Remote-Control for Telescope

Arjan te Marvelde, initial version Jan 2016, this version Mar 2024

Optimum observation time on higher latitudes is usually during the winter and early spring. These months can also be quite cold, and hence observation and astrophotography are less than comfortable. So, it is time for a remote-control facility, enabling the operation of mount and astrophotography from inside the warm house.

This document describes a self-contained control station that can be put in a box or even inside the RA unit of an NEQ-6 mount. It can be left alone to capture images during the night, but it can also be controlled remotely through wired or wireless network, from a more convenient location.

Previous versions have been based on dual Raspberry Pi2B, a single Raspberry Pi3B, but now the Pi4 has emerged and the box is upgraded once more. The Pi5 has not yet been tried, but would give about twice the performance. The document describes step-by-step how I made my software configuration, based on Ubuntu server, MATE user interface, the INDI framework, EKOS/KStars and PHD2. The description is chopped up in parts which may also be used independently.

This following topics are covered:

- **Design choices** — Detailing on the system design backgrounds
- **Hardware Configuration** — Overview of hardware related issues
- **Software Configuration** — The complete SW installation process
- **References** — Some handy references

Summary

INDI box

- **RPi4-4GB** with a fast 64GB SD, running the latest Ubuntu, providing a **WiFi hotspot** access point as well as an **Ethernet interface** to connect to an upstream network.
- The RPi4 hosts **EKOS/KStars**, **INDI-server**, **PHD2**, **Astrometry** plate solver and all **INDI drivers**, and it runs a light weight **MATE desktop** with a **VNC server** (x11vnc) for remote access.
- The RPi4 is enclosed in a small unit to be mounted inside the NEQ6 RA housing, that provides **12V power** and **USB** or **serial** data interfaces to all peripherals.
- The box also contains a U-Blox NEO8 **GPS receiver** for time synchronization and local coordinates.

Devices

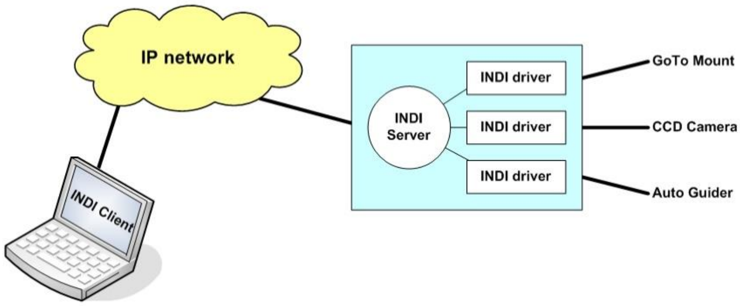
- A **Canon 450D DSLR** is connected through USB and a dedicated power adapter,
- The **NEQ6 mount** is connected internally through a serial interface,
- The **GPS** is connected internally through a serial interface, for time and position,
- A home made **joystick** is connected through USB,
- A **ZWO ASI385MC autoguider / planetary** camera also connected through USB.

User interface

- A **Laptop**, which has a wireless (or wired) connection to the INDI box.
- The Windows laptop hosts a **VNC client** (RealVNC) enabling GUI access to the RPi desktop. This laptop could be replaced with any other device hosting a VNC client.
- Alternatively, a small **LUbuntu** netbook can be connected through VNC
- Even an Android smartphone running RealVNC can be used as remote desktop

Design choices

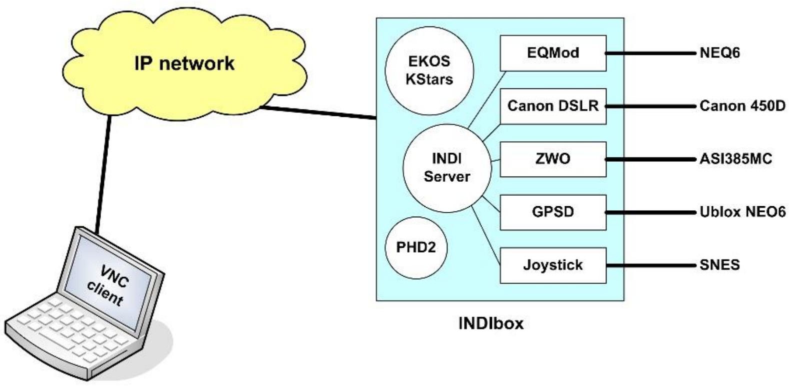
One option for implementing remote-control is to route a lot of cables from the observation location into the house. A much more elegant way is to run all device-interfacing on a local embedded computer and to have a remote workstation for controlling the observatory setup. This is exactly what the INDI framework offers: a server that provides a standardized method to access and control the variety of attached devices, such as goto, camera and auto guider. The INDI server connects to an INDI compatible client on a workstation, through any IP network, such as the home LAN.



Regular Architecture

When the workstation PC is Windows based, the choice of clients boils down to ***Cartes du Ciel*** for goto control and the photo capturing software ***CCD-Ciel***. The main alternative ***KStars / EKOS*** is running on Linux.

