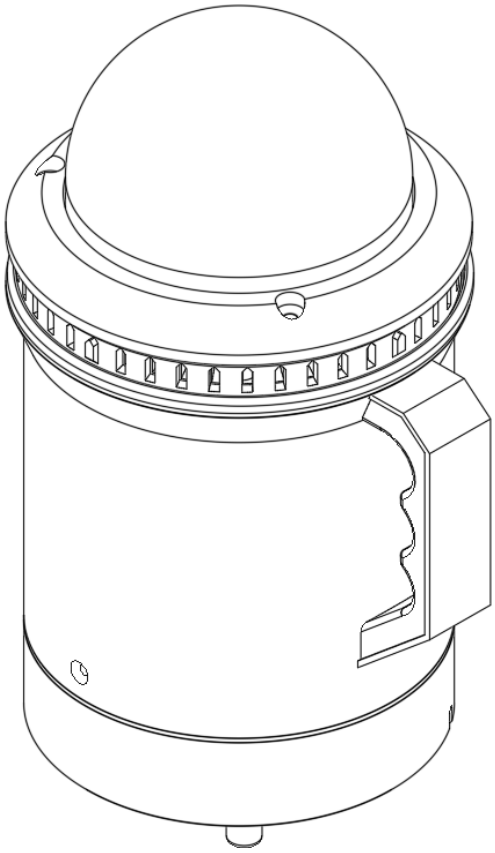
Allsky GO

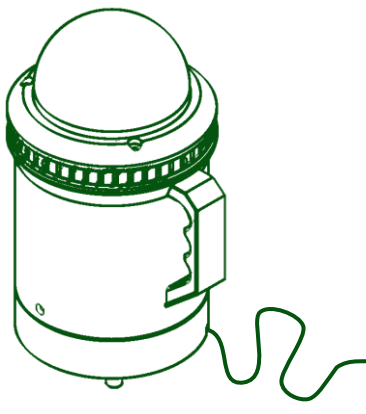


Installation Manual

v1.0

July 2025

Contents



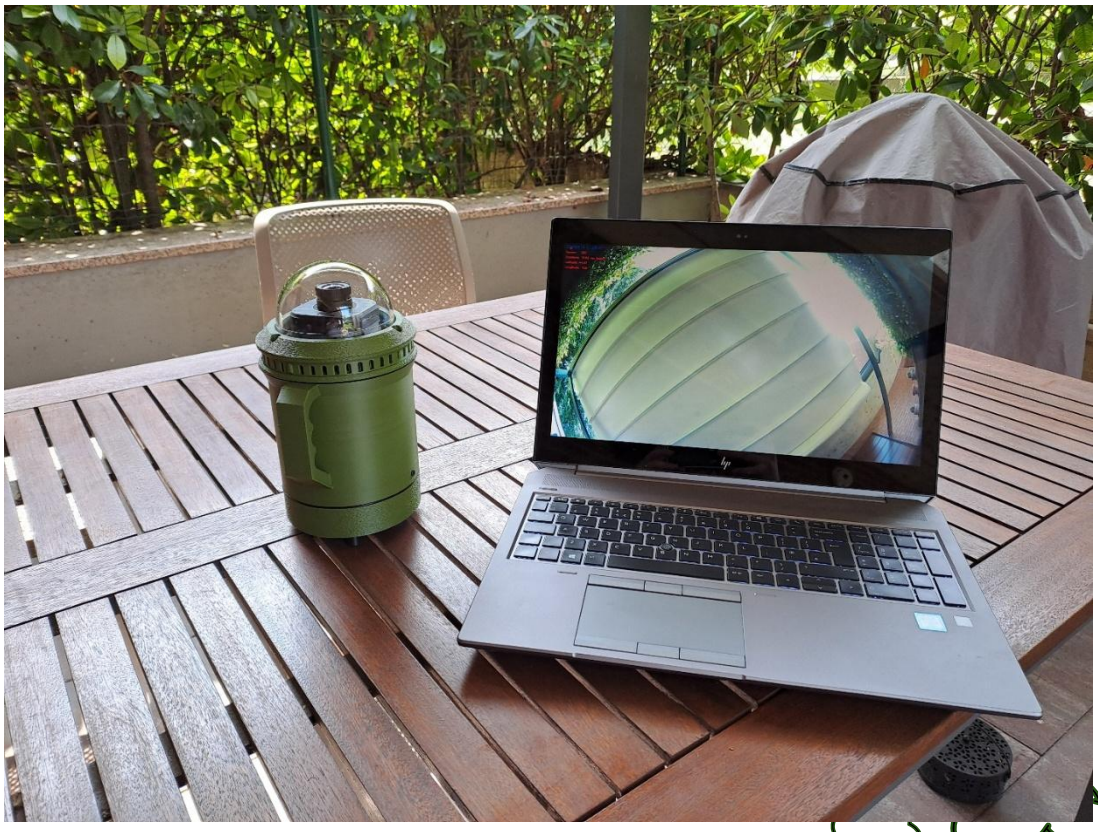
INITIAL RASPBERRY SET-UP

Install Allsky

Install the Raspberry PI Software onto an SD Card according to your Raspberry PI.

The desktop version is recommended to facilitate system file access through remote desktop.

Secondly, follow steps to install the latest distribution of the Allsky Team Software Package



Raspberry Pi Boot Configuration

🎯 Optimizing Raspberry PI Boot Configuration for Minimal power consumption

Modify Boot configuration with the following command:

```
Sudo nano /boot/firmware/config.txt
```

Add the following lines

`dtoverlay=pwm-2chan`
#**Enables dual-channel PWM** (Pulse Width Modulation), used to drive the buzzer.

`hdmi_blanking=2`
#**Disables HDMI output**, preventing unnecessary power usage if no display is connected. Perfect for off-grid or headless operation.

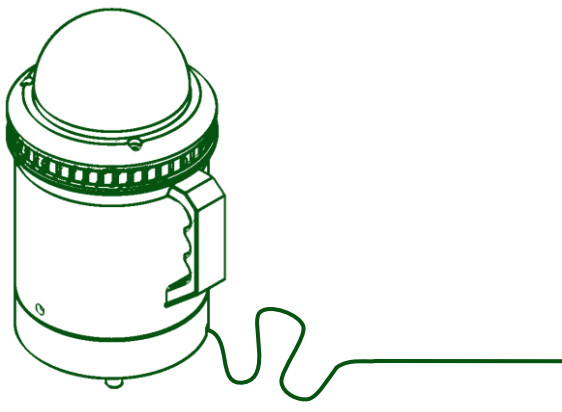
`dtoverlay=disable-bt`
