
        # DDF = Degrees + DegreeFractions
        # DMS = Degrees + Minutes + Seconds + Fractions
        # DMF = Degrees + Minutes + MinuteFraktions
        self.DDLat=49.28868056  # aktuelle Position
        self.DDLon=11.47506105
        self.DDLatOld=49.3223 # vorige Position
        self.DDLonOld=11.5000
        self.zielPtr=0
        self.course=0
        self.distance=0
        print("GPS initialized,
Position:{},{}".format(self.DDLat,self.DDLon))

    def decodeLine(self,zeile):
        latitude=["","","","","N"]
        longitude=["","","","","E"]
        angleDecimal=0
        def formatAngle(angle):  # Eingabe ist Deg:Min:Fmin
            nonlocal angleDecimal
            minute=int(angle[1])     # min als int
            minFrac=float("0."+angle[2]) # minfrac als float

angleDecimal=int(angle[0])+(float(minute)+minFrac)/60
            if self.Mode == "DMS":
                seconds=minFrac*60
                secInt=int(seconds)
                secFrac=str(seconds - secInt)

a=str(int(angle[0]))+"*"+angle[1]+"'"+str(secInt)+secFrac[1:6]
+'"'+angle[4]
            elif self.Mode == "DDF":
                minutes=minute+minFrac
                degFrac=str(minutes/60)
                a=str(int(angle[0]))+degFrac[1:]+"* "+angle[4]
            else:

a=str(int(angle[0]))+"*"+angle[1]+"."+angle[2]+"' "+angle[4]
            return a

        # GPGGA-Fields
        nmea=[0]*16
        name=const(0)
        time=const(1)
        lati=const(2)
        hemi=const(3)
        long=const(4)
        part=const(5)
        qual=const(6)
        sats=const(7)
        hdop=const(8)
        alti=const(9)
        auni=const(10)
        geos=const(11)
        geou=const(12)
        aged=const(13)
        trash=const(14)
        nmea=zeile.split(",")
        lineStart=nmea[0]
        if lineStart == "$GPGGA":

self.Time=str((int(nmea[time][:2])+self.timecorr)%24)+":"+nmea
[time][2:4]+":"+nmea[time][4:6]
            latitude[gDeg]=nmea[lati][:2]
            latitude[gMin]=nmea[lati][2:4]
            latitude[gFmin]=nmea[lati][5:]
            latitude[gHemi]=nmea[hemi]
            longitude[gDeg]=nmea[long][:3]
            longitude[gMin]=nmea[long][3:5]
            longitude[gFmin]=nmea[long][6:]
            longitude[gHemi]=nmea[part]
            self.Height,despose=nmea[alti].split(".")
            self.Latitude=formatAngle(latitude)  # mode =
Zielmodus Winkelangabe
            self.DDLat=angleDecimal
            self.Longitude=formatAngle(longitude)
            self.DDLon=angleDecimal
        if lineStart == "$GPRMC":
            date=nmea[9]
            self.Date=date[:2]+"."+date[2:4]+"."+date[4:]
            try:
                self.Valid=nmea[2]
            except:
                self.Valid="V"

    def waitForLine(self,title,delay=2000):
        line=""
        c=""
        d=delay
        if delay < 1000: d=1000
        start = ticks_ms()
        end=start+d
        current=start
        while current <= end:
            #print(end-current)
            if self.u.any():
                c=self.u.read(1)
                if ord(c) <=126:
                    c=c.decode()
                    if c == "\n":
                        test=line[0:6]
                        if test==title:
                            #print(line)
                            return line
                        else:
                            line=""
                    else:
                        if c != "\r":
                            line +=c
            current = ticks_ms()
            sleep(0.05)
        return ""

    def showData(self):
        if self.display:
            if self.DMode=="time":

self.display.writeAt("Date:{}".format(self.Date),0,0)

self.display.writeAt("Time:{}".format(self.Time),0,1)
            if self.DMode=="height":
                self.display.writeAt("Height: {}m
".format(self.Height),0,0)

self.display.writeAt("Time:{}".format(self.Time),0,1)
            if self.DMode=="pos":
                self.display.writeAt(self.Latitude+" "*(16-
len(self.Latitude)),0,0)
                self.display.writeAt(self.Longitude+" "*(16-
len(self.Longitude)),0,1)
```

