Open CPV Solar Tracker Control v0.1

June 1st 2021

© 2021, Ruediger F. Loeckenhoff, MIT License

Summary: Highly precise and universal sensor-based solar tracker control with minimal hardware and full astronomic tracking for Concentrating Photovoltaic Systems.

Keywords: Solar, energy harvesting, robotics, sustainability, home automation, monitoring, weather

1   About this project

I have a long professional experience in the field of High-Concentration-Photo-Voltaic (H-CPV) cells and systems. These systems need very precise dual axis sun tracking and I am not aware of any applicable open source solution. So, I decided to learn Arduino and actually invested months of my private time in this project. It was also a lot of fun. I really hope that it will be picked up by the CPV community and implemented in full size solar trackers.

The basic idea is, that the hardware can be very simple, if you use a compass to follow the calculated sun path. Whenever there is sun, a light sensor takes over to allow a very precise tracking better than 0.1°.

The other CPV trackers I have seen so far used limit switches and encoders which I believe is a bad idea. Such external sensors require cables which are expensive and which can break when the tracker is moving. A defective limit switch can destroy the tracker by making it move beyond limits. Furthermore, limit switches are hardly compatible with the operation in the tropics, because here the tracker needs to be able to turn both ways from the morning position (east) which requires an azimuth range of more than 360°.

My solution has all components in a small control box which is tilting and turning together with the solar generator. It is very easy to build since the motor controller is plugged onto the Arduino Uno as a shield and the soldered connections reduce to a few jumper wires between the boards. The unit is running on an external 24V power supply to make it electrically safe. A mix of motors with 12 and 24V rating can be used if you run the 12V motor at no more than 50% PWM duty cycle.

The sophisticated serial interface allows adapting the control to the tracker mechanics and motors. E.g. I developed it on a 3D-printed table-top tracker, I tested it on a real 11 m^2 steel tracker and the motor controller should have enough power for a 50 m^2 tracker. The code is well documented and I hope for comments and improvements. In particular I would appreciate help designing an all in one circuit board that would hopefully be picked up by Asian suppliers for Open Source mass production.

Apart from than, it is also a great and well documented robotics project and of course it also works for conventional PV – yet my heart beats for CPV.

I am used to scientific papers an books, so I decided to put everything into one big pdf file rather than splitting it into many files. Don't be repulsed by the length. I tried to put more or less anything I know about CPV trackers into this document and you may skip some paragraphs. The interconnection of the breakout boards is actually very easy to do.

2   A Brief Introduction to CPV and Solar Tracking

2.1   Concentrating photovoltaics CPV

In CPV-Systems lenses or mirrors focus the sunlight on tiny solar cells with ultra-high efficiency. Typically, the concentration ratio is in the range of 300-2000 times the energy density of direct sunlight. Only direct sunlight can be focused and the focus moves away from the solar cell when the system is not perfectly aligned to the direction of the direct sunlight. The aligned (left) and the misaligned (right) cases are sketched in Fig. 1. Typically, a CPV solar tracker should have a precision of 0.1° or better.
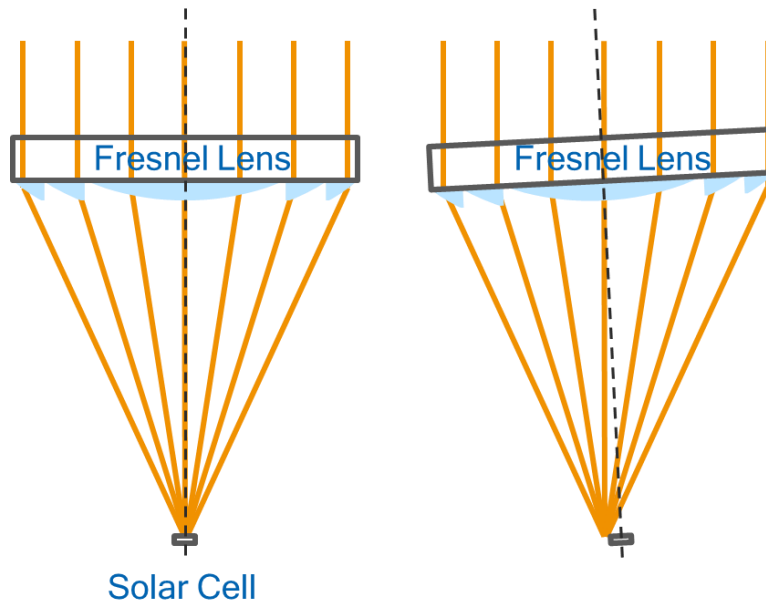
Fig. 1: Aligned and misaligned CPV module

2.2 Mechanics

This project is not primarily about building solar tracker mechanics. But I should explain what kind of solar tracker is suitable for this control and also share some of my knowledge. The present version of the software assumes an "azimuth drive" with a vertical axis which is fixed to a foundation (in case of a real solar tracker) or e.g. a wooden base plate (in case of a table top demonstrator). In other words: the azimuth drive adjusts the compass direction. On top of this azimuth drive there is a second "elevation drive" with a horizontal axis. It adjusts the pointing direction of the tracker above the horizon. We will call this angle above the horizon "elevation". The direction of the horizontal elevation rotation axis is changing, as the vertical axis is turning.

E.g. 0° azimuth is north. When the sun is coming up in the east, the elevation is 0° and the azimuth is 90°. When the sun is close to Zenith the elevation is ˜90° and the azimuth is changing rapidly.

The simplest and cheapest configuration for a demonstrator that I can think of consists of 2 12V 2 RPM motors as shown in Fig. 2, left. Depending on your toolbox you can use 3D-printed parts or wood and glue to build a solar tracker around these two motors. In case of the azimuth drive it is a good idea to add some friction, e.g. with felt pads, to suppress the backlash and prevent oscillations. For the elevation drive you might use some counter-weights.

In the description of the EEPROM parameters below you will see that you can set the speed of the up and down movements for the elevation separately. This allows you to have equal up and down speeds and step sizes even if the elevation is not balanced and under load.

You can see the cross-shaped shading beam in front which is casting a shadow on 4 LEDs which act as light detectors. This kind of tracking sensor is very easy to make and it will work. During the development of this control I have been working with a very compact imaging tracking sensor which I have developed for my employer and which allows a tracking precision of 0.01° or better for clear weather conditions[1]. Please contact me, if you should be interested.
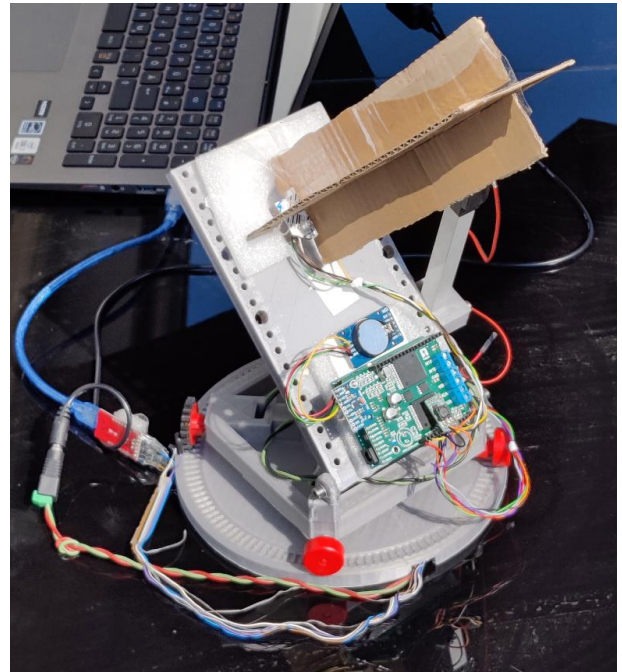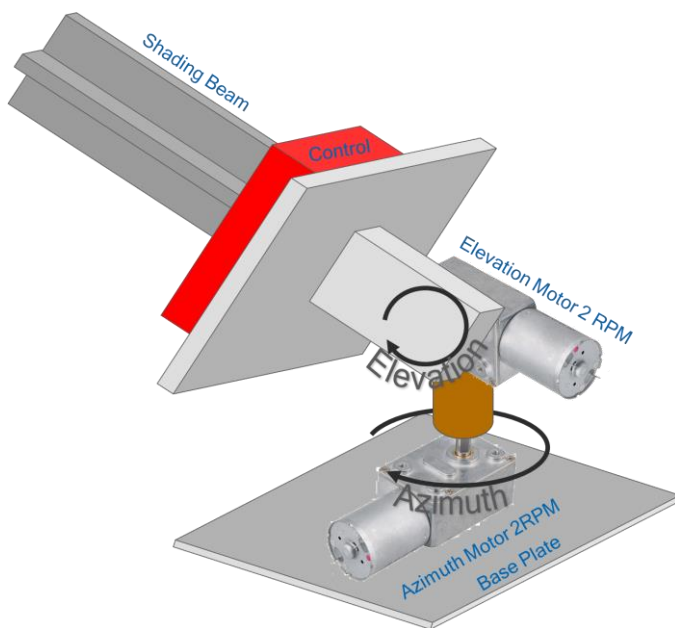
Fig. 2: Simple table top tracker and my table top tracker made from 3D-printed parts and Fischertechnik.

I have built the tracker on the left-hand side of Fig. 2 and it works. However, I wanted test hardware which is similar to real solar trackers that usually use a slew drive for azimuth and a linear drive for elevation. So, I also built the tracker on the right hand side. It uses Fischertechnik wheels for azimuth and a small linear drive for elevation. The shading beam is made from cardboard because weight is an issue for the little linear drive. In this case the tracker is powered with a USB-12V boost converter cable which proved to be very handy.

I claim that this control is so universal, that it could track a toy tipper truck to the sun which is running in circles and moving its platform as in Fig. 3. It would be great if someone could prove me right.

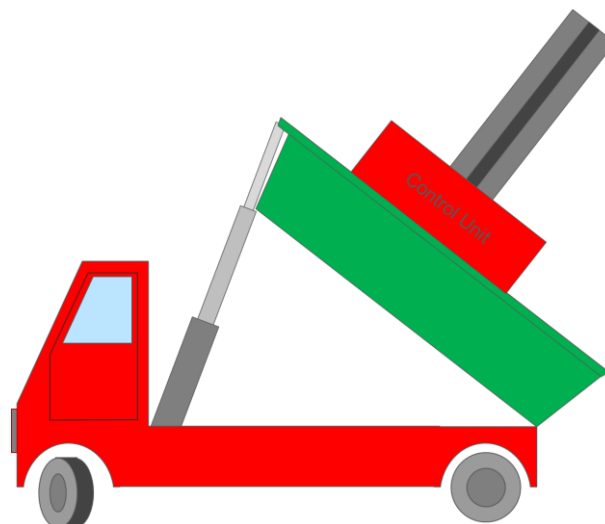

Fig. 3: Toy tipper truck tracker. Please, would someone build it for me?