#**Disables onboard Bluetooth**, freeing system resources and reducing power draw—smart move if you do not need wireless peripherals.

`dtparam=audio=off`
#**Turns off the onboard sound system**, specifically the `snd_bcm2835` driver. This prevents the audio hardware from loading—again, saving resources.

Disable ACT LED (green)
`dtparam=act_led_trigger=none`
`dtparam=act_led_activelow=off`

Disable PWR LED (red)
`dtparam=pwr_led_trigger=default-on`
`dtparam=pwr_led_activelow=off`

This helps minimize light pollution—crucial for clean night sky imaging—and reduces battery drain. **To Exit -** Press **`Ctrl + X`** & **`Yes`** to Save

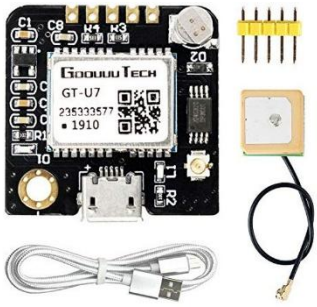


Sample Lines from boot config.txt after modification:

```
#ALLSKY:Enable pwm for Buzzer
dtoverlay=pwm-2chan

#ALLSKY:Disable HDMI Output
hdmi_blanking=2

#ALLSKY Disable bluetooth
dtoverlay=disable-bt

#ALLSKY Disable audio (loads snd_bcm2835) - ALLSKY:Changed to off
dtparam=audio=off

# Disable ACT LED (green)
dtparam=act_led_trigger=none
dtparam=act_led_activelow=off

# Disable PWR LED (red)
dtparam=pwr_led_trigger=default-on
dtparam=pwr_led_activelow=off
```