A more robust solution is to use a remote *virtual desktop* and run the INDI client on the remote computer as well. The network then may fail while the remote telescope control continues to function. However, allocating both INDI server and clients to the remote computer implies that significantly more processing power is needed. One solution is to base the control unit on dual RPi2, to distribute the load. The RPi3 and certainly the RPi4 appear to be powerful enough to run everything in one processor. The user interface is a virtual desktop, running on the Windows workstation or even on a tablet or a smart-phone.



VNC based architecture

Hardware configuration

Inside the INDI box, the RPi4 can easily be located. The GPS receiver and antenna are mounted on top of that, and to the left the 3A SMPS for the RPi4 power provisioning.

The INDI box

The first version of hardware configuration has been built into a Teco plastic enclosure, which has plenty space.

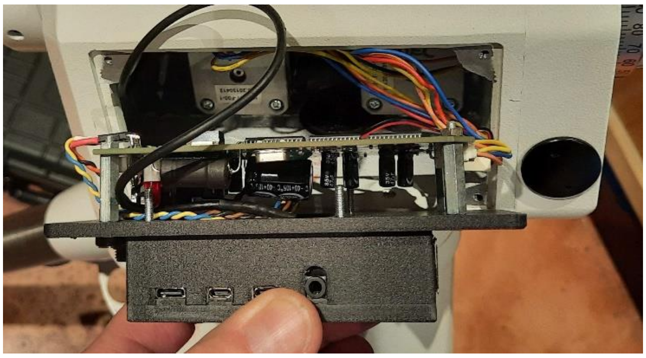
Inside the INDI box

In the picture you can see the RPi4 to the right, underneath a GPS unit (U-Blox NEO6) and a utility board. A 3Amp 5V buck converter is connected to the power pins on the RPi GPIO connector. The lot runs on approximately 12V (from a PSU or battery) which is also output directly on the front to supply the NEQ6, the DSLR and a fan. These outlets should really be fused separately...

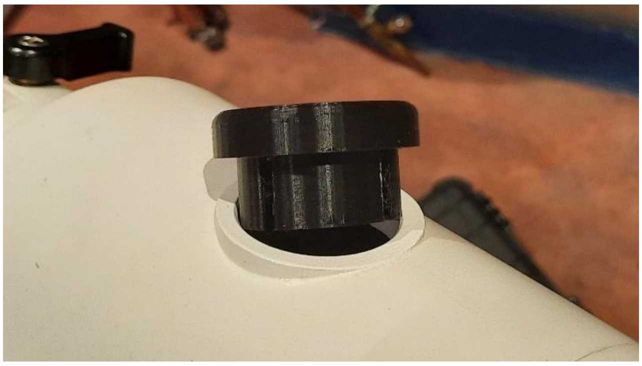
A 25mm/12V fan has been added as the RPi-4 generates more heat than its predecessors.

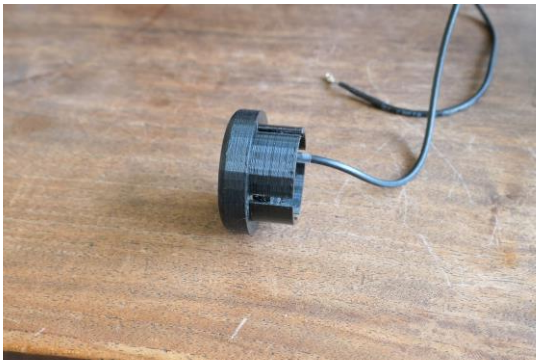
Indi inside NEQ6

The current implementation integrates the lot inside the NEQ6 mount, replacing the service panel.



Here the NEQ mount is connected directly to a serial interface on the RPi4 GPIO connector, just like the GPS module. The NEQ6 driver board is mounted on slightly larger studs and protrudes a little further into the housing than before, so the original connectors etc could be left intact. The NEQ6 housing appears to have plenty space to allow for this.





A PCB was made that plugs onto the Pi4 GPIO connector. This extension board hosts a small 5V DC-DC buck converter to supply the Pi4 from the external 12V. It also hosts the new Ublox Neo8 GPS receiver and the level shifters for the NEQ6 serial interface. The GPS receiver is powered directly from the Pi4 3V3 voltage.

The GPS antenna module has been built into a plug that replaces the cap for the polar alignment scope. You can still take this antenna plug out allowing polar alignment of the mount.

All plastic parts are 3D printed in PETG and PLA.

Pi4 serial interface usage:

To connect the GPS receiver and the NEQ6 control interface, two serial interfaces on the RPi IO connector are used. The SW adaptations are described further down, but the pin usage is described here.

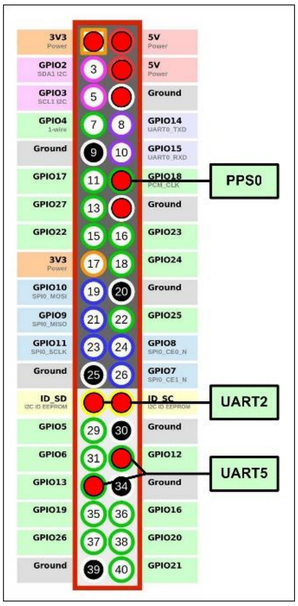
The GPS receiver is a UBlox (NEO-8M), providing 3V3 UART and PPS interfaces. These can be directly connected to a UART of the RPi.