There are a lot of servo-based Arduino solar tracker projects. These are great for educational purposes but impractical for big industrial trackers. Servos require potentiometers and it is hard to find a suitable place for them within the tracker mechanics. They are also not all to reliable when exposed to moisture.

As far as real applications are concerned, I strongly believe in a 3 wheeled solar tracker with a winch for the elevation movement. Used car wheels are a resource which is readily available anywhere in the world. They can carry tons and they can run on uneven ground such as gravel. Even a badly worn car wheel is still good enough for a solar tracker. It will not run more than 10 meters a day or 3.6 kilometres per year. If it should finally burst you just need to bring your jack and another old tire for repair. Anyone can do that anywhere in the world.

The winch is working against gravity. This means that it is preloaded and thus works with zero backlash.

Fig. 4 is a picture of a patent application that I have filed and which is now discontinued and free for anyone to use. If you want to build this tracker, please have a look at Fig. 5 and get in contact with me such that I can share some more thoughts.

DE 10 2019 004 468 A1   2020.12.31
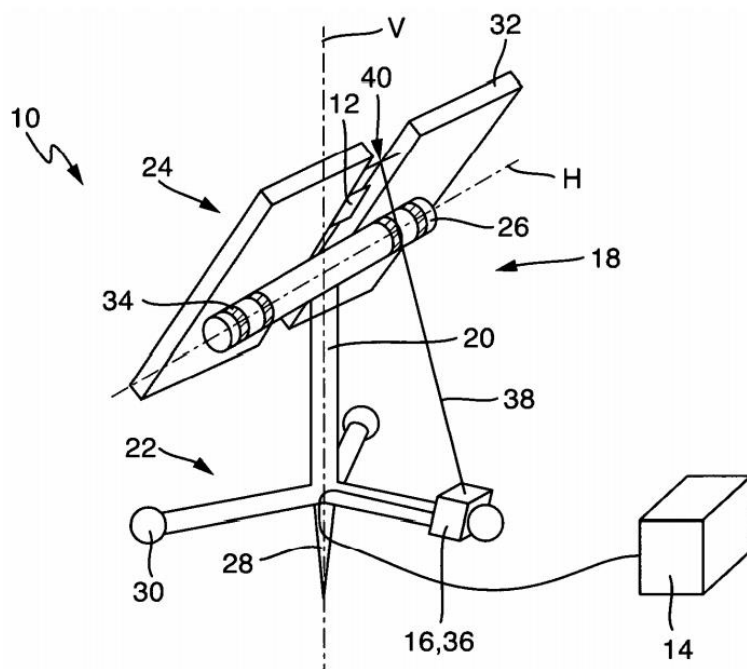
Anhängende Zeichnungen

Fig. 4: Wheel and winch tracker according to a discontinued patent application.

Wind load is always an issue for solar trackers. They need to go to stow when the wind is faster than e.g. 20 m/s. The winch may be designed in such a way, that the wire is wound around the wheel and pre-tensioned by a weight. It thus can slip in one direction but not in the other. Even without electrical power the wind will move the generator and the wire will always keep tensioned just like when a sailor tightens a rope around a bollard to pull a huge ship closer to the quay wall with each wave. Please see the sketch below.

Actually, this tracker concept made me understand that a tracker control should not depend on precise mechanics. It should orient itself by magnetism, gravity and of course the rays of the sun. Thus, the dual axis tracker mechanics can be whatever, as long as they are able to move in small steps and as long as the precision is not compromised by backlash.
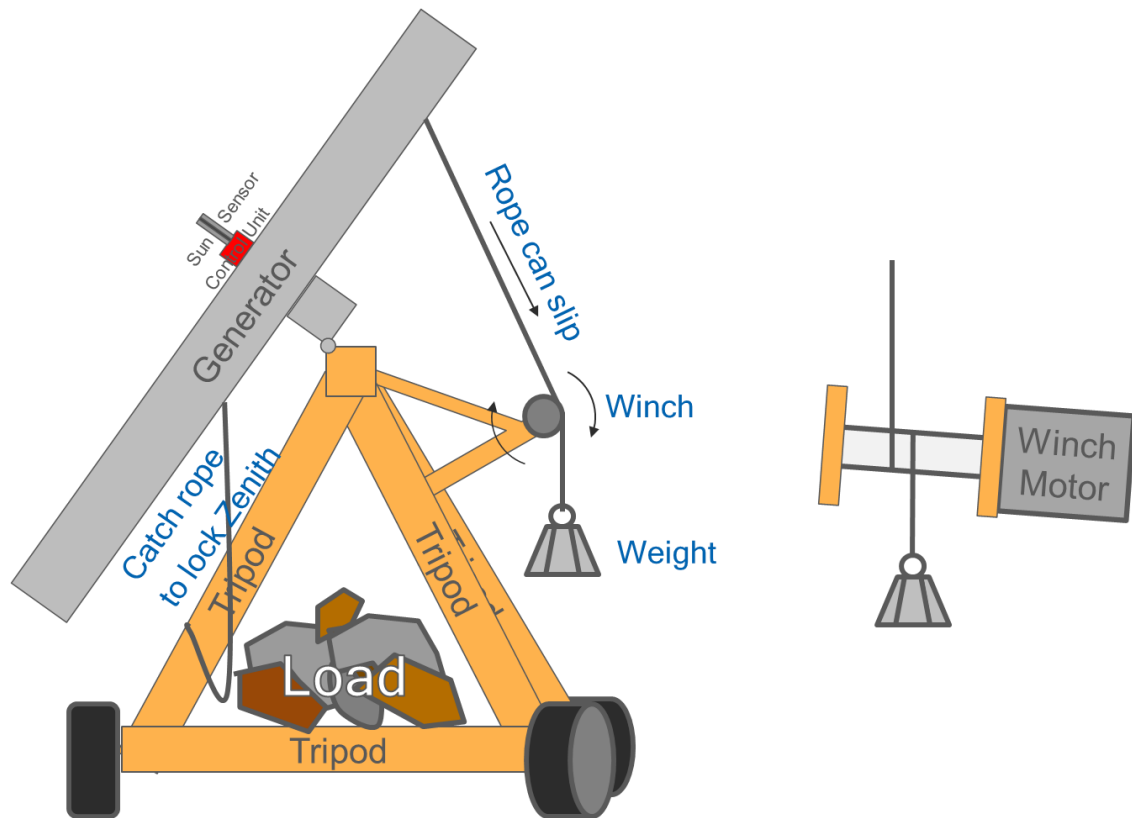
Fig. 5: One of my preferred arrangements

2.3 Astronomical vs Sensor Tracking

There are two basic approaches to solar tracking. You can calculate the sun position from time, date, longitude and latitude and move the tracker to the calculated position. This is called **astronomic tracking**. For a precise tracking the mathematical formulas can be quite complex. Here we can use a simplified version [3]. On the other hand, there is **sensor-based tracking** where light detectors sense the direction of the sunlight. In this project we follow a **hybrid tracking** approach where the tracker moves to the calculated direction whenever there is no sufficient light for sensor-based tracking. As soon as the amount of light on the detector surpasses a certain level, the control switches to sensor-based tracking mode.

Astronomic Tracking with encoders: The most common way to assume the calculated orientation is to drive each axis against a limit switch and count encoder steps from there [4]. This can take a lot of time since trackers tend to be slow and you need for the encoder calibration each time you power on or reset the tracker. You will also need to set the encoder-steps per degree for an exact calculation. If a limit switch fails, the tracker can self-destruct if you have not added an overpower protection or an automatic shutoff for out of range tracker orientations.

Astronomic Tracking with servos: This is a very common and simple approach for Arduino projects. However, you will not find a servo big enough to move a 50 m² tracker.

Compass based astronomic tracking: Actually, this is the main idea behind this project where a 9 axis compass measures tilt and compass direction. The drives are operated by simple 12 or 24V DC motors.

The tilt is calculated from the gravitation in x and z. This tilt sensing is quite precise and the gravitational field is very reliable.

Getting the compass direction from the magnetic field is much trickier. The magnetic vector usually is not parallel to the ground, it is weak and the magnetic sensors have quite a large zero drift. Furthermore, solar trackers are commonly made out of magnetic metal which distorts the earth magnetic field and the DC motors add more errors. Therefore, I had to find and program a calibration routine which corrects for all of those errors. You can find it described below in the section which concerns the first start-up.

3   Components for the control unit and housing

The control is based on an Arduino Uno with ATMEGA328 chip, a magnetic compass with tilt sensor, a real time clock together with a sun sensor made from 4 clear, red LEDs. The control box moves together with the solar generator. This allows all sensor components to be integrated in a small box. No external limit switches or encoders are required.

Component list

- Arduino Uno   x1
- 24V Power Supply (IP68 for Outdoor Operation)   x1
  Alternatively: USB-12V boost converter for testing only.
- Polulu Dual VNH 5019 Motor Shield, (or clone) [6]   x1
- Maxim Integrated DS3231 breakout board [7]   x1
- MPU9250 breakout board [8]   x1
- 24V-7V Step down converter   x1
- Clear red LEDs   x4
- Resistors e.g. 470k   x4
- Jumper wires black, brown, grey, red,
  orange, yellow, purple, white   x1
- Set of 9 nuts and 3 bolts M4x (galvanized or stainless)   x1
- SP13 6 pin built in socket (female)   x1
- SP13 6 pin connector (female)   x1
- SP13 6 pin connector (male)   x2
- Cable feed through   x4
- IP68 cable connectors   x4
- USB-FTDI serial converter   x1
- Push button for reset   x1
- Wind sensor   x1

Housing and mounting

- Hammond 1590T housing, 120.5x80x59mm, or similar   x1
- 150x100x2 mm^3 Aluminum plate   x1
- Silicone sealant   x1
- Silica-Gel sachet   x1

- 100 mm piece of 5mm tube   x1
- Piece of polymer cloth   x1
- M2x10 screws, 2 nuts each, 2 washers each   x2
- M3x12 screws, 2 nuts each   x3
- M4x40 screws, 3 nuts each, 3 washers each   x3

Tools
- Cordless drill
- Bit set
- Set of metrical drills
- Metal saw for the base-plate
- Conical drill for 12 mm holes in the housing

Mechanics for demonstrator
- 12V 2RPM Wormgear Motor   x2
- Tools, glue and wood, 3D-printer or whatever you like best.

Apps and Online Services
- Arduino IDE 1.8.7 (others are not tested but will probably work) [5]

Time
- 2 hrs. for ordering all the components
- 2 hrs. soldering
- 3 hrs. preparing the housing
- Lots of time for building and testing your solar tracker.

4   Electronic Assembly

Solder the long interconnector pins to the Dual VNH-5019 board. This is necessary for the original shield by Polulu. Some of the clones already have the pins attached. I had a bad experience with cheap clones. A capacitor blew up at 24V and another shield had a defective chip. So, make sure to get high quality components.

Next, use an M2.5 screw and nut to mount the MPU 9250 circuit board on the dual VNH5019 shield and fix it with silicone.