SENSOR INSTALLATION

Installing the GPS Module (GT-U7)



#USB Connection & Installation of GT-U7 GPS Module

#GPS Library

```
Sudo apt-get install gpsd gpsd-clients
```

#USB Utilities

```
Sudo apt-get install usbutils
```

****

#Identify USB Serial Port - check connection to USB listed as U-Blox:

```
lsusb
```

#List connected Devices

#Look for /dev/ttyUSB0 or /dev/ttyACM0 - this is your GPS Port:

```
dmesg | grep tty
```

#Check raw data feed - If you see NMEA sentences (lines starting #with $GPGGA, $GPRMC), your GPS is working:

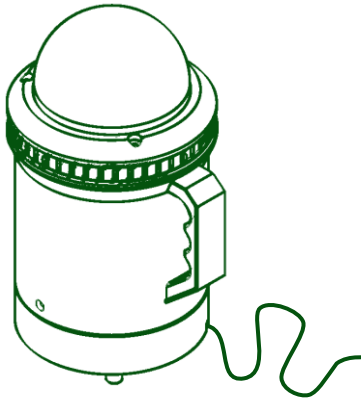
```
cat /dev/ttyACM0
```

#Verify if Running

```
sudo systemctl status gpsd
```

#If not active, restart with

```
sudo systemctl restart gpsd
```



Installing the Humidity Sensor (SHT31)

Enable I²C on the Pi

Check using the command line (non-interactive)

```
sudo raspi-config nonint get_i2c
```

- Output 0 means **enabled,** Output 1 means **disabled**

Use raspi-config to enable I²C:

```
sudo raspi-config
```

Navigate to *Interfacing Options → I2C → Enable*.

The SHT31 sensor typically communicates via the **I²C interface**, which **expects to use specific GPIO pins**:

- **GPIO2 (Pin 3)** for SDA
- **GPIO3 (Pin 5)** for SCL

✅ 3. Check for I2C device node, Run:

```
ls /dev/i2c-1
```

If you see /dev/i2c-1, I2C is enabled. If you get "No such file or directory," it is not

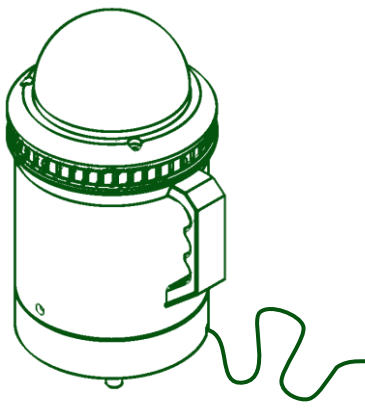
✅ 4. Scan for connected I2C devices, install tools if needed:

```
sudo apt install -y i2c-tools
```

Then scan:

```
sudo i2cdetect -y 1
```

You should see a grid with addresses like 0x44 (or 0x45) if your SHT31 sensor is connected and working.



SCRIPTS

Step1 : Loading Custom Scripts for Allsky GO from Terminal

#Create Folder for Custom Scripts on Raspberry Pi

```
mkdir /home/pi/scripts
```

#Move to Folder

```
cd scripts
```

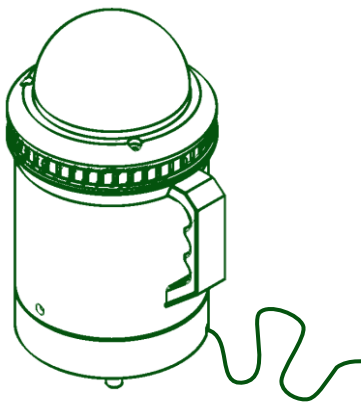
List of Scripts to copy from Github;

- button a+b.py — Monitors and Controls Button Press Actions
- update_gps.py — Monitors and Updates Overlay Data (GPS + Temp)
- Buzzer Melodies(7) — Subprocesses to drive Buzzer at Button Press
- wps_connection.py — Subprocess for WPS connection after Button Press

Note: connected hardware can be tested with some python script commands in the Test Folder

1. #Create Nano File for each script in the scripts folder: i.e., `nano button_script.py`
2. **Paste** the text from github files into nano text editor
3. **Exit** - Press **Ctrl + X** & **Yes** to Save – preserving the python file names.