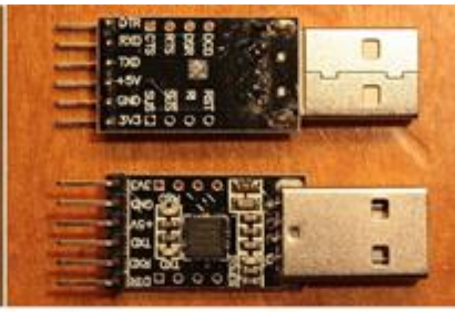
The GPIO 14 and 15 are monitored by U-Boot (as of Ubuntu 20.04) to check for interrupting key presses, and should preferably not be used for peripherals. Either the U-Boot environment must be changed for this, which requires recompiling or actually halting boot and setting the variables, or the port is simply left alone. An alternative is to use UART2 on pins 27 and 28.

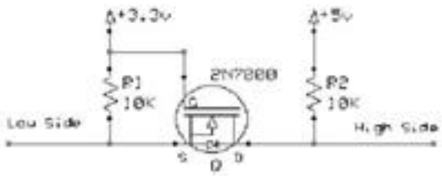
The PPS is connected to a General Purpose IO on pin12 (GPIO18). This can also be left out if subsecond accuracy is not needed.

The UART5 is used to directly connect the NEQ6 control interface inside the NEQ6 motor housing (top middle).

- 3V3 out: pin 1 → to GPS and level shifters
- Vcc in: pin 2, 4 → from DC-DC converter
- GND: pin 6, 14, 30, 34 → common ground
- GP18: pin 12 (GPIO 18) → to GPS PPS
- Tx2: pin 27 (GPIO 0) → to GPS RxD
- Rx2: pin 28 (GPIO 1) → to GPS TxD
- Tx5: pin 32 (GPIO 12) → to NEQ6 RxD (via level shifter)
- Rx5: pin 33 (GPIO 13) → to NEQ6 TxD (via level shifter)







In case an external INDI Box is used, this serial interface needs to be connected to a RPi USB outlet by means of a converter, such as CP2102 or FTDI (top right). Take care of voltage levels when interfacing the GPIO pins, the NEQ6 interface is TTL (5V) and the RPi is LVTTL (3V3). When connecting different digital signal levels, a level-converter is needed for each lead, as for example in above image (bottom right).

Rpi4 pin and GPIO overview:

The Raspberry Pi has several alternative ways to map functions onto the 40 pin GPIO header:

Pin	GPIO / Function	Default Resistor State	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
1	**3v3**							
2	**5v**							
3	2	UP	**SDA1**	SA3	LCD_VSYNC	SPI3_MOSI	CTS2	SDA3
4	**5v**							
5	3	UP	**SCL1**	SA2	LCD_HSYNC	SPI3_SCLK	RTS2	SCL3
6	**GROUND**							
7	4	UP	**GPCLK0**	SA1	DPI_D0	SPI4_CE0_N	TXD3	SDA3
8	14	DOWN	**TXD0**	SD6	DPI_D10	SPI5_MOSI	CTS5	TXD1
9	**GROUND**							
10	15	DOWN	**RXD0**	SD7	DPI_D11	SPI5_SCLK	RTS5	RXD1
11	17	DOWN	FL1	SD9	DPI_D13	RTS0	SPI1_CE1_N	RTS1
12	18	DOWN	PCM_CLK	SD10	DPI_D14	SPI6_CE0_N	SPI1_CE0_N	**PWM0**
13	27	DOWN	SD0_DAT3	TE1	DPI_D23	SD1_DAT3	ARM_TMS	SPI6_CE1_N
14	**GROUND**							
15	22	DOWN	SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST	SDA6
16	23	DOWN	SD0_CMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK	SCL6
17	**3v3**							
18	24	DOWN	SD0_DAT0	SD16	DPI_D20	SD1_DAT0	ARM_TDO	SPI3_CE1_N
19	10	DOWN	**SPI0_MOSI**	SD2	DPI_D6	I2CSL_SDA_MOSI	CTS4	SDA5
20	**GROUND**							
21	9	DOWN	**SPI0_MISO**	SD1	DPI_D5	I2CSL_SDI_MISO	RXD4	SCL4
22	25	DOWN	SD0_DAT1	SD17	DPI_D21	SD1_DAT1	ARM_TCK	SPI4_CE1_N
23	11	DOWN	**SPI0_SCLK**	SD3	DPI_D7	I2CSL_SCL_SCLK	RTS4	SCL5
24	8	UP	**SPI0_CE0_N**	SD0	DPI_D4	I2CSL_CE_N	TXD4	SDA4
25	**GROUND**							
26	7	UP	**SPI0_CE1_N**	SWE_N_SRW_N	DPI_D3	SPI4_SCLK	RTS3	SCL4
27	0	UP	**SDA0**	SA5	PCLK	SPI3_CE0_N	TXD2	SDA6
28	1	UP	**SCL0**	SA4	DE	SPI3_MISO	RXD2	SCL6
29	5	UP	GPCLK1	SA0	DPI_D1	SPI4_MISO	RXD3	SCL3
30	**GROUND**							
31	6	UP	GPCLK2	SOE_N_SE	DPI_D2	SPI4_MOSI	CTS3	SDA4
32	12	DOWN	**PWM0**	SD4	DPI_D8	SPI5_CE0_N	TXD5	SDA5
33	13	DOWN	**PWM1**	SD5	DPI_D9	SPI5_MISO	RXD5	SCL5
34	**GROUND**							
35	19	DOWN	PCM_FS	SD11	DPI_D15	SPI6_MISO	SPI1_MISO	PWM1
36	16	DOWN	FL0	SD8	DPI_D12	CTS0	SPI1_CE2_N	CTS1
37	26	DOWN	SD0_DAT2	TE0	DPI_D22	SD1_DAT2	ARM_TDI	SPI5_CE1_N
38	20	DOWN	PCM_DIN	SD12	DPI_D16	SPI6_MOSI	SPI1_MOSI	GPCLK0
39	**GROUND**							
40	21	DOWN	PCM_DOUT	SD13	DPI_D17	SPI6_SCLK	SPI1_SCLK	GPCLK1