Make the connections shown in Figs. 7&8 with jumper wires on the backside of the Dual VNH5019 shield. Make sure that you keep the soldered connections close to the board such that the pins can still be used for the plugged-on connection to the Arduino Uno. The best method is shown in Fig. 6.
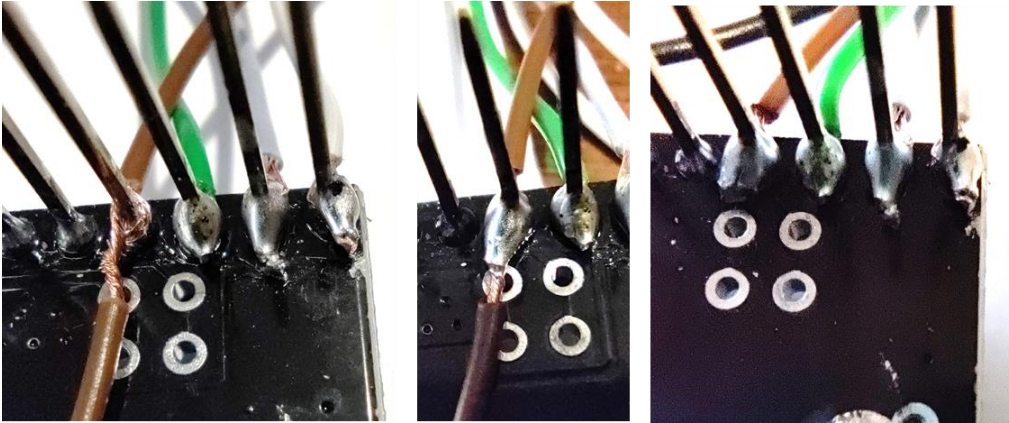
Fig. 6: Method for making connections to long pins: Pierce and twist the braid, add solder, cut or use both wires.

Solder two pins to the Vout and GND pins of the DC-DC adapter and set it to 7V output voltage. It plugs on top of the dual VNH5019 with two pins for (Vout-Vin) and GND. The Vin-pin of the DC-DC-adapter is connected to the 24V Vin of the Dual VNH5019 shield. It is a good idea to add soldered jumper wires for (Vout-Vin) and GND to have a stable power supply at Vin-pin of the Arduino.

Make the wires long enough. Some parts will sit on the top and some on the bottom of the housing. If the wires are too short you will have a hard time opening the housing and working inside.

6 wires run to the SP13 socket which will be built into the housing. The colour code is maintained on both sides of the plug. The plugged connection is not absolutely necessary if you just want to test the circuit in sunny weather without a housing. In this case just leave it away and go directly to the serial-usb adapter (FTDI YP-05) and the wind sensor.

Presently, we don't need 5V on the SP13 socket. I yet think it is a good idea to have it in place in case we want to use a wind sensor that needs a power supply. You may also use a wind sensor that produces a voltage output if you cut the "A1" trace on the dual VNH5019 shield, connect the purple wire to A1 and do the necessary changes in the code to do an analogRead(A1) instead of the pin2 interrupt routine. A0 is required for the overcurrent protection of the elevation motor.
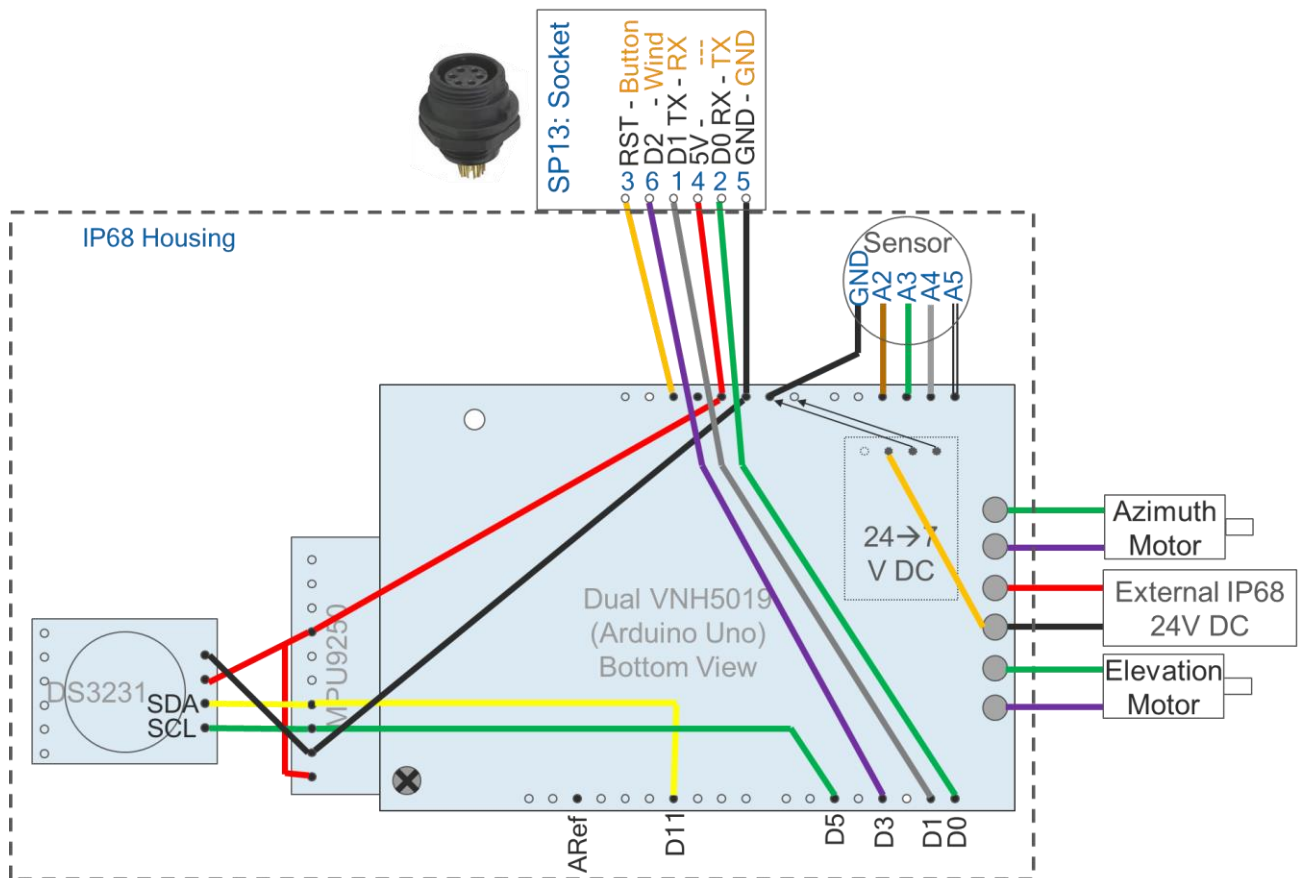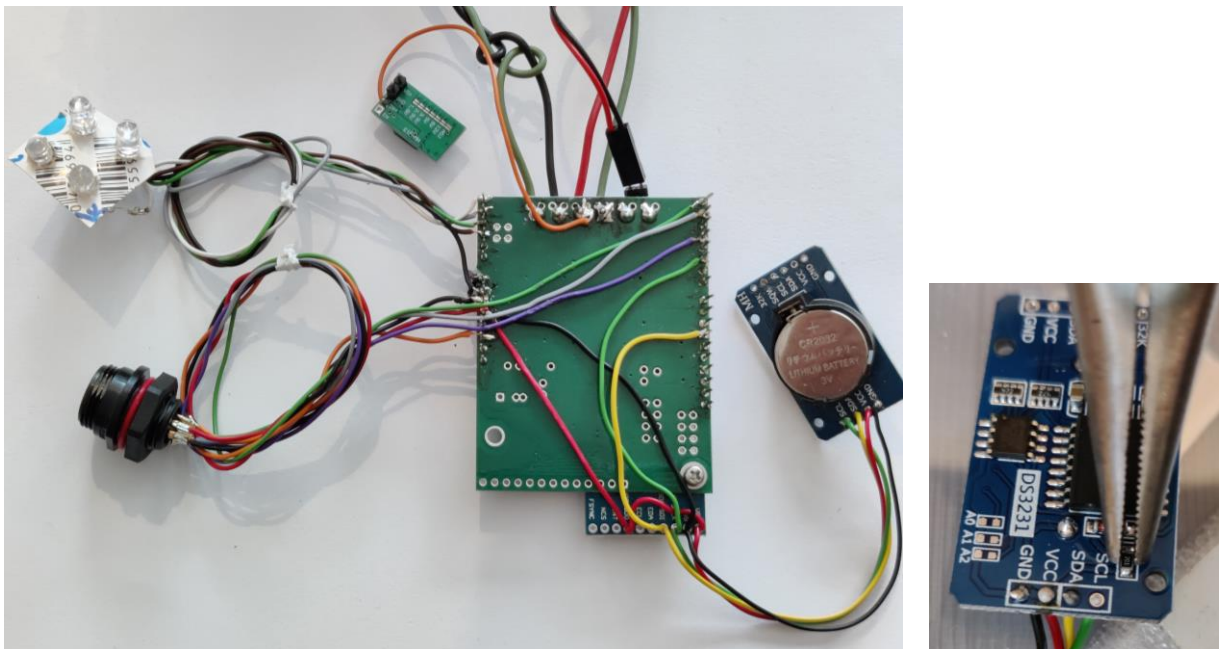
Fig. 7: Wiring Diagram

Fig. 8: Actual circuit. 4-Sensor-Unit is improvised for table-top-testing. Right: how to remove the dangerous charging resistor.

Remove the charging resistor of the DS3231 breakout board with pliers. Otherwise it overloads the battery. And of course, you will need to add a battery.

Only Rx, Tx and GND are connected to the FTDI serial-USB adapter. Do NOT connect 5V-VCC since competing power sources can damage the circuit. The adapter is powered by USB. Additionally, you need to connect a push button between the orange wire (Arduino Reset) and GND. You will need this to upload the code or restart the tracker.