```
    def printData(self):
        print(self.Date,self.Time,sep="_")
        print("LAT",self.Latitude)
        print("LON",self.Longitude)
        print("ALT",self.Height)

    def showError(self,msg):
            if self.display:
                self.display.clearAll()
                self.display.writeAt(msg,0,0)
            print(msg)

    def storePosition(self):  # aktuelle Position als DD.dddd
merken
        lat=str(self.DDLat)+","
        lon=str(self.DDLon)+"\n"
        try:
            D=open("stored.pos","wt")
            D.write(lat)
            D.write(lon)
            D.close()
            if self.display:
                self.display.clearAll()
                self.display.writeAt("Pos. stored",0,0)
                sleep(3)
                self.display.clearAll()
        except (OSError) as e:
            enumber=e.args[0]
            if enumber==2:
                print("Not stored")
                if self.display:
                    self.display.clearAll()
                    self.display.writeAt("act. Position",0,0)
                    self.display.writeAt("not stored",0,0)
                    sleep(3)
                    self.display.clearAll()

    def chooseDestination(self, wait=3):
        if not self.display: return None
        self.display.clearAll()
        self.display.writeAt("ENTER=RST-Button",0,0)
        n="positions.pos"
        try:
            D=open(n,"rt")
            ziel=D.readlines()
            D.close()
            i = 0
            while 1:
                lat,lon=(ziel[i].rstrip("\r\n")).split(",")
                self.display.clearAll()
                self.display.writeAt("{}.
{}".format(i,lat),0,0)
                self.display.writeAt("   {}".format(lon),0,1)
                sleep(wait)
                if self.key.value()==0: break
                i+=1
                if i>=len(ziel): i=0
            self.zielPtr=i
            self.display.clearAll()
            self.display.writeAt("picked: {}".format(i),0,0)
            sleep(wait)
            self.display.clearAll()
            lat,lon=ziel[i].split(",")
            lon=lon.strip("\r\n")
            print("{}. Lat,Lon: {}, {}".format(i,lat,lon))
            lat=float(lat)
            lon=float(lon)
            return (lat,lon)
        except (OSError) as e:
            enumber=e.args[0]
            if enumber==2: print("File not found")
            self.display.clearAll()
            self.display.writeAt("There is no",0,0)
            self.display.writeAt("Positionfile",0,1)
            sleep(3)
            self.display.clearAll()
            return (0,0)

    def calcNewCourse(self,delay=3): # von letzter Position
bis hier
        lat,lon=self.chooseDestination(delay)
        if lat==0 and lon==0: return
        dy=(lat-self.DDLat)*60*1852
        dx=(lon-
self.DDLon)*60*1852*cos(radians(self.DDLatOld))
        print("Start {},{}".format(self.DDLat,self.DDLon),"
Ziel {},{}".format(lat,lon))
        #print("Ziel-Start",self.DDLon-self.DDLonOld,
self.DDLat-self.DDLatOld)
        return self.calcCourse(dx,dy)

    def calcLastCourse(self): # von letzter Position bis hier
        try:
            D=open("stored.pos","rt")
            lat,lon=(D.readline()).split(",")
            D.close()
            self.DDLatOld=float(lat)
            self.DDLonOld=float(lon)
        except:
            self.DDLatOld=49.3223
            self.DDLonOld=11.50
        dy=(self.DDLat-self.DDLatOld)*60*1852
        dx=(self.DDLon-
self.DDLonOld)*60*1852*cos(radians(self.DDLatOld))
        print("Start
{},{}".format(self.DDLonOld,self.DDLatOld),"  Ziel
{},{}".format(self.DDLon,self.DDLat))
        #print("Ziel-Start",self.DDLon-self.DDLonOld,
self.DDLat-self.DDLatOld)
        return self.calcCourse(dx,dy)