To connect the GPS receiver for example, the alternate mapping of TX2/RX2 is used (to pins 27/28). To enable another UART interface, for example TX5/RX5 can be mapped to pins 32/33.

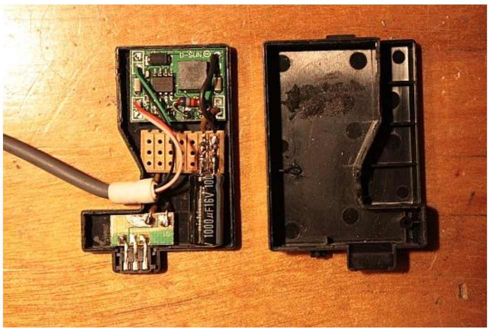
The way Raspberry Pi accomplishes correct pin mappings, is by activating predefined Device Tree overlays from **`usercfg.txt`**. For the implemented configuration on the RPi4 with Ubuntu 22.04 LTS:

```
Name:      disable-bt
Info:      Disable onboard Bluetooth on Pi 3B, 3B+, 3A+, 4B and Zero W,
           restoring UART0/ttyAMA0 over GPIOs 14 & 15.
           N.B. To disable the systemd service that initialises the modem so it
           doesn't use the UART, use 'sudo systemctl disable hciuart'.
Load:      dtoverlay=disable-bt
Params:    <None>



Name:      pps-gpio
Info:      Configures the pps-gpio (pulse-per-second time signal via GPIO).
Load:      dtoverlay=pps-gpio,<param>=<val>
Params:    gpiopin                 Input GPIO (default "18")
           assert_falling_edge     When present, assert is indicated by a falling
                                   edge, rather than by a rising edge
                                   (default off)
           capture_clear           Generate clear events on the trailing edge
                                   (default off)



Name:      uart2
Info:      Enable uart 2 on GPIOs 0-3. BCM2711 only.
Load:      dtoverlay=uart2,<param>
Params:    ctsrts                  Enable CTS/RTS on GPIOs 2-3 (default off)



Name:      uart5
Info:      Enable uart 5 on GPIOs 12-15. BCM2711 only.
Load:      dtoverlay=uart5,<param>
Params:    ctsrts                  Enable CTS/RTS on GPIOs 14-15 (default off)
```

See further below in the section addressing the ***Extensions***.

DSLR Power:

For powering the Canon 450D directly instead of a battery (which will run empty), I found a cheap plastic adapter on the web. This is nothing more than a battery shaped enclosure that just contains a pair of elco's, which should be powered from an external net adapter.




Since I wanted to connect this directly to the 12V outlet of the INDI box, instead I pried the adapter open and used the empty space to put in a cheap tiny buck converter. The voltage setting potentiometer did not work, and was replaced with a suitable fixed resistor, to yield a 7.6V output voltage. I re-used one of the elco's and added a 100nF capacitor for further filtering purposes.

Game controller / Joystick:

For corrections and scanning the sky a gamepad can be used as a joystick. This could be a Super Nintendo SNES controller with a USB interface:

The version I had broke down, so I decided to build a more robust one based on an Arduino Micro and a handful of buttons. You can use any other, as long as it is USB interface and recognized by Ubuntu.

Software configuration

The software setup procedure is chopped-up in several parts, that can be selected independently for installation. This article is more or less a log of how I did it, based on what I collected from the web and the INDI forum: see the links at the end of this article. Attributes in yellow background, used in text or code examples should be customized to your own situation.

The following steps will be followed:

1. Install Ubuntu on the RPi,
2. Setup the networking,
3. Install and configure VNC for virtual desktop access to the Rpi, including MATE desktop,
4. Install the GPS and Time drivers, and add-ons to support the HW configurations
5. Install applications: EKOS/KStars/INDI and drivers, Astrometry, PHD2

1 Ubuntu installation

SD-card preparation

The easiest way to prepare the SDXC card: use Raspbery Pi Imager to download and format/write your card in one go. Select the Pi4, Ubuntu Desktop 22.04.4 LTS (64 bit) and the micro SDXC card plugged into your PC. I use a very fast 64GB one, with the intention to speed up local image handling.