I am assuming a wind sensor which uses a reed relay to short circuit the purple wire (Arduino D2 interrupt pin) in pulses when the wheel is turning (Fig. 9). The control is counting pusles per second to determine the wind speed.
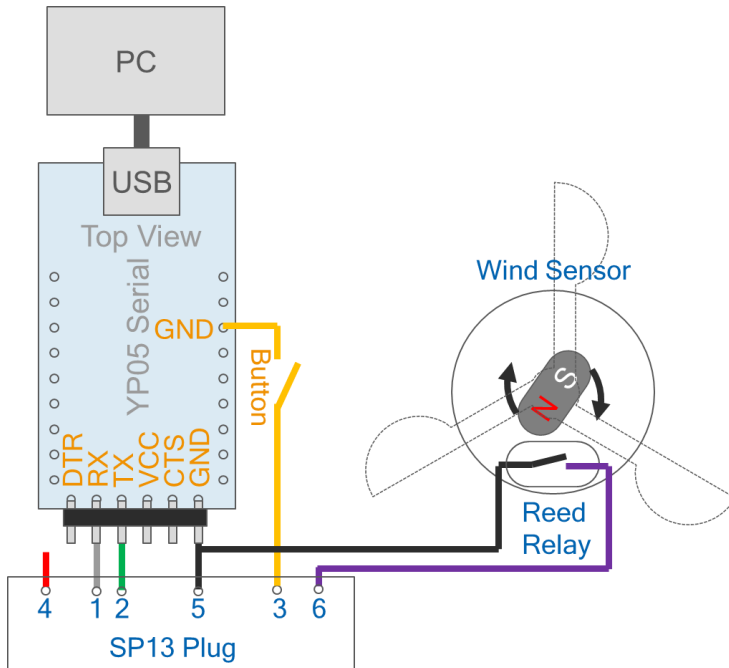

Fig. 9: USB-serial-converter and reed relay wind sensor

For the sun sensor we are using clear red LEDs as tiny solar cells. They need to be set into holes in the sunny side of the housing as seen in Figure 10. Drill the holes and set the LEDs into them from the top, to use the housing as a soldering jig. Finally, they need to be set into the housing from the inside. Wait until you are completely done with all electronics and metalworks before you glue them in place an seal them.

In open circuit the LEDs will go up to about 1.4V, but we want to use them as linear sensors for sunlight so we need to connect shunts to adapt them to the 1.1V range of the analogue pins with internal reference. In my case 470k proved to be the best choice. You will have to find the right resistive value for yours. LEDs with a red housing probably wont work because they block too much of the sunlight.

We are using a 200 mm high cross beam to shade the LEDs and turn them into a sunlight direction sensor. The LED lenses reduce the acceptance angle and the sensitivity to stray light. 20° acceptance angle should be more than enough to find the sun from astronomical tracking. If you should find out that the acceptance angle is too small, you might grind off the lenses. This will also result in a higher sensitivity to stray light.

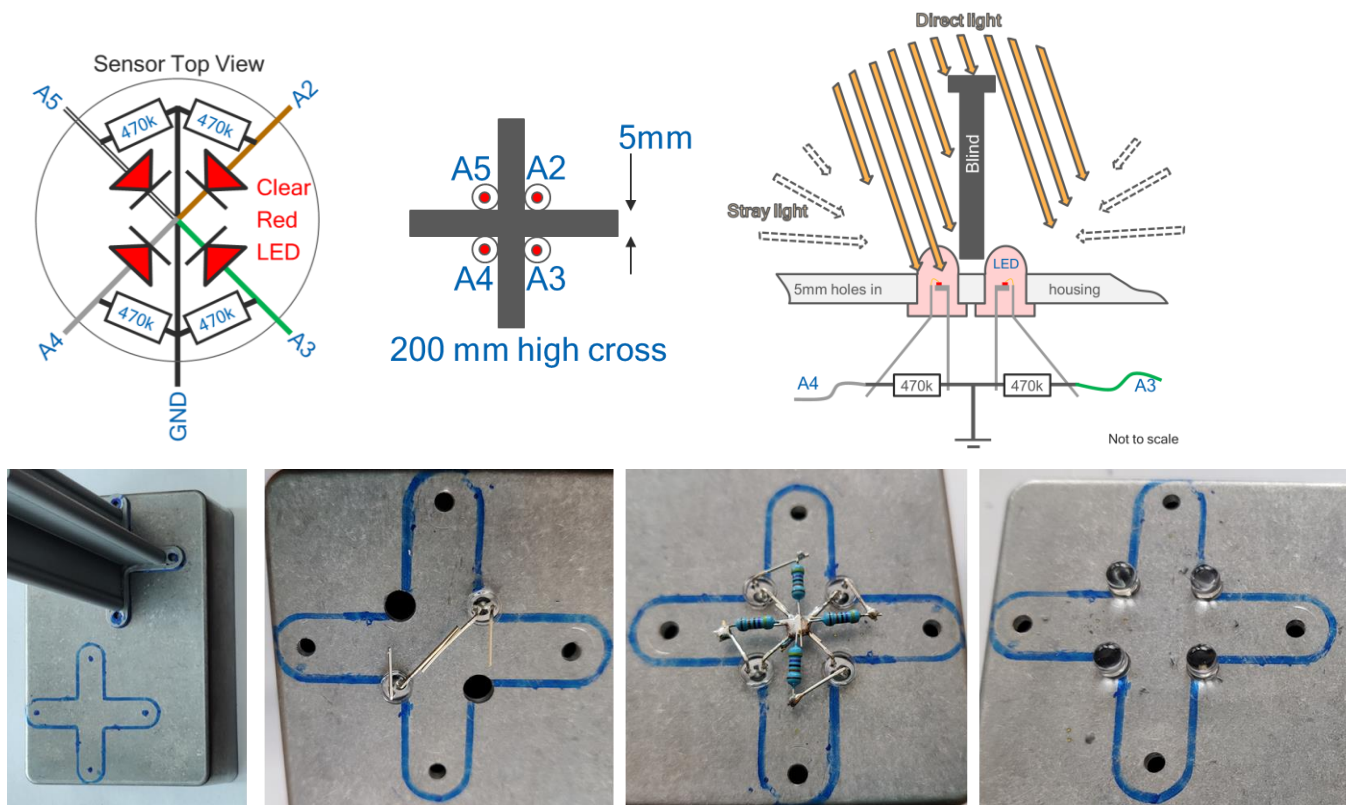Figure 11 shows all external connections.

Fig. 10: Sun sensor electronics and shading beam.
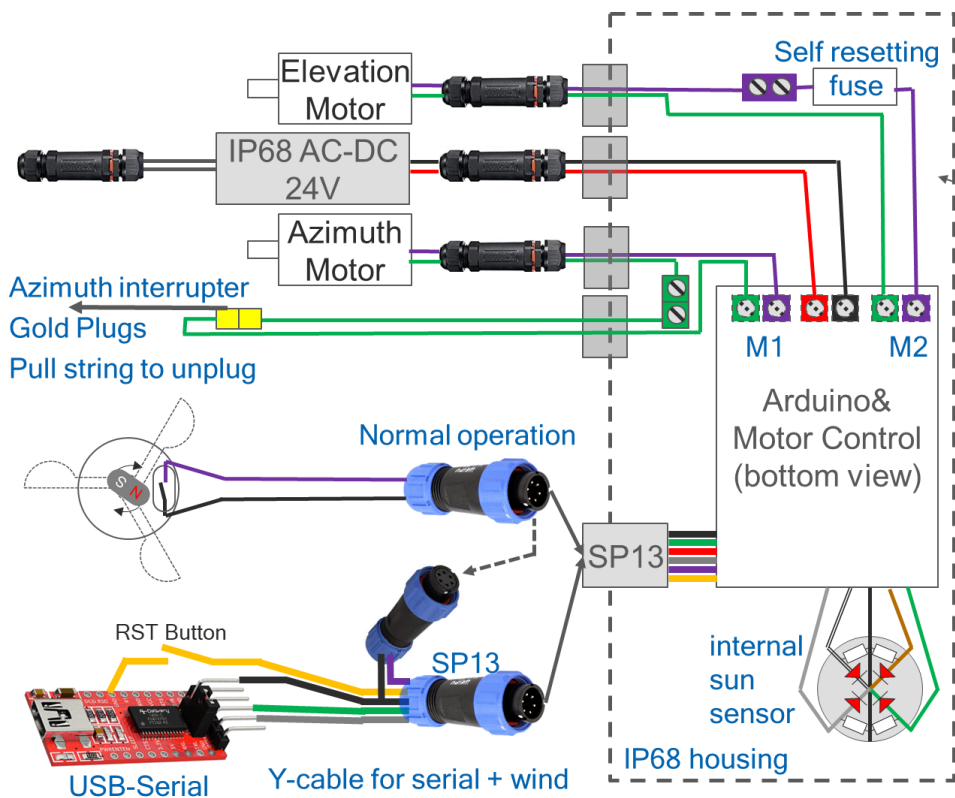

Fig. 11: Sketch of the sun sensor electronics and shading beam.

The external wires to the DC motors and the IP68 24V AC-DC power supply go through waterproof cable feed throughs. Furthermore, I have added IP68 cable connectors. They come very handy since they allow you to work with short wires in your workshop and connect them to longer wires in the field. Avoid complicated work outdoors such as soldering whenever you can. Your soldering iron can cool down in the wind and prevent you from making a clean soldered connection.

One extra wire runs to the gold plug which acts as a pull string interrupter. The string should run in parallel to the wires and be shorter, such that the plug unplugs before wires get damaged. The optional self-resetting fuse in the elevation circuit is redundant to the overcurrent limit "u" (see Appendix). If you trust software, you could leave it away.

If you hope to have the tracker running for decades, make sure to use UV proof cables. You can use any kind of DC-Motors, as long as they won't exceed the 24V, 12A rating of the VNH5019. You can use motors of a lower voltage rating with reduced PWM duty cycles. If you get close to 12A you need to improve the cooling.

All external signals go through the SP13 6 pin socket. During normal operation you will usually have the wind sensor connected. When you want to program and test the control, you will want the USB-serial interface (AKA FTDI) as well as the wind sensor. So, you need to make a Y-cable with a SP13-6 plug to the control and a SP13-6 socket to the wind sensor.

The USB-serial interface also needs an extra push button connected to GND on the interface and to Reset on the Arduino. Resetting is necessary for uploading code.

The LEDs are set into the housing in holes which are arranged in a square. We don't want to have the lens-shaped tips, so grind them of after gluing in the LEDs. Silicone is the preferred glue. Use lots of it to make a good seal. The cross-shaped shadow-beam is mounted on top of the LEDs, such that all 4 LEDs will receive sunlight only when the beam is pointing to the sun.

Mind that you see the Arduino and motor control from the bottom side with the M1 connectors on the left. The internal sun sensor is shown from the "sunny side" with the white wire in the top left.

5   Metal works housing and external cabling

I am assuming that you are using a Hammond 1590T die cast aluminium housing. You may also use any other housing of the same size. Drill 12 mm holes as shown in Fig. 12 and insert the SP13 connector and the cable feed throughs. The shading beam is attached with 4pcs M3x10 and 4pc nuts M3, see Fig. 10. One nut per screw keeps a distance between the housing and the beam to let water and dirt escape.

Fig. 12: Metal works, cable feed throughs and SP13 connector.

