Remote-Control for Telescope

Arjan te Marvelde, initial version Jan 2016, this version Apr 2020

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 article describes a self-contained control station that can be mounted on or in the RA unit of an NEQ-6 mount. It can be left alone to make photos through 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 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 article contains the following sections:

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

This description results in a remote-control setup with the following items:

**INDI box**

- **RPi4** with a fast 64GB SD, running the latest Ubuntu, providing a **WiFi 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 **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 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 directly through a serial interface,
- A **Nintendo** game controller **joystick** (SNES) is connected through USB,
- A **ZWO ASI385MC autoguider** camera also connected through USB.

**User interface**

- A **Laptop** running Windows, which has a wireless (or wired) connection to the INDI box.
- The Windows laptop hosts a **VNC client** (TightVNC) enabling GUI access to the RPi desktop. This laptop could be replaced with any other device hosting a VNC client.
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 running on Linux is KStars / EKOS.

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. On top of that the GPS receiver and antenna are mounted, and to the left the 3A SMPS for the RPi4 power provisioning. The hardware setup is described in more detail towards the end of the document.

The INDI box

My 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 since the RPi-4 generates more heat than its predecessors.
**Indi inside NEQ6**

Currently the lot has been integrated 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 extended a little further into the housing, so the original connectors etc could be left intact. The housing appears to have plenty space to allow this.

In this case a smaller size DC buck converter was used, with even better specs (small pcb on the right). The GPS is now a UBlox NEO8 (small pcb on the left), which is a big improvement over the NEO6. I put the antenna into a replacement cap for the polar alignment scope. You can still take this out for alignment.

All plastic parts are 3D printed in PLA.
**Serial interfaces:**

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

The GPS receiver is a UBlox (NEO-6M or better NEO-8M), providing 3V3 UART and PPS interfaces. These can be directly connected to the UART0 of the RPi as follows:

- **Vcc:** IO pin 1,
- **GND:** IO pin 14,
- **Tx:** IO pin 10,
- **Rx:** IO pin 8,
- **PPS:** IO pin 12

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 on IO pins 32 and 33 is used to directly connect the NEQ6 control interface inside the NEQ6 motor housing (top middle). 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, the NEQ6 interface is TTL (5V) and the RPi is LVTTL (3V3). When connecting different levels a level converter is needed for each lead, as for example in above image (bottom right).
**Rpi4 GPIO assignments:**

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

<table>
<thead>
<tr>
<th>Pin</th>
<th>GPIO / Function</th>
<th>Default Resistor State</th>
<th>ALT0</th>
<th>ALT1</th>
<th>ALT2</th>
<th>ALT3</th>
<th>ALT4</th>
<th>ALT5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3V3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>UP</td>
<td>SDA1</td>
<td>SA3</td>
<td>LCD_VSYNC</td>
<td>SPI3_MOSI</td>
<td>CTS2</td>
<td>SDA3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>UP</td>
<td>SCL1</td>
<td>SA2</td>
<td>LCD_HSYNC</td>
<td>SPI3_SCLK</td>
<td>RTS2</td>
<td>SCL3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>UP</td>
<td>GPCLK0</td>
<td>SA1</td>
<td>DPI_D0</td>
<td>SPI4_CEO_N</td>
<td>TXD3</td>
<td>SDA3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>DOWN</td>
<td>TXD0</td>
<td>SD6</td>
<td>DPI_D10</td>
<td>SPI5_MOSI</td>
<td>CTS5</td>
<td>TXD1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>DOWN</td>
<td>RXD0</td>
<td>SD7</td>
<td>DPI_D11</td>
<td>SPI5_SCLK</td>
<td>RTS5</td>
<td>RXD1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>DOWN</td>
<td>FL1</td>
<td>SD9</td>
<td>DPI_D13</td>
<td>RTS0</td>
<td>SPI1_CE1_N</td>
<td>RTS1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>DOWN</td>
<td>PCM_CLK</td>
<td>SD10</td>
<td>DPI_D14</td>
<td>SPI6_CEO_N</td>
<td>SPI1_CEO0</td>
<td>PWM0</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>DOWN</td>
<td>SD0_DATA3</td>
<td>TE1</td>
<td>DPI_D23</td>
<td>SD1_DATA3</td>
<td>ARM_TMS</td>
<td>SPI6_CE1_N</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>DOWN</td>
<td>SD0_CLK</td>
<td>SD14</td>
<td>DPI_D18</td>
<td>SD1_CLK</td>
<td>ARM_TRST</td>
<td>SDA6</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>DOWN</td>
<td>SD1_CMD</td>
<td>SD15</td>
<td>DPI_D19</td>
<td>SD1_CMD</td>
<td>ARM_RESET</td>
<td>SCL6</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>DOWN</td>
<td>SD0_DAT0</td>
<td>SD16</td>
<td>DPI_D20</td>
<td>SD1_DAT0</td>
<td>ARM_TDO</td>
<td>SPI3_CE1_N</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>DOWN</td>
<td>SPI0_MISO</td>
<td>SD2</td>
<td>DPI_D6</td>
<td>I2C5L_SDA_MOSI</td>
<td>CT54</td>
<td>SDA5</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>DOWN</td>
<td>SPI0_MISO</td>
<td>SD1</td>
<td>DPI_D5</td>
<td>I2C5L_SDI_MISO</td>
<td>RXD4</td>
<td>SCL4</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>DOWN</td>
<td>SPI0_DAT1</td>
<td>SD17</td>
<td>DPI_D21</td>
<td>SD1_DAT1</td>
<td>ARM_TCK</td>
<td>SPI4_CE1_N</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>DOWN</td>
<td>SPI0_SCLK</td>
<td>SD3</td>
<td>DPI_D7</td>
<td>I2C5L_SCL_SCLK</td>
<td>RTS4</td>
<td>SCL5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>UP</td>
<td>SPI0_CEO_N</td>
<td>SD0</td>
<td>DPI_D4</td>
<td>I2C5L_CE_N</td>
<td>TXD4</td>
<td>SDA4</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>UP</td>
<td>SPI0_CE1_N</td>
<td>SD1</td>
<td>SPI4_SCLK</td>
<td>RTS3</td>
<td>SCL4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>UP</td>
<td>SDA0</td>
<td>SA5</td>
<td>PCLK</td>
<td>SPI3_CEO_N</td>
<td>TXD2</td>
<td>SDA6</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>UP</td>
<td>SCL0</td>
<td>SA4</td>
<td>DE</td>
<td>SPI3_MISO</td>
<td>RXD2</td>
<td>SCL6</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>UP</td>
<td>GPCLK1</td>
<td>SA0</td>
<td>DPI_D1</td>
<td>SPI4_MISO</td>
<td>RXD3</td>
<td>SCL3</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>UP</td>
<td>GPCLK2</td>
<td>SOE_N_SE</td>
<td>DPI_D2</td>
<td>SPI4_MOSI</td>
<td>CT53</td>
<td>SDA4</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>DOWN</td>
<td>PWM0</td>
<td>SD4</td>
<td>DPI_D8</td>
<td>SPI5_CEO_N</td>
<td>TXD5</td>
<td>SDA5</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>DOWN</td>
<td>PWM1</td>
<td>SD5</td>
<td>DPI_D9</td>
<td>SPI5_MISO</td>
<td>RXD5</td>
<td>SCL5</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>DOWN</td>
<td>PCM_F5</td>
<td>SD11</td>
<td>DPI_D15</td>
<td>SPI6_MISO</td>
<td>SPI1_MISO</td>
<td>PWM1</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>DOWN</td>
<td>FL0</td>
<td>SD8</td>
<td>DPI_D12</td>
<td>CTS9</td>
<td>SPI1_CE1_N</td>
<td>CT51</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>DOWN</td>
<td>SD0_DAT2</td>
<td>TE0</td>
<td>DPI_D22</td>
<td>SD0_DATA2</td>
<td>ARM_TDI</td>
<td>SPI5_CE1_N</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>DOWN</td>
<td>PCM_DIN</td>
<td>SD12</td>
<td>DPI_D16</td>
<td>SPI6_MOSI</td>
<td>SPI1_MOSI</td>
<td>GPCLK0</td>
<td></td>
</tr>
</tbody>
</table>