Configuration

The files on the readable part of the SD card (such as `config.txt`, `usercfg.txt`, `syscfg.txt`) can be changed already now, but better at a later stage. After the preparation of Ubuntu they can be found in the directory `/boot/firmware/`.

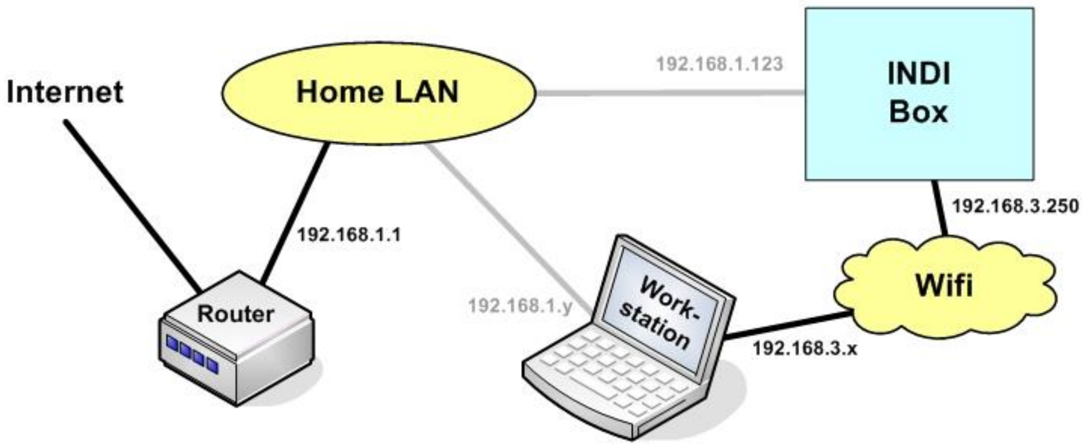
Plug in LAN, display, mouse and keyboard, then boot and let the process finish, while providing the parameters when asked for. Change the default user / password, set autologin upon first boot. Make sure the RPi can reach the internet through your home LAN, during startup. The process takes a while, after a reboot a popup will appear asking to install updates: seems like a good idea. Reboot after updating.

Still some more updating is required:

```
sudo apt update
sudo apt -y upgrade
```

2 Networking

The intended network topology allows for stand-alone operation in the field, but it can also be hooked up to the wired home LAN network. For this reason, the WiFi interface will be set up as an *access point* while upstream traffic is routed to the Ethernet interface. The Ethernet interface address is static, matching the home LAN. In case the Ethernet is not connected, the INDI box operates in stand-alone mode and there will be no internet access. The virtual desktop device (Workstation) normally connects through the WiFi AP, and when the INDI box Ethernet is connected to a host LAN, it can also reach the internet through the INDI box router. Alternatively, the Ethernet interface can be used to connect the virtual desktop device via the home LAN.



Example Network Layout

Basic network settings:

In Ubuntu 22.04, netplan and NetworkManager are actually working quite well, so setting up the network is very easy now:

- Start with setting up the hotspot, through Settings → Wi-Fi and clicking the three dots button in the title bar.
- The ethernet IP address can be set to static in Settings → Network and clicking the cogwheel in the wired box. Fill in manual address (`192.168.1.123`), netmask (`255.255.255.0`) and gateway address (`192.168.1.1`).

A route from the hotspot to the wired network is made automatically.

If the hotspot disappears for some reason, another can be installed by calling the NetworkManager CLI:

```
nmcli dev wifi hotspot ifname wlan0 ssid test password "password"
```

The configuration of the hotspot is in `/etc/NetworkManager/system-connections`.
Make sure that in the Hotspot file `autoconnect=true`.

4 Virtual desktop

Desktop light

Ubuntu uses Gnome as desktop, which is relatively heavy. Therefore this will be replaced by MATE and lightdm to start with. This can be done from the command line in a terminal window:

```
sudo apt -y install mate-desktop-environment lightdm
```

When a command complains about a lock, just reboot. The MATE install will take a while, when asked during MATE installation select **lightdm** as display manager.

Then reboot and login through the **lightdm** popup, after first selecting MATE as desktop (click round icon). Then configure the desktop environment to your liking. While doing this, it is best to *disable the lockscreen* (Control Center → Power Management) and to make sure that *autologin* is enabled for the user account.