Note: Python Scripts and Processes based around Raspberry Pi5. Read the code to check any references that may need to be updated for your specific set-up.



Step2: GPS Service at Start-up [update_gps.service]

To automatically run your script /home/pi/scripts/update_gps.py at startup on your Raspberry Pi, the most reliable and modern method is to use a **systemd service**. Here is how to set it up:

✅ Step-by-Step: Create a systemd service

1. **Create a service file**:

```
sudo nano /etc/systemd/system/update_gps.service
```

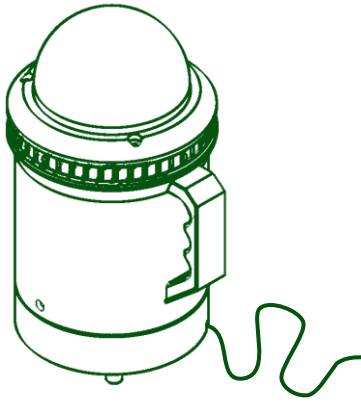
2. **Paste this configuration**:

```
[Unit]
Description=Update GPS Text File
After=network.target

[Service]
ExecStart=/usr/bin/python3 /home/pi/scripts/update_gps.py
WorkingDirectory=/home/pi/scripts
StandardOutput=journal
StandardError=journal
Restart=on-failure
User=pi     #CHANGE TO YOUR USER IF REQUIRED

[Install]
WantedBy=multi-user.target
```

Save and exit (Ctrl+O, Enter, then Ctrl+X)

3. **Enable the service to run at boot**:

```
sudo systemctl daemon-reload
```



```
sudo systemctl enable update_gps.service
```

4. **Start it now (optional)**:

```
sudo systemctl start update_gps.service
```

Step3. Button Press Service [button_script.service]

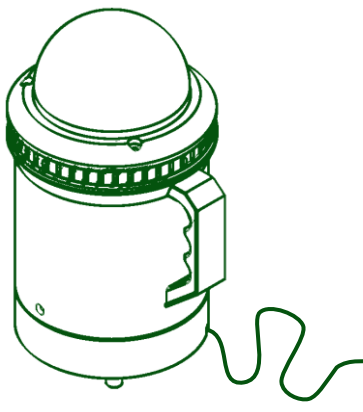
✅ Step-by-Step: Create a systemd service

```
sudo nano /etc/systemd/system/button_script.service
```

Paste into: /etc/systemd/system/button_script.service the following text;

```
Description=Button Script for Allsky
After=network.target

[Service]
ExecStart=/usr/bin/python3 /home/pi/scripts/button_a+b.py
WorkingDirectory=/home/pi/scripts
StandardOutput=inherit
StandardError=inherit
Restart=always
User=pi                 #CHANGE TO YOUR USER IF REQUIRED

[Install]
WantedBy=multi-user.target
```

RESTART Service and Daemon



```
sudo systemctl restart button_script.service
sudo systemctl enable button_script.service
sudo systemctl daemon-reload
```

Debugging System Services

Check its status:

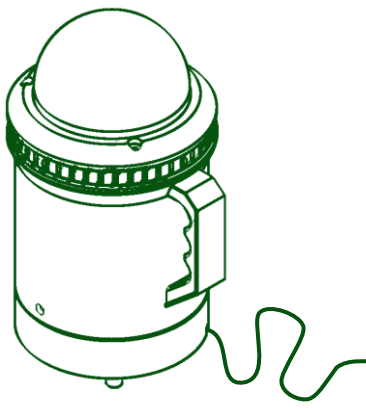
```
sudo systemctl status update_gps.service
sudo systemctl status button_script.service
```

Check the Debug Log

```
journalctl -u update_gps.service
journalctl -u button_script.service
```