    def calcCourse(self,dx,dy): # von letzter Position bis
hier
        course=0
        distance=0
        #print(dx,dy,degrees(atan2(dy,dx)))
        if abs(dx) < 0.0002:
            if dy > 0:
                course=0
                #print("Trace: 1")
            if dy < 0:
                course=180
                #print("Trace: 2")
            if abs(dy) < 0.0002:
                course=None
                #print("Trace: 3")
        else:  # dx >= 0.0002
            if abs(dy) < 0.0002:
                if dx > 0:
                    course=90
                    #print("Trace: 4")
                if dx < 0:
                    course=270
                    #print("Trace: 5")
            else:  ## dy > 0.0002
                course=90-degrees(atan2(dy,dx))
                #print("Trace: 6")
                if course > 360:
                    course -= 360
                    #print("Trace: 7")
                if course < 0:
                    course += 360
                    print("Trace: 8")
        self.course=int(course)
        self.distance=int(sqrt(dx*dx+dy*dy))
        print("Distance: {}, Course:
{}".format(self.distance,self.course))
        return (self.distance,self.course)

# ********************** Ende GPS
******************************

# ******************** Beginn SIM808
***************************
```

```
class SIM808(GPS):
    DEFAULT_TIMEOUT=const(500)
    CHAR_TIMEOUT=const(100)
    CMD=const(1)
    DATA=const(0)

    def __init__(self,switch=4,disp=None,key=None):
        self.switch=Pin(switch,Pin.OUT)
        self.switch.on()
        super().__init__(disp,key)
        self.display=disp
        self.key=key
        print("SIM808 initialized")

    def simOn(self):
        self.switch.off()
        sleep(1)
        self.switch.on()
        sleep(3)

    def simOff(self):
        self.switch.off()
        sleep(3)
        self.switch.on()
        sleep(3)

    def simStartPhone():
        pass

    def simGPSInit(self):
        self.u.write("AT+CGNSPWR=1\r\n")
        self.u.write("AT+CGNSTST=1\r\n")

    def simGPSDeinit(self):
        self.u.write("AT+CGNSPWR=0\r\n")
        self.u.write("AT+CGNSTST=0\r\n")

    def simStopGPSTransmitting(self):
        self.u.write("AT+CGNSTST=0\r\n")

    def simStartGPSTransmitting(self):
        self.u.write("AT+CGNSTST=1\r\n")

    def simFlushUART(self):
        while self.u.any():
            self.u.read()

    # Wartet auf Zeichen an UART -> 0: keine Zeichen bis
Timeout
    def simWaitForData(self,delay=CHAR_TIMEOUT):
        noOfBytes=0
        start=ticks_ms()
        end=start+delay
        current=start
        while current <= end:
            sleep(0.1)
            noOfBytes=self.u.any()
            if noOfBytes>0:
                break
        return noOfBytes

    def
simReadBuffer(self,cnt,tout=DEFAULT_TIMEOUT,ctout=CHAR_TIMEOUT
):
        i=0
        strbuffer=""
        start=ticks_ms()
        prevchar=0
        while 1:
            while self.u.any():
                c=self.u.read(1)
                c=chr(ord(c))
                prevchar=ticks_ms()
                strbuffer+=c
                i+=1
                if i>=cnt: break
            if i>= cnt:break
            if ticks_ms()-start > tout: break
            if ticks_ms()-prevchar > ctout: break
        return (i,strbuffer) # gelesene Zeichen

    def simSendByte(self,data):
        return self.u.write(data.to_bytes(1,"little"))

    def simSendChar(self,data):
        return self.u.write(data)

    def simSendCommand(self,cmd):
        self.u.write(cmd)

    def simSendCommandCRLF(self,cmd):
        self.u.write(cmd+"\r\n")

    def simSendAT(self):
        return self.simSendCmdChecked("AT","OK",CMD)

    def simSendEndMark(self):
        self.simSendChar(chr(26))

    def
simWaitForResponse(self,resp,typ=DATA,tout=DEFAULT_TIMEOUT,cto
ut=CHAR_TIMEOUT):
        l=len(resp)
        s=0
        self.simWaitForData(300)
        start=ticks_ms()
        prevchar=0
        while 1:
            if self.u.any():
                c=self.u.read(1)
                if ord(c) < 126:
                    c=c.decode()
                    prevchar=ticks_ms()
                    s=(s+1 if c==resp[s] else 0)
                if s == l: break
            if ticks_ms()-start > tout: return False
            if ticks_ms()-prevchar > ctout: return False
        if type==CMD:
            self.simFlushUART()
        return True

    def 
simSendCmdChecked(self,cmd,response,typ,tout=DEFAULT_TIMEOUT,c
tout=CHAR_TIMEOUT):
        self.simSendCommand(cmd)
        return 
self.simWaitForResponse(response,typ,tout,ctout)