Create **/etc/lightdm/lightdm.conf.d/12-autologin.conf**, if it doesn't exist. Add the username in this file:

```
[SeatDefaults]
autologin-user=ubuntu
```

VNC Server

At this moment the RPi board is running Ubuntu, has a light weight Mate desktop environment, provides a WLAN access point and also allows access to the internet. We now need to set up a remote GUI towards the workstation PC. For this purpose, the lightweight VNC server **x11vnc** is used:

```
sudo apt install x11vnc -y
```

Create a startup file **~/.vnc/passwd**, containing:

```
yourpassword
```

Configure VNC with a password:

```
x11vnc -storepasswd
```

Create a startup file **~/.vnc/startvnc.sh**, containing:

```
x11vnc -usepw -shared -display :0 -geometry 1280x768 -forever
xrandr --fb 1280x768
```

Make it executable:

```
chmod 755 ~/.vnc/startvnc.sh
```

In the MATE GUI use **Preferences** → **Startup Applications** to add the script to auto startup, using the full pathname and maybe 5 sec delay. Together with the autologin it makes sure VNC environment is set up and starts auomatically.

Then update **/boot/firmware/config.txt** to force a desktop even when no screen is attached, and set the proper display parameters:

```
hdmi_force_hotplug=1
hdmi_group=2
hdmi_mode=22
```

Thess are working settings for a standard 1280x768 display, but you may need to experiment with video modes.

Also the use of CM4 connector to connect any USB devices is not needed, so comment out:

```
#dtoverlay=dwc2,dr_mode=host
```

Finally, comment out the line loading the MKS overlay, otherwise VNC is very slow:

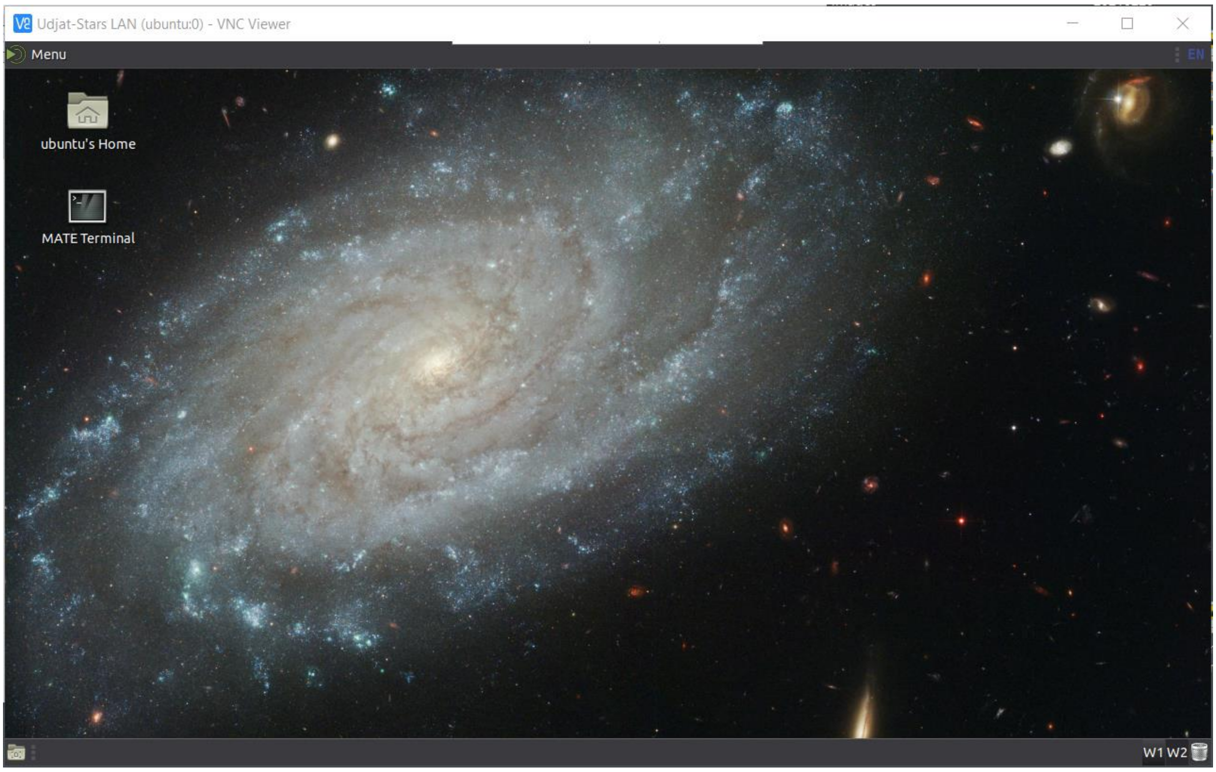
```
#dtoverlay=vc4-kms-v3d
```

Note: The RPi *must* have autologin enabled for the user, otherwise there will be no desktop available for VNC to connect to when in headless mode.

Windows VNC client:

On a workstation you can install any VNC client, but I have used RealVNC successfully in the past. With Ubuntu 20.04 and MATE desktop, Qt applications tended to get corrupted graphics except with RealVNC viewer, so that's what I use now. From Android I have used RealVNC Viewer as well. All you need to do is fill in the RPi IP address (WLAN side, ex: `192.168.3.1`) without any port.

A working connection will give something like:



RPi 4 and Ubuntu (20.04) + MATE desktop seen remotely in RealVNC

From here everything works on GUI basis, as if connected locally with a display and keyboard/mouse.

3 Extensions

Some extensions of the installation are needed to connect to the specific hardware and to provide some of the necessary services.