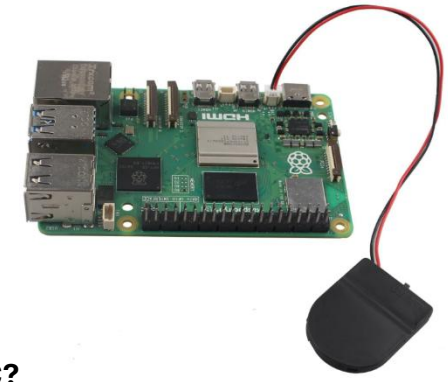
Check all Running Processes & Resource Level

```
htop
```



RTC & TIME SYNCHRONISATION

In an Allsky camera setup — where capturing accurate night sky images depends on knowing exactly when day turns to night — keeping your Raspberry Pi's system clock in sync is essential. Let's break down how that works and the three main ways your Pi can maintain reliable time.



🧠 What Is RTC?

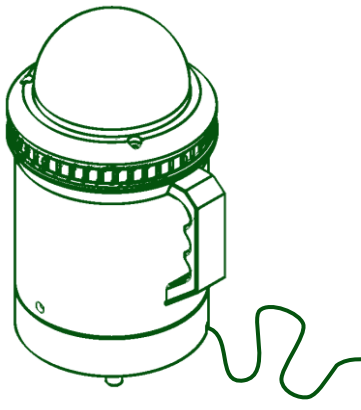
RTC stands for **Real-Time Clock** — a tiny onboard chip that keeps track of time even when the system is powered off. On Raspberry Pi 5, the RTC is built into the power management IC (PMIC), but it needs a **battery backup** to retain time during shutdowns or power loss.

Without an RTC or external time source, the Pi boots up thinking it is 1970 — not ideal for astrophotography!

🌓 Why Time Sync Matters for Allsky

Allsky software uses timestamps to:

- Trigger **nighttime image capture**
- Generate **timelapse videos**, **keograms**, and **startrails**
- Overlay **date/time/location** on images
- Schedule **dark frame subtraction** and other modules



If your Pi does not know the correct time, it might start imaging during daylight or miss celestial events entirely.

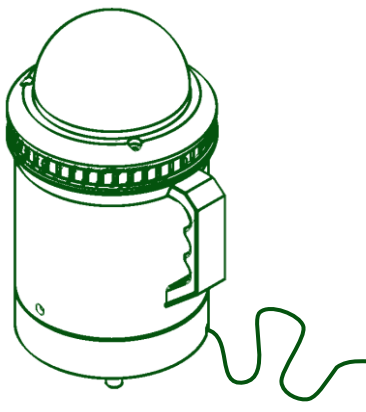
🛠️ Three Ways to Keep Time Accurate - Pros & Cons

Option	Pros ✅	Cons ❌
❶ **Built-in RTC + External Battery** 🔋	- Keeps time during shutdown or power loss - No dependency on network or GPS - Simple and low power	- Accuracy can drift over time - Requires correct battery and config - Doesn't auto-correct or sync
❷ **Network NTP Servers** 🌐	- Highly accurate with low offset - Easy to set up with Pi OS - Can auto-sync continuously	- Needs stable internet connection - Not ideal for remote observatories - Vulnerable to connectivity drops
❸ **GPS Module (GT-U7)** 🛰️	- Atomic-level precision - Works offline, great for remote locations - Provides both date and time from satellites	- Requires clear sky view - Longer Time-To-First-Fix without A-GPS - Initial setup is more complex

The GPS module is an excellent solution for Off-Grid time keeping but is heavily dependent on a **valid GPS Fix** ; without which no timing synchronization can be obtained. The Built in RTC with external battery is recommended / necessary as primary back-up.

For a robust Allsky setup:

- Use **GPS time** as your primary source.
- Add **RTC battery backup** to preserve time across reboots.
- Optionally fall back to **NTP servers** if internet is available.



The external RTC battery pack on the Raspberry Pi 5 and NTPsec serve complementary roles in keeping your system's time accurate.