# ********************** Ende  SIM808 
***************************

# *********************** Beginn GSM 
****************************

class GSM(SIM808):

    def __init__(self, switch=4, disp=None, key=None):
        super().__init__(switch,disp,key)
        self.gsmInit()
        print("GSM module initialized")

    def gsmInit(self):
        if not self.simSendCmdChecked("AT","OK\r\n",CMD):
            return False
        if not 
self.simSendCmdChecked("AT+CFUN=1","OK\r\n",CMD):
            return False
        if not self.gsmCheckSimStatus():
            return False
        return True

    def gsmIsPowerUp(self):
        return self.simSendAT()

    def gsmPowerOn(self):
        self.switch.off()
        sleep(3)
        self.switch.on()
        sleep(3)

    def gsmPowerReset(self):
        self.switch.off()
        sleep(3)
        self.switch.on()
        sleep(3)
        self.switch.off()
        sleep(1)
        self.switch.on()
        sleep(3)

    def gsmCheckSimStatus(self):
        n=0
        a=""
        self.simFlushUART()
        while n < 3:
            self.simSendCommand("AT+CPIN?\r\n")
            a=self.simReadBuffer(32)
            if "+CPIN: READY" in a[1]:
                break
            n+=1
            sleep(0.3)
        if n == 3:
            return False
        return True


    def gsmSendSMS(self,phoneNbr,mesg):
        if not
self.simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD):
            print("SMS-Mode not selcted")
            return False
        sleep(0.5)
        self.simFlushUART()
        if not
self.simSendCmdChecked('AT+CMGS="'+phoneNbr+'"\r\n',">",CMD):
            print("Phonenumber Problem")
            return False
        sleep(1)
        self.simSendCommand(mesg)
        sleep(0.5)
        self.simSendEndMark()
        sleep(1)
        return self.simWaitForResponse(mesg,CMD)
        #return self.simReadBuffer(50)

    def gsmAreThereSMS(self,stat):
        buf=""
        # SMS-Modus aktivieren
        self.simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD)
        sleep(1)
        self.simFlushUART()
        # ungelesene SMS listen ohne Statusänderung ",1"
        #print("SMS-Status",stat)
        self.simSendCommand('AT+CMGL="{}",1\r\n'.format(stat))
        sleep(2)
        # OK findet sich in den ersten 30 Zeichen nur, wenn
        # keine ungelesene SMS vorliegt
        a=self.simReadBuffer(30)[1]
        #print(30,a)
        if "OK" in a:
            sleep(0.1)
            return 0
        else:
            # restliche Zeichen im UART-Buffer entsorgen
            self.simFlushUART()
            # erneuter Aufruf zum Einlesen

self.simSendCommand('AT+CMGL="{}",1\r\n'.format(stat))
            sleep(2)
            a=self.simReadBuffer(48)[1]
            #print(48,a)
            # suche nach der Position von "+CMGL:"
            p=a.find("+CMGL:")
            if p != -1:
                pkomma=a.find(",",p)
                #print("gefunden",a[p+6:pkomma])
                return int(a[p+6:pkomma])
            else:
                #print("CMGL not found", a)
                return None
        #print("Kein 'OK'")
        return None

    def gsmReadAll(self,stat,cnt=500):
        # SMS-Modus aktivieren
        self.simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD)
        sleep(1)
        self.simFlushUART()
        # SMS listen ohne Statusänderung ",1"
        self.simSendCommand('AT+CMGL="{}",1\r\n'.format(stat))
        sleep(2)
        a=self.simReadBuffer(cnt)
        #print("BUFFER:\n",a[1],"\n")
        return a