Making a really hermetic housing is no good idea. Temperature and pressure will change and high pressures may build up. Once there is a leak it will suck in raindrops and the inside of the housing will get wet. It is much better to have it breathing in a controlled way. For this, I drilled a 12 mm whole into the base plate and fit a cable feedthrough inside. A 5mm tube runs through the hole. It is bent to always point down, whether in Zenith or morning position. I pushed a piece of polymer cloth into the tube as a filter against dust and I put a bag of silica gel into the box to buffer the moisture. It doesn't take the moisture away for always, but it can absorb more water as it cools down and thus prevents condensation at low temperatures. Fig. 13 shows the baseplate with the breathing pipe. The Arduino is fixed with 3 screws M3x10 and distance holders in M3 threads in the base plate. You can also use nuts instead of threads, then the screws should be at least 12 mm long. The 3 pcs M3x40 with 3 nuts each are for mounting and adjustment.
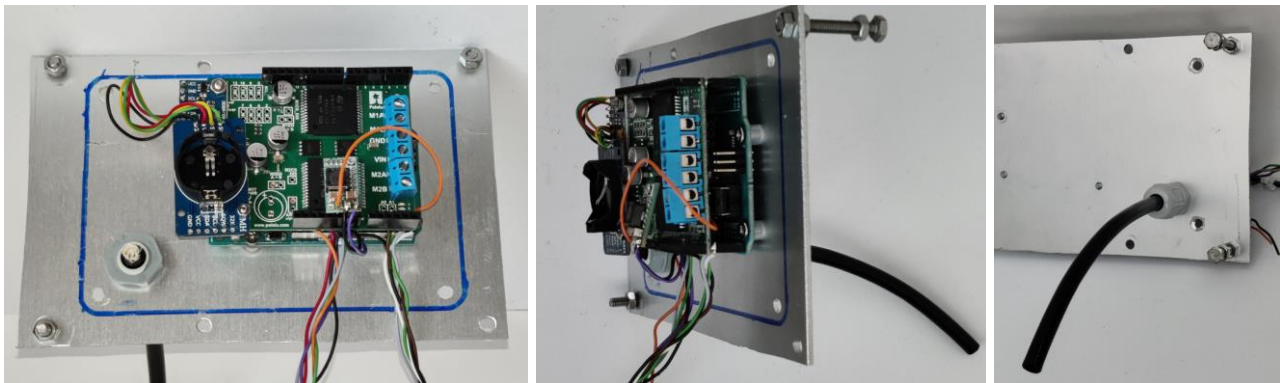

Fig. 13 left: bottom plate of housing, right: housing with electronics and silica gel.

Once you have the metalworks done, you can add the final soldered connections to the sun sensor. For inserting the SP13 socket into the side wall it was necessary to cut the wires. As you can see in Fig. 12, you should leave pieces of the colored wires on the socket. Thus, you can replace them bit by bit without a chance for errors.

Fig. 14 shows the whole arrangement with only the motor, interrupter and power cables missing. The back plate needs to be flipped up to assure the correct orientation of the compass. Add a sachet of silica gel for moisture control. After tests insert silica gel and seal everything with neutral silicone, including the backside of the sun sensor and all threads in the housing.

Fig. 14: Full assembly, only the power cables and silica gel are still missing

6 Upload and test the code

Create a folder OS_CPV_TRACKER in your Arduino directory and copy all code files into that folder. Start the Arduino IDE 1.8.7 and open OS_CPV_TRACKER.ino. The other tabs 010_header, 011_def … should open automatically.
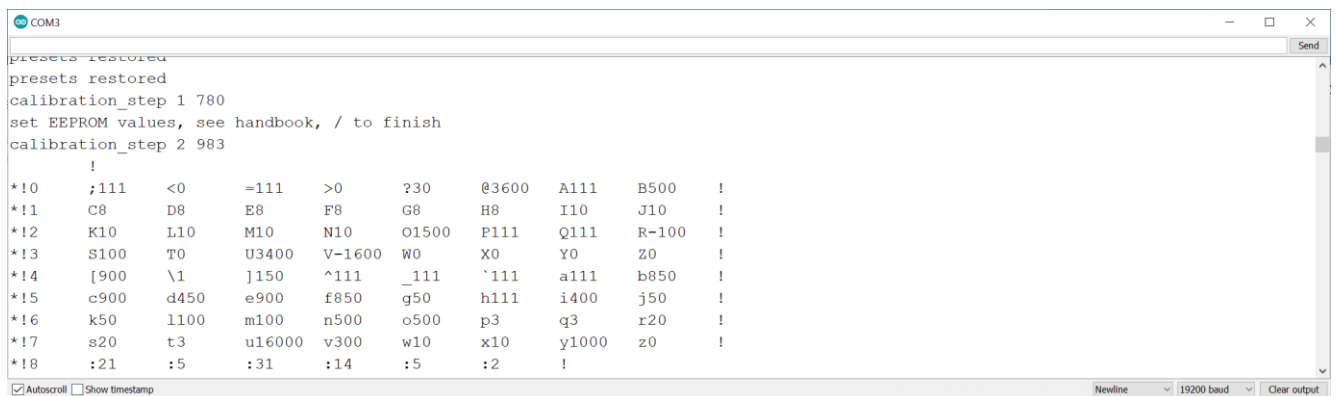
This is a good chance to test the serial connection. Connect the USB-Serial adapter to a USB port on your computer and to the circuit. Don't connect the 5V pin of the Arduino to the 5V-Pin of the USB-Serial adapter. The tracker needs its own 12V or 24V power supply. The adapter should show up as a COM port. Chose Board:Arduino/Genuino Uno.

Open the serial monitor at 19200 bauds.

Press upload and press the reset pushbutton you have attached to the USB-Serial adapter as soon as the Arduino IDE says "uploading". You may need to try several times to get the timing right. Once the LED on the USB-Serial adapter is flickering, you know that the code is uploading. Soon the Arduino IDE will state "done uploading".

After Reset, you should see some printouts on the monitor, in particular "calibration step 1", which shows you, that the control has detected a first start and wants to calibrate. Ignore this for the time being and
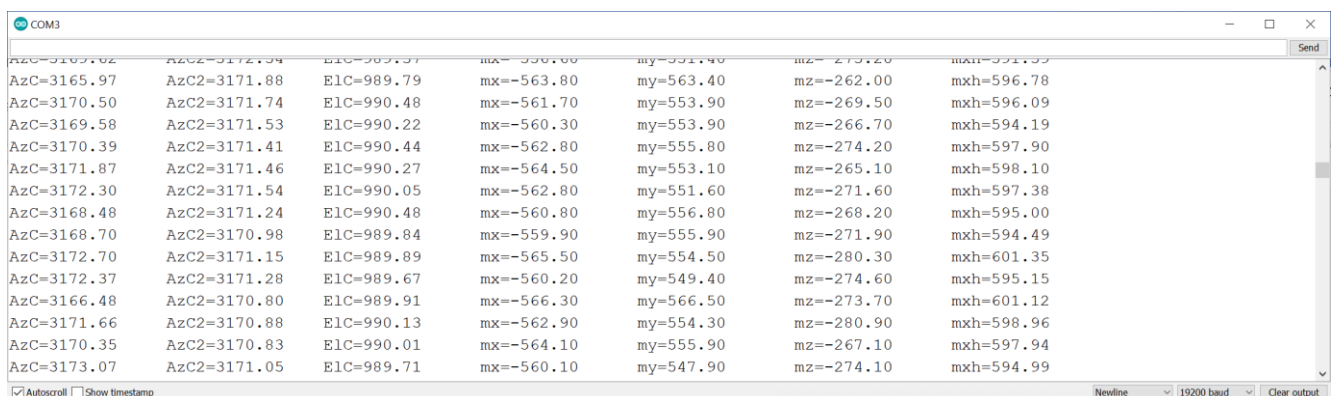
- Type "~" to display parameters in EEPROM, which have been restored to factory pre-sets automatically during the first start.

Fig. 15: Plot of factory presets

- If you enter "~" several times, you should see the real time clock in line *!8 counting up. You can set the correct time now, as explained in the section "time and date".

- Type "%" to toggle the display of the magnetic readings and elevation. If you can see values similar to the ones below you know that all components work correctly.

Fig. 16: Plot of magnetic readings

- AzC is the azimuth compass reading in 0.1° and will have an arbitrary value before calibration. AsC2 uses a floating average to reduce noise. EIC is the elevation measured with the tilt sensor. mx, my, mz are the magnetometer readings and mxh is the horizontal component of the magnetic field in x-direction.

7   Orientation for tests

You may want to test the electronics on a table top tracker before taking it to the field. In any case the compass and the sun sensor need to be mounted on the "photovoltaic generator" and move in azimuth and elevation with the tracker.

The orientation of the sun sensor and the electronic compass matter. The code will work correctly, if you mount these parts as shown below seen from top or rather from the "sunny side". If you mount them in other directions, you might need to adapt the code (swap x-y-z, and/or negate some values).
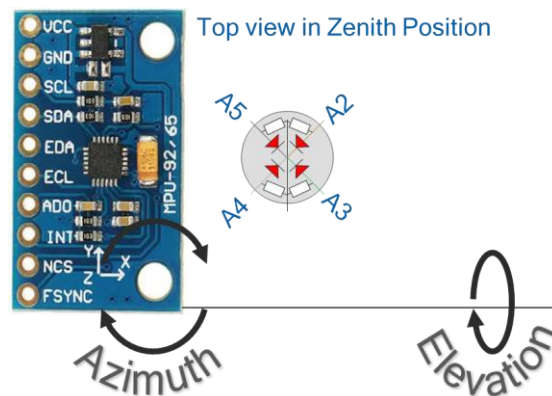
Fig. 17: Correct orientation of the compass chip

In order to align the CPV generator to the sun, you must be able to align the sun sensor to the generator. Since the sun sensor is an integral part of the control unit, the whole control unit needs to be aligned. You can do this with three long rods with adjustable nuts that fix the plate with the control box to the generator as shown in Fig. 18. If you just want to do a demonstration on a table-top-tracker without a CPV module or if you want to track a flat plate PV generator you can skip this alignment.
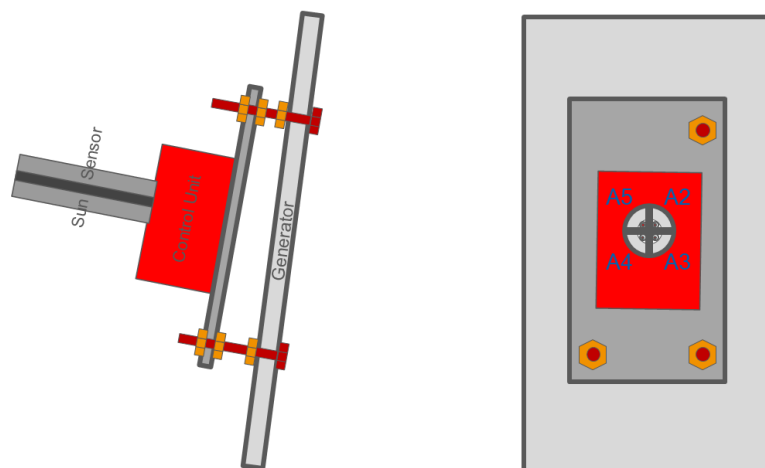
Fig. 18: How to adjust the control unit with three bolts

8   Calibration

For the calibration the control box should be mounted on a functional tracker which can freely rotate by more than 360° in azimuth.