📡 **Setting Up Time Synchronization through GPS Module**

Here is a quick breakdown of what you are configuring:

⏱️ check Installation of NTPsec (an updated version of NTP)

- Installs the Network Time Protocol daemon, which helps keep your system clock synchronized with external time sources.

```
#Time Synchronization from GPS
Sudo apt install ntpsec
```

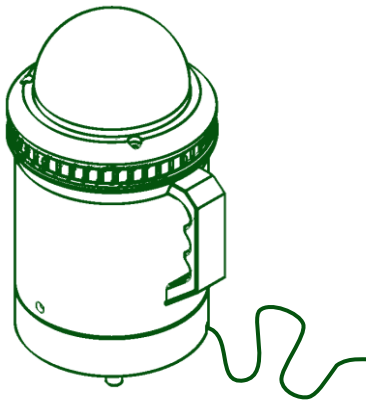
⏱️ Add NTP Sources to the configuration (GPS and PPS signals from the GT-U7)

```
sudo nano /etc/ntpsec/ntp.conf
```

Add the following lines in /etc/ntp.conf to identify your GPS;

```
#GPS Standard signal through Driver 28 (USB Connection)
server 127.127.28.0
fudge 127.127.28.0 time1 0.350 refid GPS
#PPS signal if available offers extremely precise timing and complements the NMEA data source - Driver 46 - Requires GPIO Connection
server 127.127.46.0
fudge 127.127.46.0 refid PPS
```

📡 Configuring gpsd for persistency in a modern systemd setup:



1. **Create a systemd override for gpsd**:

```
sudo systemctl edit gpsd
```

2. Add this to the override file:

```
[Service]
ExecStart=
ExecStart=/usr/sbin/gpsd -n /dev/ttyACM0
```

The first ExecStart= clears the previous command, and the second defines your custom startup. Ensures deterministic behavior at boot. It removes ambiguity and guarantees gpsd binds immediately to your module. /dev/ttyACM0 is the USB location*

📡 Save and reload:

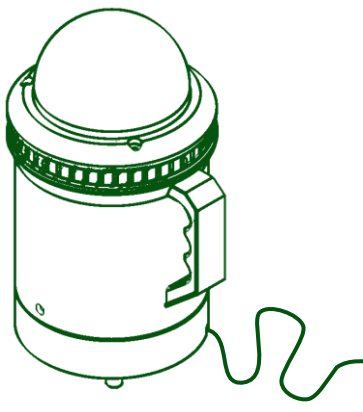
```
sudo systemctl daemon-reexec
sudo systemctl restart gpsd
sudo systemctl restart ntpsec
```

***Note:** USB Port Address

Above example assumes USB location of the connected GPS module is: "/dev/ttyACM0". Use the command to identify the correct USB address;

```
ls /dev/tty*
```

This will show entries like /dev/ttyUSB0, /dev/ttyACM0. These are typically assigned to USB-to-serial adapters.



Example gspd override;

```
  GNU nano 7.2                /etc/systemd/system/gpsd.service.d/.#override.conf56169b232d1ee0d9
### Editing /etc/systemd/system/gpsd.service.d/override.conf
### Anything between here and the comment below will become the new contents of the file

[Service]
ExecStart=
ExecStart=/usr/sbin/gpsd -n /dev/ttyACM0


### Lines below this comment will be discarded

### /lib/systemd/system/gpsd.service
# [Unit]
# Description=GPS (Global Positioning System) Daemon
# Requires=gpsd.socket
# # Needed with chrony SOCK refclock
# After=chronyd.service
#
# [Service]
# Type=forking
# EnvironmentFile=-/etc/default/gpsd
# ExecStart=/usr/sbin/gpsd $GPSD_OPTIONS $OPTIONS $DEVICES
#
# [Install]
# WantedBy=multi-user.target
# Also=gpsd.socket
```

GPS & RTC Debugging

#Check if GPS is providing time data:

```
cgps - s
```

🕰️ Check time sources of ntpd

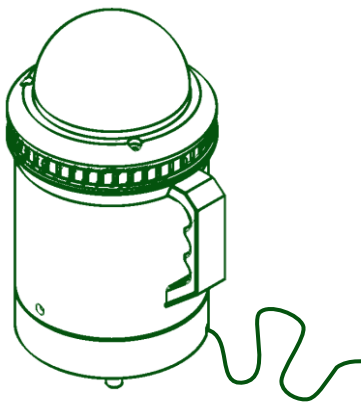
```
ntpq -p
```

shows a mix of internet-based servers and / or GPS time source:

🌟 Primary Time Source Example

```
*99.214.199.85.in-addr.arpa .GPS. 1 u 44 64 1 ...
```

- That asterisk (*) means **this GPS source is currently selected** by ntpd as the **primary time source**.
- The .GPS. refid confirms it is coming from gpsd via shared memory (SHM).