    def gsmFindSMS(self,stat="ALL",cnt=150):
        Liste=[]
        i=0
        p0=0
        again=-1
        self.simFlushUART()
        m,a=self.gsmReadAll(stat,cnt)
        while again == -1:
            again=a.find("OK")  # wenn OK gefunden, letzter 
Durchgang
            #print("again",again) # only for debugging
            #print("@@@@",a,"@@@@") # only for debugging
            while 1:
                merker=p0
                #print("merker: ",merker) # only for debugging
                p0=a.find("+CMGL:",p0)
                #print("While1_p0",p0)
                if p0 == -1:
                    p0=merker
                    #print("@p0_break",p0) # only for 
debugging
                    break
                p1=a.find(",",p0)
                #print("p0 und p1: ",p0,p1) # only for 
debugging
                if p1 == -1:
                    p0=merker
                    #print("@p1_break",p0) # only for 
debugging
                    break
                n=int(a[p0+6:p1])
                #print("Liste: ",Liste,"neu: ",n) # only for 
debugging
                Liste.append(n)
                p0=p1
            m,x=self.simReadBuffer(cnt)
            #print(p0,": Buffer: \n",a[p0:],"*****",x,"<<<<<")
# only for debugging
            a=a[p0:]+x
            p0=0
            if m == 0:
                break
        return Liste

    def gsmReadSMS(self,index,mode=0): # mode=1: Status nicht 
veraendern
        # SMS-Modus einschalten
        self.simSendCmdChecked("AT+CMGF=1\r\n","OK\r\n",CMD)
        sleep(1)
        self.simFlushUART()
        try:
            if mode==1:

self.simSendCommand("AT+CMGR={},1\r\n".format(index))
            elif mode==0:

self.simSendCommand("AT+CMGR={}\r\n".format(index))
```

```
            sleep(1)
            a=self.simReadBuffer(250) # Antwort=(Anzahl,
Zeichen)
            #print(a[1]) # only for debugging
            if a[0] != 0:
                a=a[1].split(',',4+mode)
                #print(a)  # only for debugging
                p0=a[0+mode].find('"')
                p1=a[0+mode].find('"',p0+1)
                Status=a[0+mode][p0+1:p1]
                Phone=a[1+mode].strip('"')
                Date=a[3+mode].lstrip('"')
                Time=a[4+mode].split('"')[0]

Message=a[4+mode].split('"')[1].strip("\r\n").rstrip("\r\nOK")
                return (Status,Phone,Date,Time,Message)
            else:
                return None
        except (IndexError,TypeError) as e:
            print("Index out if range",e.args[0])
            return None

    def gsmShowAndDelete(self,stat,disp=None,delete=None):
        SMSlist=self.gsmFindSMS(stat, cnt=100)
        while 1:
            if len(SMSlist)==0:
                print("nothing to do")
                break
            for i in SMSlist:
                response=self.gsmReadSMS(i,mode=1)
                print(i,response[0],"\n",response[4])
                if disp:
                    disp.clearAll()
                    disp.writeAt("{}.
{}".format(i,response[0]),0,0)
                    disp.writeAt(response[4],0,1)
                    sleep(2)
                if delete or
not(response[1]=='+49xxxxxxxxxxx'): #evtl. weitere PhoneNbr +
stati
                    if self.gsmDeleteSMS(i):
                        print("!!! deleted",response[1])
            first=SMSlist[0]
            SMSlist=self.gsmFindSMS(stat,cnt=100)
            if len(SMSlist)==0 or SMSlist[0] == first: break
        if disp: disp.clearAll()

    def gsmDeleteSMS(self,index):
        print("SMS[{}] deleted".format(index))
        return
self.simSendCmdChecked("AT+CMGD={},0\r\n".format(index),"OK\r\
n",CMD)
```

```
# ********************** Ende GSM
*****************************

# ******************* Beginn GPS6MV2
**************************
class GPS6MV2(GPS):
    # Befehlscodes setzen
    GPGLLcmd=const(0x01)
    GPGSVcmd=const(0x03)
    GPGSAcmd=const(0x02)
    GPVTGcmd=const(0x05)
    GPRMCcmd=const(0x04)

    def __init__(self,delay=1,disp=None,key=None):
        super().__init__(disp,key)
        self.display=disp
        self.delay=delay  # GPS sendet im delay Sekunden
Abstand
        period=delay*1000