Kernel configuration

During boot the file `config.txt` is executed, and this file as supplied with the Ubuntu 20.04 image includes a number of other files, notably `syscfg.txt` and `usercfg.txt`. In 22.04 this is no longer the case and all modifications are made directly in `config.txt`.

We need to make sure that UART2 and UART5 can be used as UARTs for GPS input and NEQ6 control respectively. This means the right overlays must be loaded and any other use of the UARTs must be disabled. Moreover, we need a GPIO pin to be configured as PPS input.

So edit `config.txt` as follows:

To enable a serial console at boot time, after loading the kernel (`kernel=`…), add line:

```
enable_uart=1
```

At the end of the file, load the overlays to configure the GPIO usage:

```
dtoverlay=disable_bt
dtoverlay=pps-gpio,gpiopin=18
dtoverlay=uart2
dtoverlay=uart5
```

The pps kernel module needs to be loaded during boot, add this line to `/etc/modules`:

```
pps-gpio
```

After a reboot UART2 and UART5 will be visible as a new `/dev/ttyAMA1` and `/dev/ttyAMA2` device respectively. Also, there is now a `/dev/pps0` device. The permissions should be 660 and the group must be `dialout`. If not, use `chgrp` and `chmod` to change these, so e.g.:

```
sudo chgrp dialout /dev/pps0
sudo chmod 660 /dev/pps0
```

Check that the commandline file `cmdline.txt` does not link the console to `ttyAMA1` or `ttyAMA2`, but for example:

```
console=serial0
```

Finally, it is best to also check in `/dev` whether any symbolic links are made to these devices, and if so, remove them. Typically `serial0` is linked to `ttyAMA0`.

Connecting the GPS receiver

If not already the case (check with `groups`), the user must be added to the `dialout` group to obtain access to serial ports:

```
sudo adduser ubuntu dialout
```

This might require a reboot to take effect.

Since the system will run without internet connection, a separate source for time and position needs to be used (or you would have to set these manually). A U-Blox Neo-8M GPS module is connected to the HW UART2 for this purpose. The use of the PPS signal can be enabled if the GPS module generates this. This can be done by activating overlays that have already been configured.

The **/dev/pps0** as well as the **/dev/ttyAMA1** devices must be available and free to use. The **ttyAMA1** device is connected to HW **UART2**, and available on pins 27 and 28 (GPIO 0 and 1). The **pps0** device is available on pin 12 (GPIO 18). The GPS receiver must be properly connected to these pins (see HW section).

To change tty settings use the **stty** command, for example to switch off echo or set speed (see **man stty** for options):

```
sudo stty -F /dev/ttyAMA1 9600 -echo
```

The working of the GPS receiver can now be checked, assuming it is connected, by typing:

```
cat </dev/ttyAMA1
```

The output of the GPS receiver will scroll over the screen when all is in order.

Now **gpsd** needs to be set up to serve as time/position reference. First install **gpsd** and clients:

```
sudo apt install gpsd gpsd-clients
```

Then, edit the daemon file **/etc/default/gpsd** to contain the following:

```
START_DAEMON="true"
USBAUTO="false"
DEVICES="/dev/ttyAMA1 /dev/pps0"
GPSD_OPTIONS="-n"
GPSD_SOCKET="/var/run/gpsd.sock"
```

After a reboot, when **cgps** or **gpsmon** is used, the screen should show status info and running messages from the GPS. Depending on GPS device and location it might take a while to establish first fix.

Note: When the gpsd service is not started (check with **service gpsd status**), it may be done manually in the user home-directory file: **.profile**. Add towards the end:

```
sudo stty -F /dev/ttyAMA1 9600 -echo
sudo stty -F /dev/ttyAMA2 9600 -echo
```

To test gpsd (this will not auto start at reboot, see further below):

```
sudo service gpsd start
```

See https://gpsd.gitlab.io/gpsd/troubleshooting.html for troubleshooting gpsd.

Time server

When the GPS receiver works, a time server must be installed that can synchronize system time with **gpsd**. Use **chrony** for this, which is a newer NTP server implementation than **ntpd**.

```
sudo apt -y install chrony
```

Change the configuration file **/etc/chrony/chrony.conf** to use local time reference (i.e. GPS). Comment all references to internet NTP servers, like for example:

```
#pool 2.debian.pool.ntp.org offline iburst
```

Uncomment or add the line defining a local reference:

```
local stratum 10
```

Add lines to indicate the **gpsd** output as reference:

```
# set larger delay to allow the NMEA source to overlap with
# the other sources and avoid the falseticker status
refclock SHM 0 refid GPS precision 1e-1 offset 0.9999 delay 0.2
refclock SOCK /var/run/chrony.ttyAMA1.sock refid PPS
```

Change this line to force time update when the difference is more than 1sec:

```
makestep 1 -1
```

Save the file and make **gpsd** service to run automatically at boot time:

```
sudo systemctl enable gpsd.service
```

Now reboot the RPi. Check whether the services are running:

```
service --status-all
```

There should be a **(+)** indication next to **chronyd** and **gpsd**.

Use **gpsmon** or **cgps** to see whether the **gpsd** has a fix and gets PPS. Then use **chronyc** to see whether **chronyd** has the right time. If everything runs as it should, the system clock should be set to the right value.