- Stratum 1: Indicates it is a **direct time source**, like a GPS receiver!

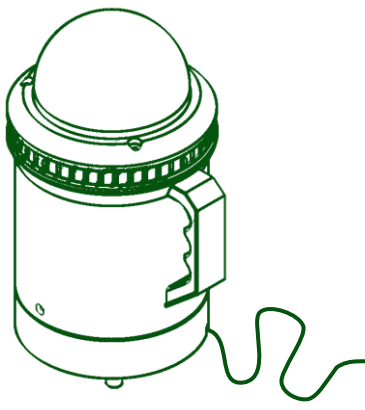
If you see the above status, your system is **successfully using GPS** as its **authoritative time source**, even though it remains connected to Wi-Fi. The internet servers are available, but being used as backups or not at all

🌐 Normal Internet Based Sources

- All the other entries are typical NTP servers from the internet.
- In the NTP.conf file - the Internet Sources are listed as below - they can be disabled for debugging with #.

```
pool 0.debian.pool.ntp.org iburst
```

.



ALLSKY SETTINGS

🌞🌒 Image Capture Settings

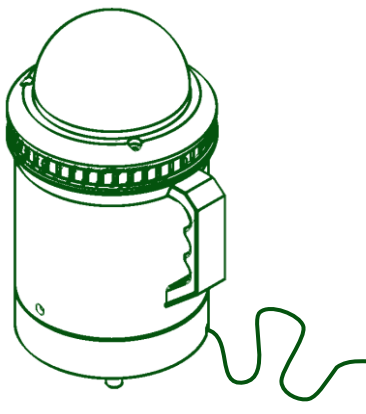
Setting	Status / Value	Function
Save Day/Night Images	✅ Enabled	Ensure image capture continues even if time zone data is missing.
Auto Exposure (Day/Night)	✅ Enabled	Adjusts light levels automatically for consistent image quality.
Max Exposure Duration	30 seconds	Prevents errors during day/night transition misidentification.

🌌 Startrail Configuration

Setting	Value / Status	Function
Startrail Threshold	0.3	Raised from default 0.1 to tolerate foreground elements.
Generate Visuals	❌ Disabled	Startrail, Timelapse, Keogram generation off to conserve battery.

🌍 Web Upload

Setting	Status	Function
Upload Images	❌ Disabled	Disables sync attempts—ideal for off-grid use and testing.



DEBUGGING AND TROUBLE SHOOTING

Raspberry Shuts down when connecting USB cable to charge Power bank When connecting external cable to the USB-C Port, the Power bank interrupts the supply to the raspberry Pi; enough to shut down the PI. The USB port is not hot swappable. Ensure Raspberry PI is shut down before connecting Power bank Charger. The Charger can be charged as it powers the raspberry pi – but is not recommended, may shorten battery life.

Functions within Scripts do not work Scripts developed with Raspberry Pi5. Hardware or OS changes in the Pi 5 may affect legacy script compatibility (e.g., GPIO mapping, overlay handling) if using previous version.

Power Button Light does not Turn On when pressed. Power bank is flat (0%)

***No Fix* showing in Overlay.** Please ensure the camera has direct access to the sky on all four sides. Large buildings or trees may hinder the visibility of Satellites to obtain reliable fix. The GPS Service will continue looking for a valid fix.

***False Fix* showing in Overlay**. Unfortunately, the GT-U7 module can lock onto a false fix; it thinks it is connected but resulting Coordinates at 0:0 or negative altitude. The GPS Service will continue looking for valid fix.

From Terminal

- **Service Debugging**; *journalctl -u [SERVICE NAME]*
- **Check Running Processes**: htop