```
#!/usr/bin/env python

# standard imports
import struct
from binascii import hexlify
from time import sleep

from ate.instrument import Instrument

# imports from pip
import serial
import serial.tools.list_ports
# mannualy import hwgrep protocol for pyinstaller issues
import serial.urlhandler.protocol_hwgrep
import crcmod.predefined
import png

DEFAULT_TIMEOUT = 5

SENSOR_WIDTH = 32
SENSOR_HEIGHT = 32

VID = 0x0483
PID = 0x5740


class TeraRangerEvoThermal(Instrument):

    def __init__(self):
        super().__init__()
        self._crc = crcmod.predefined.mkCrcFun('crc-32-mpeg')

    def __repr__(self):
        return 'TeraRanger Evo'

    def connect(self, port=None):
        wait = 0
        if port:
            self._conn = serial.Serial(port)
        while self._conn is None:
            try:
                self._conn = serial.serial_for_url('hwgrep://{:04x}:
{:04x}'.format(VID, PID))
            except serial.SerialException:
                if wait > DEFAULT_TIMEOUT:
                    raise
                sleep(0.1)
                wait += 0.1
        self.setTimeout(DEFAULT_TIMEOUT)
```

```
    def setTimeout(self, timeout):
        self._conn.timeout = timeout

    def command(self, cmd):
        self._conn.write(cmd)

    def read(self, count):
        return self._conn.read(count)

    def activate(self, state):
        if state:
            # activate
            cmd = b"\x00\x52\x02\x01\xDF"
        else:
            # deactivate
            cmd = b"\x00\x52\x02\x00\xD8"
        self.command(cmd)
        resp = self.read(4)
        if resp[2] == 0xFF:
            raise Exception(
                    "{} error: NACK recevied after sending command {}"
                    .format(type(self).__name__, hexlify(cmd)))

    def get_temps_once(self):
        self.activate(True)
        raw_data = self.read(2070)
        self.activate(False)
        crc_result = self._crc(raw_data[2:2066])
        crc_bytes = struct.unpack("H"*2, raw_data[2066:])
        crc_found = (crc_bytes[0] << 16) + crc_bytes[1]
        if crc_result == crc_found:
            temp_data = struct.unpack("H"*1024, raw_data[2:2050])
            temps = [round(x/10.0 - 273.15, 2) for x in temp_data]
            return temps
        raise Exception(
                "{} error: bad crc during temperature read"
                .format(type(self).__name__))

    def get_max_temp(self):
        return max(self.get_temps_once())

    def write_png(self, filename, mintemp=None, maxtemp=None,
depth=8):
        temps = self.get_temps_once()
        if mintemp is None:
            mintemp = min(temps)
        if maxtemp is None:
            maxtemp = max(temps)

        def greyscale(temp):
```

```
            return round((2**depth-1)*(temp - mintemp)/(maxtemp -
mintemp))

        rows = list()
        for x in range(SENSOR_WIDTH):
            col = list()
            for y in range(SENSOR_HEIGHT):
                value = greyscale(temps[x * SENSOR_HEIGHT + y])
                # constrain to range allowed by depth
                col.append(min((2**depth-1), max(0, value)))
            rows.append(col)

        with open(filename, 'wb') as f:
            p = png.Writer(SENSOR_WIDTH, SENSOR_HEIGHT,
greyscale=True, bitdepth=depth)
            p.write(f, rows)


if __name__ == '__main__':
    ir = TeraRangerEvoThermal()
    ir.connect()
    for i in range(10):
        print(i, ir.get_max_temp())
    ir.write_png('thermal.png')
    ir.disconnect()
```