SetPeriod=bytearray([0x06,0x08,0x06,0x00,period&0xFF,(period>>
8)&0xFF,0x01,0x00,0x01,0x00])
        self.sendCommand(SetPeriod)
        self.sendScanOff(bytes([GPGLLcmd]))
        self.sendScanOff(bytes([GPGSVcmd]))
        self.sendScanOff(bytes([GPGSAcmd]))
        self.sendScanOff(bytes([GPVTGcmd]))
        self.sendScanOn(bytes([GPRMCcmd]))
        print("GPS6MV2 initialized")

    def sendCommand(self,comnd):  # comnd ist ein bytearray
        self.u.write(b'\xB5\x62')
        a=0; b=0
        for i in range(len(comnd)):
            c=comnd[i]
            a+=c  # Fletcher Algorithmus
            b+=a
            self.u.write(bytes([c]))
        self.u.write(bytes([a&0xff]))
        self.u.write(bytes([b&0xff]))

    def sendScanOff(self,item):  # item ist ein bytes-objekt
        shutoff=b'\x06\x01\x03\x00\xF0'+item+b'\x00'
        self.sendCommand(shutoff)

    def sendScanOn(self,item):
        turnon=b'\x06\x01\x03\x00\xF0'+item+b'\x01'
        self.sendCommand(turnon)
```

## GSM - or Simsen for the ESP32

In addition to the AT commands for switching on the GPS unit and opening the UART connection on the SIM808, there are a few more for SMS operation, which I will briefly discuss. In principle, AT commands must be terminated with a \ r \ n (carriage return and line feed) so that they are recognized as such by the SIM808. The SIM808 sends back responses or results. Except for user data such as message texts, the only important thing here is whether the response corresponds to the expected one. There is therefore a gsm method that does exactly this check. The class attribute CMD (value = 1) identifies the AT command as a command. For commands, the remainder of the UART buffer is automatically emptied after the response from the SIM808 has been read in.

*simSendCmdChecked ("AT + CMGF = 1 \ r \ n", "OK \ r \ n", CMD)*
AT + CMGF = 1: switches to text mode and thus enables SMS mode

The status of SMS messages allows their selection, for example, with a list command. stat therefore contains one of the following strings: "ALL", "REC UNREAD" or "REC_READ". This value is built into the command by the format instruction.

*simSendCommand ('AT + CMGL = "{}", 1 \ r \ n'.format (stat))*
The '1' ensures that the status does not change when listing.

When sending SMS messages, the command must first be given the number of the connection. The SIM808 replies with a ">".

*simSendCmdChecked ('AT + CMGS = "' + phoneNbr + '" \ r \ n', ">", CMD)*

If ">" was recognized, the message can be sent.

*simSendCommand (mesg)*

The end of the message is announced to the other station by sending an end-of-text character (chr (26)).

In order to read an SMS message from the SIM808's memory, its index must be known. After sending the read command, the UART buffer of the SIM808 can be read out. It contains the status, date, time and text of the message.

*"AT + CMGR = {} \ r \ n" .format (index))*

Messages can also be deleted by specifying the index.

*simSendCmdChecked ("AT + CMGD = {}, 0 \ r \ n" .format (index), "OK \ r \ n", CMD)*

SMS operation is only possible after a SIM card has been recognized. The command

simSendCommand ("AT + CPIN? \ r \ n")

checks this. The SIM card must not be secured by a PIN.

Some of the GSM methods are used for internal processing and control. The main program relais.py shows the application of the methods of the class.

The application program relais.py has become a little more extensive compared to the pure GPS application from Part 2. This is due to the various examples for querying sensors and SMS message traffic. The program has a lot to offer. In detail it demonstrates:

- Time-controlled SMS (every x hours, minutes ...)
- event-controlled SMS (temperature outside of a range)
- SMS on demand (reply to an SMS)
- GPS tracker (distance exceeded or waypoint transmission)
- Transmit measured value

```
# File: relais.py
# put contents into boot.py after successful testing
# Purpose: Booting SMS-Relais Station
# Author:  J. Grzesina
# Rev.:1.0 - 2021-04-29
#********************** Beginn Bootsequenz
************************
# Dieser Teil geht 1:1 an boot.py fuer autonomen Start
#************************ Importgeschaeft
*************************
# Hier werden grundlegende Importe erledigt
import os,sys       # System- und Dateianweisungen

import esp          # nervige Systemmeldungen aus
esp.osdebug(None)

import gc           # Platz fuer Variablen schaffen
gc.collect()
#
from machine import ADC, Pin, I2C
from button import BUTTONS, BUTTON32
from time import sleep,time,sleep_ms, ticks_ms
from gps import GPS,SIM808, GSM
from lcd import LCD
from keypad import KEYPAD
from bmp280 import BMP280