I strongly recommend that you should first test your control unit on a small and safe benchtop tracker before taking it to a full-size tracker outside. This way you can also get yourself accustomed to all functions. Full size solar trackers can be quite dangerous. They can self-destruct when the move beyond limits, parts may detach and fall down. AC cables may get damaged and give you an electric shock. Most importantly, these trackers are usually strong enough to crash your hand or head. It doesn't necessarily help, that they are so slow. You may get yourself in a situation, where you cannot pull your hand or head out or stop the tracker and it will just keep on going and going and going. Beware, I refuse all responsibility as stated in the MIT license below. Always work in company when you are dealing with heavy machinery.

Make sure that the cables can follow the tracker movement. To be on the safe side, they should allow the tracker to rotate several times before they rip.

Establish a serial connection.

For the outdoor testing of a big tracker it is best to have the USB-Serial converter close to the control unit. I have been using a 20m active USB cable to bridge the distance to a comfortable spot from where I could control the tracker. Instead of the reset button I had connected the reset pin to a single long wire that ran in parallel to the USB cable. To do a reset, I just needed to touch a USB port of the Notebook computer to pull Reset to ground. This is "makeshift" but worked very well. Of course, you can also go to the tracker (be careful!) and press the reset button. It's up to you.

As explained above, the control will automatically go into calibration mode at the first startup. In particular it will prompt you:

"Set EEPROM values, see handbook, type / to finish."

8.1   Calibration_step 1: Set your parameters

- Hard Reset with the push-button.
- The Arduino serial monitor is prompting:

calibration_step 1

set EEPROM values, see handbook, / to finish
calibration_step 2 406

- You can reset the tracker to factory defaults by entering "#" and confirming with "y".
- You can display a list of the user defined parameters in EEPROM by sending "~" over the serial interface (see Fig. 15)

- You can copy and safe this list in a text file.
- You can restore a full line by copying it from the text file to the entry line of the Arduino serial monitor and pressing send or the return key.
- You can edit a line before sending it.
- Line *!8 shows the present date and UTC-time in the format yy mm dd hh mm ss. Copy this line to the serial monitor, correct it as necessary and confirm with send or return.
  In particular you need to make the following entries now:

  **** : Number of double steps in elevation. E.g. 1 will yield 2 steps between three positions.

  [: Upper limit of the elevation range. This needs to be lower than the mechanical limit.

  ] : lower limit of the elevation range. This needs to be higher than the mechanical limit.

  C, D, F, G: duty cycles for the elevation drive. Reduce them to a max. of 8, if the elevation motor.

  E, H : Same for azimuth
- For a complete list with explanations refer to the Appendix.
- When you are done, enter "/" to go to the next step.

8.2   Calibration_step 2: Find sun

The serial monitor will prompt:

Find sun: U D L R c_ontinue s_top f_inish

calibration_step 3

I assume that you run the control on top of a full size CPV tracker with CPV-modules installed. The control unit is sitting on the solar generator between the CPV-modules. In the precise sun sensor mode, the control unit aligns itself to the sun. So you first need to make sure that the CPV modules have the correct orientation to the control unit with its sun sensor.

Adjusting all CPV modules to the sensor would be very tedious. It is much better, to adjust the control unit on the generator. Since the compass sits inside this very same control box, we first need to calibrate the sun-sensor-based tracking. That means, that you need direct sunlight for this step!

Your job is to orient the tracker orientation close enough to the sun to let the sun sensor tracking take over. "U" will move the tracker up by one step, "D" will move down, "L" will move left and "R" will move right (clockwise). "c" → "R" will continue in the clockwise direction indefinitely, until you type "s" for stop.

At some point, the sun sensor should take over and the tracker will orient itself to the sun. Have a look inside the CPV modules (use dark glasses) to check the alignment. Now adjust the alignment of the control unit to the generator by moving the nuts on the rods up or down. The generator will follow in the opposite direction. You can stop, when the generator is rea-

sonably well aligned. Fine adjustments can be done later. Just be aware, that these later adjustments will also affect the astronomical, compass based tracking.

If the control unit doesn't switch to sun sensor mode, you may need to adjust the parameters "i" to "o", in particular the total sensor signal for direct sun "i".

If you are just testing your unit on a benchtop tracker, you don't need to adjust the mechanics but you may want to use this chance to test the tracker movements and decide whether every motor is running in the right direction at the right speed.

When you are done, enter "f" to finish.

8.3 Calibration_step 3: Correct elevation up limit

Now the control will prompt:

corr 70° u U d D, f finish

calibration_step 4

where 70° is replaced by the elevation up limit that you have entered as parameter "[". Use U or down for big up or down steps and u or d for small steps. These commands change the "elevation up correction" parameter "R" for astronomical tracking in the EEPROM and the control unit will react accordingly. If you are not happy with the precision or if the elevation steps up and down without ever coming to a rest, you may need to reset the tracker and try other values for "calibration elevation error" "t" in step 1. You can skip step 2 by typing "f" to return to step 3.

Use a water level and maybe a wedge (e.g. 20° in the example) to measure the elevation angle.

8.4 Calibration_step 4: Correct west

The control unit prompts:

corr West r R l L, c ontinue, s top, f

calibration_step 6

Move the tracker to west by typing "c" "L" for a continuous counter clockwise movement. Stop with "s", do stepwise corrections with "R" and "L" for clockwise or counterclockwise moves. Other serial inputs may cause hang-ups → reset. "~" is allowed. A rough orientation to west by a few degrees will be enough. In this step you are just defining the turnaround point for the automatic calibration. The calibration precision is not affected.   Finish with "f".

8.5 Calibration_step 6: Correct elevation down limit

You will see a prompt where 30° is replaced by your lower elevation limit:

correct 30° u U d D f

calibration_step

30° is replaced by the elevation down limit "]" that you have defined in step1. Proceed like in step 3.

8.6   Calibration_step 8: Start automatic calibration

Now you are done and the automatic calibration starts. The output of the serial interface allows you to analyze the calibration especially in case it doesn't work as expected. It starts with:

calibration_step 8 56367

starting automatic calibration

calibration_step 9 71320

29

28…

The mz neutral correction temp_mz is measured and it is written to EEPROM at the end of the calibration routine. Now, the tracker moves up in steps to measure the magnetic readings that define the turnaround points during the 400° azimuth swings.

When the tracker has reached the uppermost elevation position it will start with a short counterclockwise move and then performs a 400° clockwise swing followed by a move down, a 400° counterclockwise swing and so on until it reaches the lowermost position. During this procedure it shows the compass readings for analyses. The last prompt is something like:

mxh<469.16

my<1103.22

mxh>-219.61

my>424.91

mxh_n=124

my_n=764

mxh_r=344

my_r=339

ms/deg=93

complete --> restart

The first 4 lines show the maximum and minimum values of the horizontal x and y magnetic readings. These values are converted to mxh_n and myh_n, which are the neutral values and the ranges mxh_r and myh_range.

The tracker uses a floating average to improve the precision of the azimuth measurement, even when it is moving. For this correction we need the speed of the azimuth movement given in milliseconds per degree. In this case, a 360° turn will take 37.8 seconds. This is the pretty fast tabletop tracker that I have shown you in the "mechanics" section.

8.7 Calibration explained

I want you to understand what happens during the automatic calibration. Below you can see a graph of the magnetic y "my" reading and the mx_horizontal reading, which is calculated as the horizontal component of the mx-mz vector. I have set the number of elevation steps to its minimal value, which is "\=1". Since "\" is the number of double steps, we have two steps in elevation or 3 positions. The maximum elevation "[" is set to 700*0.1° = 70° and the minimum elevation is set to 300*0.1° = 30°.

The graph Fig. 19 shows the magnetic readings for the >360° azimuth swings at different elevations in different colors.
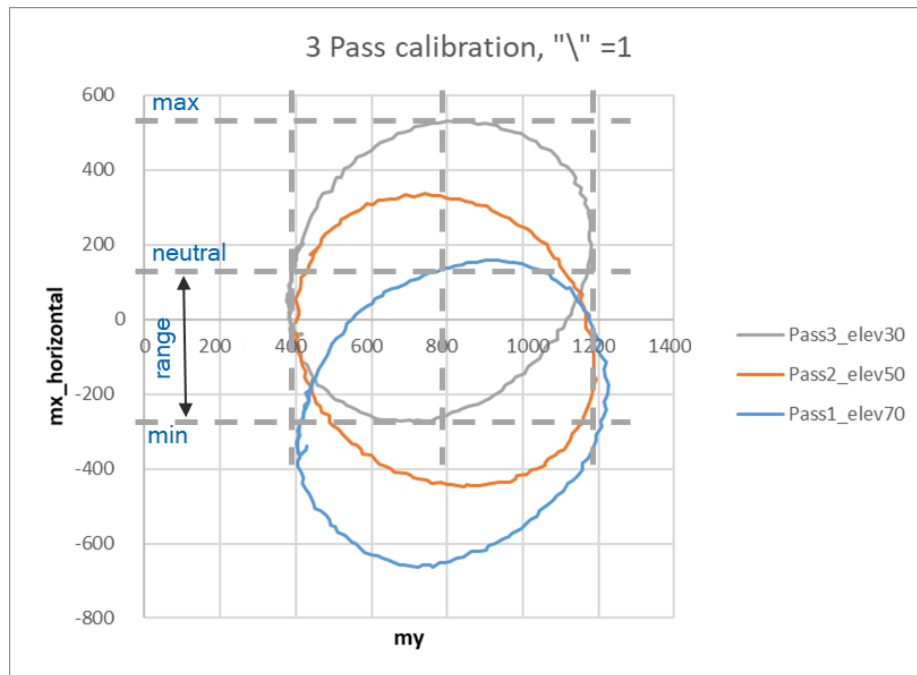

Fig. 19: How to adjust the control unit with three bolts

Of course, the Arduino Uno doesn't have enough memory to record such graphs. But it can record the maximum and minimum magnetic readings for my and mx_horizontal. I have shown this with dashed lines. The neutral position is the average of max and min and the range is half of the difference. During normal operation, we are subtracting the average and we are scaling by 1/range. This approximates the readings of my and mx_horizontal to a circle of radius 1 around the origin for each of the discrete elevations, such that we can calculate the azimuth. For elevations in between, I introduced a linear interpolation of the neutral positions and the range.

You will notice, that the neutral my position is pretty far off zero, while mx_horizontal is approximately zero. This is because mz has already been set to zero in the 30° west position in step 6 and mx in the 70° west position in step 8 of the calibration. Big offsets of mx and mz would make the azimuth calculation very sensitive to elevation so I needed to eliminate them before the azimuth calibration runs.