Setting your timezone is done with **datetimectl**:

```
timedatectl list-timezones
sudo timedatectl set-timezone Europe/Amsterdam
```

FTP server

To enable file exchange (e.g. photo's) it is helpful to have an FTP server running. This can be enabled by installing Very Secure FTP daemon (**vsftpd**) on the RPi:

```
sudo apt install vsftpd
```

After rebooting the FTP daemon runs and can be accessed from the laptop with for example the *FileZilla* client.

SSH server

Install OpenSSH server:

```
sudo apt install -y openssh-server
```

5 Applications

KStars/EKOS/INDI:

To install the Kstars/EKOS/INDI suite on the RPi, make sure it is connected to the internet and from a CLI (SSH or terminal) enter:

```
sudo apt-add-repository -y ppa:mutlaqja/ppa
sudo apt update
sudo apt -y install indi-full kstars-bleeding
```

Note:

Sometimes KStars or indilib are not completely installed, then it can be done again by executing once more:

```
sudo apt update
sudo apt upgrade indi-full kstars-bleeding
```

Configuration hints:

Use the Ubuntu **`gpsd`** service as direct source of location and time for EKOS, if not installed already:

```
sudo apt install -y indi-gpsd
```

In EKOS, select the **`gpsd`** device under *Aux* → *Others*

The mount to be used is *SkyWatcher* → *EQMod*. To get the mount working, the right port has to be chosen to connect. The device **`ttyAMA2`** is connected to HW **`UART5`**, and available on pins 32 and 33 (GPIO12 and 13). This serial port device needs to be selected in the *EQMod Mount / Connection* tab in the Kstars control panel.

Astrometry:

The Astrometry plate solver can be used to accurately align the telescope in a very easy way. After rough polar alignment, go to a known object. The plate solver takes a DLSR image and tries to match it with stored and indexed images. When successful, the fix can be used to sync the mount.

Which files you actually need depends on the telescope resolution; the lower numbers have smaller tiles:

- **`astrometry-data-2mass-00`** 2' – 2.8'
- **`astrometry-data-2mass-01`** 2.8' – 4'
- **`astrometry-data-2mass-02`** 4' – 5.6'
- **`astrometry-data-2mass-03`** 5.6' – 8'
- **`astrometry-data-2mass-04`** 8' – 11'
- **`astrometry-data-2mass-05`** 11' – 16'
- **`astrometry-data-2mass-06`** 16' – 22'
- **`astrometry-data-2mass-07`** 22' – 30'
- **`astrometry-data-2mass-08-19`** 30' – 2000'

The file **`astrometry-data-2mass`** will load them all, but this is a total of many GB.

Per package installation:

```
sudo apt -y install astrometry-data-tycho2
sudo apt -y install astrometry-data-2mass-08-19
sudo apt -y install astrometry-data-2mass-07
sudo apt -y install astrometry-data-2mass-06
sudo apt -y install astrometry-data-2mass-05
sudo apt -y install astrometry-data-2mass-04
sudo apt -y install astrometry-data-2mass-03
sudo apt -y install astrometry-data-2mass-02
```

Alternatively, download the debian packages from http://data.astrometry.net/debian , FTP and install with:

```
sudo apt -y install astrometry-data-*.deb
```

To be able to use this, also the plate solving software itself must be installed, this may already have been done with the KStars/indi package:

```
sudo apt -y install astrometry.net
```

PHD2:

PHD2 is maintained by **http://openphdguiding.org/** but only for Windows and MacOS. The Linux variants are maintained by Patrick Chevally on his launchpad:

```
https://launchpad.net/~pch/+archive/ubuntu/phd2
```

To install, enter the following:

```
sudo add-apt-repository ppa:pch/phd2
sudo apt update
sudo apt -y install phd2
```

SkyChart

Find the website tyo get the instructions for installation:

https://www.ap-i.net/skychart/en/documentation/installation_on_linux_ubuntu

CCDCiel, ASTAP, …

To Be Completed

Other settings

For the game controller a driver is already available in the Ubuntu installation, for testing the controller the package jstest can be used:

```
sudo apt -y install jstest-gtk
```

This test package shows the numbers for the different buttons, to which KStars will refer.

Astroberry

As an open-source alternative to e.g. Stellarmate or ASIAir, which basically are also RPi based devices, Astroberry server provides a complete SD card image that can easily be installed in a RPi 4. Unfortunately it is not updated anymore, so new application SW lacks the required support.

Some References

The INDI tutorials ("Painless remote control with Ekos/INDI")

http://indilib.org/support/tutorials.html

INDI Forum, (search for Pi 4)

https://www.indilib.org/forum/index.html

Ubuntu 22.04 for the RPi4 (64 bit):

https://ubuntu.com/download/raspberry-pi

Terminal emulator for SSH, PuTTY:

http://www.putty.org/

Writing a disk image to SD Card, Win32DiskImager:

https://sourceforge.net/projects/win32diskimager/

PHD2 (Ubuntu/INDI):

https://launchpad.net/~pch/+archive/ubuntu/phd2