#**************** Variablen/Objekte deklarieren
****************
# -------------------------------------------------------------
----
# **************** create essential objects
*******************
# -------------------------------------------------------------
----
rstNbr=25
rst=BUTTON32(rstNbr,True,"RST")
```

```
ctrl=Pin(rstNbr,Pin.IN,Pin.PULL_UP)
t=BUTTONS() # Methoden für Buttons bereitstellen

i2c=I2C(-1,Pin(21),Pin(22))

k=KEYPAD(35)

d=LCD(i2c,0x27,cols=16,lines=2)
d.printAt("SIM808-GSM",0,0)
d.printAt("RELAY booting",0,1)

sleep(1)

g=GSM(switch=4,disp=d,key=ctrl)
print("***********************")
g.simGPSInit()
g.simOn()
g.mode="DDF"

b=BMP280(i2c)
#sleep(10)

timeFlag    =0b00000001 # Intervallsteuerung
distanceFlag=0b00000010 # Streckenalarm
tempFlag    =0b00000100 # Tempraturalarm
orderFlag   =0b00001000 # SMS Anforderung per SMS
Trigger=orderFlag # Hier die Flags der Dienste eintragen

distLimit=100 # Entfernungsrichtwert in Metern
tempMax=25 # *C
tempMin=20 # *C

heartbeat=0 # Actionblinker

timeBase=3600*1 # Zeitintervall ins Sekunden
distanceBase=3600*1#3600*2 # Sekunden
tempBase=20#3600*2 # Sekunden
timeEnde=time()+5
distanceEnde=time()+5
tempEnde=time()+5

# Die Dictionarystruktur (dict) erlaubt spaeter die
Klartextausgabe
# des Verbindungsstatus anstelle der Zahlencodes
#*******************Funktionen deklarieren
***********************

def timeJob():
    t=str(b.calcTemperature())+"*C"
    p=str(b.calcPressureNN())+"hPa"
    print("sende Wetterdaten")
    g.simFlushUART()
```

```
g.gsmSendSMS("+49xxxxxxxxxxx","Wetterwerte:\n{},\n{}".format(t
,p))
    return  True

def tempJob(temp):
    t=str(temp)+"*C"
    return g.gsmSendSMS("+49xxxxxxxxxxx
","WARNUNG!!:\n{},\nTemperatur nicht im Limit!!".format(t))

def positionJob(dist):
    g.gsmSendSMS("+49xxxxxxxxxxx ","Entfernung: {},\n
Positionswerte\n{},\n{}".format(dist,g.Latitude,g.Longitude))
    print("Position gemeldet")

def orderJob(Nachrichten):
    # decode message
    if "wetter" in Nachrichten.lower():
        print("Wetterdaten angefordert")
        return timeJob()
    # do jobs
    # send results
    # return True # if no errors occured

def getDistance():
    rmc=g.waitForLine("$GPRMC",delay=1000)
    if rmc:
        try:
            g.decodeLine(rmc)
            if g.Valid == "A":
                try:
                    gga=g.waitForLine("$GPGGA",delay=2)
                    g.decodeLine(gga)
                    g.printData()
                    if d: g.showData()
                    entfernung=g.calcLastCourse()[0]
                    print("Distanz: {}".format(entfernung))
                    return entfernung
                except:
                    g.showError("Invalid GGA-set!")
        except:
            g.showError("Invalid RMC-set!")
# ******************   Funktionen Ende
***********************

print("Check SMS")
g.gsmShowAndDelete("REC_READ",d,delete=True)
g.gsmShowAndDelete("ALL",d,delete=False)