These readings were taken with my benchtop tracker. You can see that the passes constitute ovals rather than perfect circles. I guess this is because of steel parts in my lunch table which create an inhomogeneous magnetic field. For a big installation you won't have that, as long as you don't decide to put the tracker on steel rails as a foundation. Yet, it is a good idea to choose a place for the control box which is not to close to the motors and which has a sym-

metrical arrangement of steel parts around it. During my tests on an 11 m^2 steel tracker I have put it centered in the upper half of the generator as in Fig. 20.
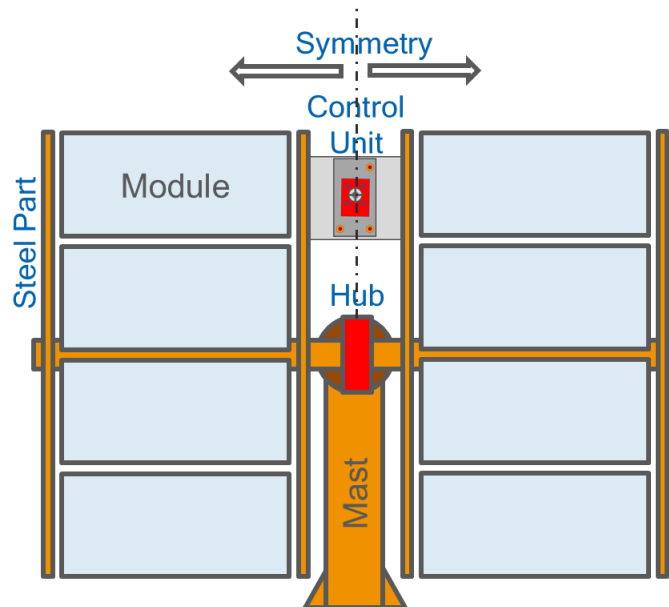

Fig. 20: Symmetric mounting position on a steel tracker far away from the motors

9   Operation

There are only a few commands to operate the tracker. You can find them in the appendix. Most manipulations are rather done by changes of the user defined EEPROM parameters. Probably you will first want to test the astronomical tracking. So first make sure that the time is right. Enter "~" and check line *!8 of the serial output against UTC (not local time). Also set or check longitude "W" and latitude "X". Meanwhile the tracker will already move to the calculated azimuth position. Cover the sensor to make sure that it is not going to sensor mode. While the sun is shining you will see whether the direction is realistic. If not, you can play with parameter "T" to introduce a correction of the compass direction.

Now you can move to different positions by changing the cleaning position "c", "d" and going to this position with "$1". You can move up and down for one azimuth position. An error of a few degrees is normal. The tracker may readjust slightly as it is moving up or down.

If you are not happy with the result, you may attempt to recalibrate with more positions. Usually that should not be necessary since the tracking sensor covers about +/- 20°. The astronomical tracking just needs to be precise enough to keep the tracker within this error margin.

It's fun to type "1:<" to get into turbo mode and see how the tracker would move during a day, but much faster. You will have to set the clock again afterwards.

If you are happy with all that, remove the cover from the sensor. It should find the direct sun. You may adjust the parameters "i".."o" for the sensor sensitivity and "F".."N" for the motor step parameters. High sensitivity at high hysteresis together with small steps at low duty cycles is preferred for precise and efficient tracking.

If the tracker is running in the wrong direction in sensor mode, you may have made a mistake in the cabling or you may have rotated the sensor by accident. This can either be corrected in the code or better with a soldering iron.

Monitor your tracker for a couple of days to see whether the astronomical errors are building up. Based on this experience, you can decide about the number of days ">" that may pass before the automatic recalibration.

10   More to be done

1. Improved sensor tracking to be on sun on average. Presently wind oscillations will make the tracker run ahead. → OK, using floating average, still needs elevation dependent threshold levels or automatically adjusted thresholds.

2. Extensive tests on sun. Several months.

3. Serial communication through RS485 for field control.
   → Add RS485 level shifter. Tracker IDs. Adjust Serial Routine.

4. Replace external wind sensor by internal detection of wind-oscillations.

5. All in one circuit board.

Not all applications will require all of the features above. Tracker fields will receive the go to stow signal through the RS485 bus, thus individual wind sensing is not necessary. We have 20% of 32 kB left. Plenty of room.

11   Appendix

11.1 EEPROM Parameter List and Commands List

This paragraph explains all user definable parameters and how to enter them in the serial monitor of the Arduino IDE. All parameters are signed integers. Degrees are expressed in 0.1° steps. E.g. an elevation of 900 means that the tracker is facing the Zenith at 90°. All parameters have limits and the software will automatically correct entries which are out of limit to the closest allowed value. There are also presets which can be restored with "#" → "y".

I recommend printing out the table below as a separate page, preferably on cardboard and maybe laminated. Computer screens are not always easy to read when you are working outside. The color code defines blocks with similar parameters, e.g. parameters that affect the motor movements in light red.

There are three ways to enter parameters into the EEPROM.

1) You can either type "~" which will display the whole parameter block. Copy a complete line to the serial input line and change the parameters as required. Hit return. This will change 8 parameters at the same time. However, there are predefined limits. If your new value is out of limits, the control will automatically save the closest allowed value (lower or upper limit).

2) You can type in a relative change of a single parameter. E.g. "10T" will increase the azimuth correction T by 10 which is 1°.

3) Or you can enter 15° as an absolute value by typing "150:T" where ":" marks an absolute value. The reversed order "T150" will not result in any change because it is a relative change of 0 and "150" is discarded. Entering "T", with or without a change, will first display the start position of the byte in the EEPROM (50) and the new value of T (e.g. 15). "350:T" will display a value of 300, since 300 is the hard coded upper limit of T. Below I will explain the parameters in detail.

EEPROM Parameters List

*!0	; -32k..32k 1234 after calibration	< 0..1 Turbo Mode	=111	> 0..1000 Days since calibration	? 0..1000 Days for calibration	@ 0..32767 Count for RTC correct	A111	B 200..20k Brake cycles	!
*!1	C 1..16 Up duty cycle	D 1..16 Down duty cycle	E 1..16 Left/right duty cycle	F 1..16 step up duty cycle	G 1..16 Step down duty cycle	H 1..16 Left/right duty cycle	I 1..255 Ramp cycles up	J 1..255 Ramp cycles dwn	!
*!2	K 1..255 Ramp cycl. left/right	L 1..5k Step up plateau	M 1..5k Step down plateau	N 1..5k Left/rigth plateau	O 1k..5k Cont. move plateau	P111	Q111	R -100..100 Elevation up correct.	!
*!3	S -100,,100 Elevation down corr.	T -300..300 Azimuth correction	U 3k..3.4k Azim. right limit	V -1.6..1.2k Azim left limit	W 0..3599 longitude	X -800..800 latitude	Y -32k..32k mx neutal position	Z -32k..32k My neutral position	!
*!4	[700..900 Elevation up limit	\ 1..12 Elev double steps	] 0..300 Elevation down limit	^ Reserved for ID	_111	`111	a111	b 0..900 Elevation Stow	!
*!5	c-900..2.7k Azimuth clean	d 0..900 Elevation clean	e-900..2.7k Azimuth Home	f 0..900 Elevation Home	g-50..200 Elev.trigger sleep	h 111	i 10..20000 sensor direct sun	j 10..500 sensor step azimuth	!
*!6	k 10..500 sensor step elevation	l 10..1000 sensor run azimuth	m 10..1000 sensor run elevation	n 0..800 sensor hysteresis	o 0..800 sens direct. Hysteresis	p 1..200 astro error step azim.	q 1..200 astro error step elev.	r 1..200 astro error run azim.	!
*!7	s 1..200 astro error run elev	t 1..20 calibration elev. err.	u 0..32767 Elev. over-current	v-1..1023 windspeed stop	w1..100 wind trig-ger time	x1..100 wind re-turn time	y azi rotation speed	z azim left of east	!
*!8	:21 year	:5 month	:1 day	:17 hour	:45 minute	:33 second	!		

Commands List

#	Restore factory defaults for all EEPROM parameters
%	Show compass readings in the serial monitor
$0	return to normal tracking (careful, this also works from calibration routines and may have undesired results)
$1	Disables sensor tracking and sets cleaning position "c" "d" as target. The tracker will move to the cleaning position. You can also use this for manual moves. Type $1 again, for changes of "c" and "d" to take effect.
$8	Initiate a recalibration
$9	Initiate a new calibration
$|	This will jump into calibration step 3, where you can run the tracker manually. This comes handy in some cases but it is also dangerous because all limits are disabled. Return with $0. Be in company if you should attempt this.

11.1.1 *Time and Date *!8*

Typing "~" into the Serial Monitor will display e.g.:

*!8   :20   :7   :2   :5   :39   :59   !

as the last line. This is 5:39:59 am on July 2nd 2020.

Modify this line to the present UTC time and date and enter it into the Serial Monitor to set the new time and date. You can find the UTC world time on the internet. Your local time will yield wrong results (if you are not living in Greenwich). You can also choose other times and longitudes/latitudes to see how the tracker will behave.

11.1.2 *Marker for primary calibration ;*

This value is used to detect a missing primary calibration and it will change to 1234 after calibration. You cannot change this value manually. After the primary calibration, the upper range of the EEPROM will be used for a safety copy, even in the event of a power failure during recalibration.

11.1.3 *Days since calibration > 0..1000*

This value will keep increasing by 1 at UTC midnight. When it reaches the value "?" the tracker will recalibrate at night without the need for user inputs. Usually, you don't need to change this parameter, but you can.

11.1.4 *Days for calibration ? 1..1000*

After how many days should the tracker do an automatic recalibration? You may start with a rather high value and monitor how the astronomical tracking slowly drifts away. When you believe that it is becoming unacceptable, set the present value of ">" as the new value for "?" and the tracker will recalibrate after sunset.

11.1.5 *Counter overflow value for RTC time correction @ 0..32767 seconds*

We want to have the tracker running indefinitely without the need for user interference and the RTC would slowly drift away. Therefore, we detect situations, where the astronomical tracking is ahead of the sun sensor-based tracking and reduce the time by 2s, when an overflow occurs. The overflow value is given in seconds on sun. This also works the other way around. If the astronomical tracking is falling behind, 2s are added at an overflow in negative direction. The default value of 3600 seconds means, that a correction occurs when the astronomical tracking has been in front or behind for one hour.

11.1.6 *Turbo Mode < 0..1*

You can activate the turbo mode to speed up the progression of the time by a factor of about 50. Thus, you can watch how the tracker will move during one day.

(Internally, each time the seconds in the RTC show 00, they will be set to 59 again. Thus, each minute lasts only a little more than a second.)

11.1.7 *Number of break Cycles B 200..20000*

How long should the motor be in the short circuit break condition before it is ready for another movement? In order to get the number of brake cycles, multiply this time by 4000, where 4000 Hz is the Arduino UNO PWM time @ 16 MHz. 8 MHz Arduinos will only have 2000 Hz PWM. Up to 20000 corresponding to 5s are possible.

Type e.g. 500:@ into the Serial Monitor.

11.1.8 *Plateau Duty Cycles C, D, E: 1..16*

The motor control is using duty cycles from 0 (no operation) to 16 (full speed). If you are running 12V motors on 24V, keep the duty cycle below 13. Also check with the supplier if the motor will tolerate 50% duty cycle at 24V and 4000 Hz.