# *********************   Main Loop
**************************
while 1:
    getDistance()
    if d:
        heartbeat+=1
        if heartbeat % 2:
            d.writeAt("X ",14,0)
        else:
            d.writeAt(" "+g.Valid,14,0)

    if (Trigger & timeFlag) == timeFlag and time() >=
timeEnde:
        timeJob()
        timeEnde=time()+timeBase

    if (Trigger & distanceFlag) == distanceFlag and time() >=
distanceEnde:
        g.calcLastCourse()
        w=g.distance
        print(w)
        if w > distLimit:
            positionJob(w)
            ## naechsten Befehl aus kommentieren für
Kreisfläche
            g.storePosition()
        distanceEnde = time()+ distanceBase

    if (Trigger & orderFlag) == orderFlag:
        L=g.gsmFindSMS("REC UNREAD")
        if L:
            Nachricht=g.gsmReadSMS(L[0],mode=1)[4]
            print("neue SMS gefunden", L)
            orderJob(Nachricht)
            print("Job erledigt, mache jezt Kaffepause")
            sleep(5)
            g.gsmDeleteSMS(L[0])
            sleep(2)

    if (Trigger & tempFlag)==tempFlag and time() >= tempEnde:
        t=b.calcTemperature()
        #print("Temperatur: {} ## {}".format(t,tempEnde))
        if t < tempMin or t > tempMax:
            tempJob(t)
        tempEnde=time()+tempBase
        #print(tempEnde)

    # Job-Schleifenende
    if ctrl.value() == 0: break
    sleep(0.1)
```

**Note**:
So that you also get messages on your cell phone, please use your own cell phone number instead of + 49xxxxxxxxxxx.

There is a tax flag for each job. this activates the function. All functions except for SMS on demand have an additional time block in addition to the tax flag. This means that, for example, after the temperature has been exceeded, text messages are not sent continuously until the value is within the range again. The time-out is specified in seconds, but can be stretched almost at will using the appropriate factors.

In event mode, a sensor is queried at fixed time intervals. A message is only sent if the sensor value does not meet the specifications.

A type of event mode is also the SMS message about GPS positions. Of course, many more are possible than the two options shown. When the distance flag is activated, the waypoint mode is preset. If you comment out the saving of the last position, the system reacts to any distance beyond the specified distance, at fixed time intervals, as described above.

The purely time-controlled mode sends the result of a measurement independently of a sensor value. By default, the measurement results of the BMP280 are accessed here, representing any other sensors.

With SMS on demand, code words can be sent to the ESP32 via SMS. Unread SMS corpses are deleted with every restart. Then the system informs about other messages still in memory.

In the job loop, incoming messages are checked. The job function must decode the content and, if necessary, initiate appropriate actions or carry out measurements. Actions that act directly on the SIM808 and, for example, delete all SMS, are also conceivable. In any case, the mail that triggered the action is also automatically deleted. This operating mode is preset.

Further messages are waiting in the input buffer of the SIM808 and are processed one after the other. The waiting times are very important (bold). If they are left out or set too short, SMS are sent out continuously because the next loop pass finds the message and processes it again.

```
if (Trigger & orderFlag) == orderFlag:
    L=g.gsmFindSMS("REC UNREAD")
    if L:
        Nachricht=g.gsmReadSMS(L[0],mode=1)[4]
        print("neue SMS gefunden", L)
        orderJob(Nachricht)
        print("Job erledigt, mache jetzt Kaffepause")
        sleep(5)
        g.gsmDeleteSMS(L[0])
        sleep(2)
```

As you can see, the possible uses are very diverse. In addition, nobody but you can use this control without authorization, unless you put the telephone number of your SIM card on the notice board, together with all your control codes and the mobile phone. Because even access from other phone numbers besides your own is blocked in the program.

So, and so that everything starts autonomously when the ESP32 is switched on, you simply have to copy the program into the boot.py file and upload it to the ESP32, restart, that's it. If something goes wrong afterwards and the ESP32 can no longer be addressed by Thonny or another terminal program, then simply pull the emergency brake, which aborts the program. It is the RST button on the LCD keypad that you press until the REPL prompt >>> appears in the terminal. You can now make changes.

I hope you enjoy texting with your ESP32. In the next episode we will allow our controller to cheat together with the SIM808. One or more radio modules that can be equipped with ESP8266 or ESP32 will expand the controller's radius of action. The transmission protocol will be UDP, which is easily sufficient for our purposes and is much faster and simpler than TCP. Stay tuned!

Have fun implementing the project!

More download links:
PDF in deutsch
PDF in english