The parameters

C: up

D: down

E: left/right

specify the plateau duty cycles under full speed operation. The duty cycles C (up) and D (down) can be set separately in case the elevation axis is under load during operation.

11.1.9 Step duty cycles F, G, H: 1..16

The motors can run in small steps for tracking. These steps have separate plateau duty cycles that should be as low as possible to reduce the wear of the motors.

F: step up

G: step down

H: left / right

11.1.10 Ramp repeat I, J, K: 1..255

The PWM is ramped from 0 to plateau PWM duty cycle (max. 16). The parameters I, J, K define how often a certain PWM duty cycle should be repeated. The maximum value is 255 corresponding to 64 ms on each step. Consequently, a full ramp from 0 to 16 takes one second.

I: up

J: down

K: left/right

11.1.11 Plateau repeat step L, M, N: 1..5000

Each step has a plateau time given by the number of PWM cycles. At 4 kHz a maximum plateau time of 5000 correspondig to up to 1250 ms is possible.

L: up

M: down

N: left right

11.1.12 Full speed plateau time O: 1..5000

Each motor movement is a step – even continuous movements. The only difference is, that during continuous movements the plateau counter is reset to 0 in every pass of the main loop. Consequently, the movement continues.

O needs to be chosen such that the plateau time is longer than the time required to pass through the main loop. 1400 (350 ms) has proven to be a good value. If you should experience interruptions of the motor movement, you can set it a little higher.

11.1.13 Elevation up and down correction R, S, -100..100

These values are used during the primary calibration to correct for errors of the tilt measurement. They are locked during normal operation

11.1.14 Azimuth correction T: -200..200

Use this to correct the astronomical azimuth tracking by up to +/- 20°.

11.1.15 Azimuth limits U 3000..3400, V -1600..1200

You may chose the limits for the azimuth. The recommended settings of -1600..3400 azimuth range will mean +/- 250° rotation from the East position. East is chosen to be the neutral po-

sition because the sun always comes up in the east and may move both ways round depending on where you are.

11.1.16 *Longitude and Latitude W -1800..1800, X -800..800*

Look up the coordinates for your site on the web. E.g. 51° 31′ N , 0° 7′ W

Now you need to calculate the Longitude and Lattitude in 0.1° steps:

Longitude = 10 * (0 + 7/60)= 1   → type   1:W   in Serial Monitor

Latitude = 10 * (51 + 31/60) = 515   → type   515:X   in Serial Monitor

11.1.17 *Neutral positions for magnetic x and z: Y, Z -10000 10000*

The tracker uses the horizontal components of the magnetic field to calculate the azimuth. An automatic calibration routine is used to find the neutral position of the magnetic y reading my. For the other direction we need to determine the horizontal x component mx_horizontal from mx and mz. The automatic calibration routine also measures the neutral position of mx_horizontal but high errors of mx and mz may result in a higher sensitivity of the compass reading to the elevation position. Therefore, the neutral position of mx and mz is estimated during the calibration routine and stored to the positions Y and Z in increments of 10. Y and Z are not user definable.

11.1.18 *Elevation up and down limits [800..900,] 0..300*

Not all trackers can perform movements in the full elevation range of 0 to 90° (0..900 * 0.1°). Consequently, you can limit the movement range. These values can only be changed at the beginning of a new calibration. They will override each target position which is out of range. Sensor tracking is disabled, when the astronomical position is out of limits.

[upper limit,

]   lower limit of the elevation range

11.1.19 *Half number of elevation steps \, 1..12*

The backlash "\" let's you configure the number of elevation steps at the beginning of a new calibration. I's defined as the half number of steps to make sure that only even numbers of steps and odd numbers of elevation positions are possible. Keep this number as low as possible. Many calibration positions will mean that your azimuth drive is wearing down in each recalibration. In many cases a value of 1, corresponding to three elevation positions (uppermost, lowermost and in between) will be enough to achieve a sufficiently precise astronomical azimuth tracking. Example: To set 4 elevation steps and 5 elevation positions type

5:\

11.1.20 *Reserved: Tracker ID ^*

RS485 field communication requires a unique tracker ID. It should be in one block with the other parameters that can only be changed during the primary calibration.

11.1.21 *Elevation target position for stow b 0.900*

When the wind gets to strong, the tracker should lay itself flat at a close to Zenith position to avoid high wind loads. The wind may come from any direktrion, so an azimuth stow position makes no sense and running in azimuth during storm may damage the motor. You can decide whether the tracker should be completely flat during stow or whether it should be slightly

tilted to let rain run of. The stow position is given as a relative value compared to the target position. E.g. Elevation up limit is 900 so type.

-50:b

to set the elevation stow position to 850 = 85°

11.1.22 *Azimuth position for cleaning c -900..2700*

You can get into cleaning mode by typing $1 from normal tracking and the tracker will go into a predefined position in a full 360° range from -900..2700.

11.1.23 *Elevation position for cleaning d 0..900*

E.g.

0:d

11.1.24 *Azimuth home position e -900..2700*

The tracker goes to a defined home position at night. Type the following, to set the target position to 90° azimuth:

900:e

11.1.25 *Elevation home position f 0..900*

Elevation position during the night. Slightly below 900 (90°) is recommended to let rain run off.

850:f

11.1.26 *Elevation position to trigger sleep g -50..200*

Night is defined by a calculated elevation position which is below g. The higher you set g, the earlier night will start and the later it will end. It is a good idea to set this value slightly below the elevation down limit such that the tracker will wake up a little early and have time to move to the sun. This position is defined in 0.1° steps. Values lower than 0 are possible since this is only a value used in the calculations. E.g. type

100:g   to set the wake up position to 10° elevation

11.1.27 *Sensor level to detect direct sun i 10..20000*

When the sensor is receiving a certain level of light, the tracker is switching into sensor-based sun tracking. The higher the value, the later the tracker will go to sensor-based tracking. I allowed for a max. value of 20k such that you can set the tracker to never enter sun sensor mode.

11.1.28 *Sensor error for azimuth step j 10..500 and elevation step k 10..500*

The software calculates the relative difference of the upper and lower sensor elements. When the value is beyond a certain limit, a tracking step is initiated. The lower the value j, the earlier the step will start.

11.1.29 *Sensor run error for azimuth l 10..1000 and elevation step m 10..1000*

Make sure that these values are higher than the step errors.

11.1.30 *Sensor tracking hysteresis n 0..800%*

We want to prevent that the tracker is moving from astronomical tracking to sensor tracking and back all the time. This is why we enter a hysteresis. The sun sensor reading is increased by n percent when the tracker already is in sun sensor mode.

11.1.31 *Sensor tracking direction hysteresis o 0..2000%*

The same principle applies to direction reversals in tracking step mode. A high value makes steps in the same direction easier while there is no effect of this value on direction reversals.

11.1.32 *Compass errors for azimuth p and elevation q steps 1..200*

A value of 100 means that a tracking step will be initiated when the compass measures an azimuth (elevation) deviation from the calculated target of more than 10°. You may consider setting these values higher than the values for full speed movements. In this case, there wont be any tracking steps in compass based tracking which may reduce the wear of the motors.

11.1.33 *Compass errors for azimuth r and elevation s corrections at full speed 1..200*

Deviations in degree that will trigger full speed corrections.

11.1.34 *Compass errors for elevation steps during calibration t 1..200*

It is important that the elevation is set correctly during calibration. This is why we are setting a lower trigger level for the elevation elevation steps during calibration. A high value will compromise the magnetic compass precision, a low value may mean, that the elevation keeps clicking forever in the calibration routine and the routine will not continue.

11.1.35 *Elevation overcurrent u 0..32767 mA*

This defines the current in mA at which the tracker will come to a complete halt and display "overcurrent". Even so the VNH5019 only allows 16A continuous, I set the upper limit to the highest signed integer since the currents may fluctuate. The default is 16000. Reduce it as required.

11.1.36 *Stow windspeed in clicks per 10s v -1..1024*

The present version of the software assumes a sensor with a reed relay which will close at certain angles while the wheel is rotating. Thus it is generating "clicks". The more clicks there are in 10s, the faster the wind.

-1:v will always enable storm (for testing)

1024:v will disable storm

300:v preset for 20 meters per second and N25FR sensor*/

11.1.37 *Wind trigger time in seconds w 1..1000*

A wind gust above the stow wind speed will be detected but it only triggers "go to stow" if the windspeed remains above the limit for longer than the wind trigger time w which may be set in the range of 1 to 1000 seconds.

11.1.38 *Wind return time in seconds x 1..1000*

In the same way, the tracker only returns from stow if the wind speed is below the limit for longer than the wind return time x.

11.1.39 *Milliseconds per degree azimuth y*

Actually, this value is not user defined, but you can see it in the parameter block for reference. Magnetic readings are so noisy that they need to be averaged, even when the tracker is moving. To do that properly, we need the rotation speed.

11.1.40 *Azimuth left of east z*

Again, this is not a user defined parameter. The control sets it automatically when it is moving over the east position to remember whether it has been right or left of east. A hysteresis suppresses rapid changes, that might wear down the EEPROM.

11.2 MIT License

11.3 Previous Work and Acknowledgements

The basic concept of compass tracking has been presented at the CPV-17 conference on behalf of my employer AZUR SPACE Solar Power GmbH [0]. Yet, I am publishing the present work on a strictly private basis. All necessary rights for this open source publication have been transferred to me. AZUR SPACE doesn't hold any rights that might imply any form of liability.

I greatly thank my employer for his support. May this work facilitate the success of CPV.

11.4 References

0. R. Loeckenhoff, CPV-17, Freiburg, "A minimalistic, universal and highly precise sun sensor and compass based CPV-tracker control".

1. R. Loeckenhoff, "Sonnenstandssensor", DE102018001181B3

2. R. Loeckenhoff, "Sonnennachführungsvorrichtung", DE102019004468A1

3. Saheli Ray, "Calculation of Sun Position and Tracking the Path of Sun for a Particular Geographical Location" in International Journal of Emerging Technology and Advanced Engineering, Vol. 2, Issue 9, Sept 2012.

4. Muhammad E. H. Chowdhury, Amith Khandakar, Belayat Hossain, Rayaan Abouhasera, "A Low-Cost Closed-Loop Solar Tracking System Based on the Sun Position Algorithm", Journal of Sensors, vol. 2019, Article ID 3681031, 2019; https://doi.org/10.1155/2019/3681031

5. www.arduino.cc

6. "Pololu Dual VNH5019 Motor Driver Shield for Arduino (ash02a)", https://www.pololu.com/product/2502.

7. maxim integrated, "DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal", https://datasheets.maximintegrated.com/en/ds/DS3231.pdf, 2015.

8. invensense, "MPU-9250 Product Specification Revision 1.1"

11.5 Revision Index

01.06.2021	R. Loeckenhoff	Initial open source publication
06.06.2021	R. Loeckenhoff	Detailed instructions for housing and one correction in the wiring (Wind sensor on D3)
13.06.21	R. Loeckenhoff	Improvements to sensor tracking with floating average and direction hysteresis o extended to 2000%.