To connect the GPS receiver for example, the default mapping of TX0/RX0 is used (to pins 8/10, BCM GPIOs 14/15). To enable another UART interface, for example TX5/RX5 can be mapped to pins 32/33 (corresponding to Broadcom GPIOs 12/13).

A way to accomplish this is by activating predefined Device Tree overlays from config.txt.
**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:**

For corrections and scanning the sky a gamepad can be used as a joystick. In my case this is a Super Nintendo SNES controller with a USB interface:
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, including MATE desktop
2. Setup the networking
3. Update Ubuntu and add specific packages
4. Install and configure VNC for virtual desktop access to the RPi
5. Install applications: EKOS/KStars/INDI and drivers, Astrometry, PHD2

1 Ubuntu installation

SD-card

Download latest Ubuntu server for RPi4 (for 32 bit it is now Ubuntu 19.10.1) from the ubuntu.com website, unpack it with 7-zip and write it to a fresh Micro-SD (using e.g. Win32DiskImager) according to instructions that can be found on the web. I use 64GB very fast one, esp. to speed up local image handling.

Note: Reclaiming the SD-card from a previous installation works as follows:

- Plug the SD card into the PC
- Run the DISKPART tool from a cmd box
- Enter LIST DISK and determine which of the listed disks is the SD card
- Then enter SELECT DISK x, where x must be the correct disk number!
- Then enter CLEAN to wipe the disk (this is an irreversible operation!)
- Finally use CREATE PARTITION PRIMARY to create the partition
- FORMAT QUICK finally formats the new partition

The SD card is now ready to write the Ubuntu image using Win32DiskImager.

Configuration

The file config.txt on the SD card can be edited, when still in the PC, to change the boot behavior. Ubuntu startup may hang in U-Boot, in that case the file should be adapted to force boot from the RPi4 EEPROM (not present in earlier variants!).

Under the [all] tag add / modify the lines:

```plaintext
kernel=vmlinuz
initramfs initrd.img followkernel
#device_tree_address=0x03000000
```

This file, or other configuration files, can be changed afterwards from Ubuntu in the directory /boot/firmware/ whenever required.

Plug in internet, display, mouse and keyboard, then boot and let it finish. Login as ubuntu, with password ubuntu. This password needs to be changed upon first boot. Make sure the RPi can reach the internet through your home LAN during startup. The process takes a while, but when it is completely finished, updates can be fetched, crap deleted and MATE installed as desktop environment:

```plaintext
sudo apt update
sudo apt -y upgrade
sudo apt -y purge unattended-upgrade cloud-init
sudo apt -y install mate-desktop-environment lightdm
```
When a command complains about a lock, just reboot. Both upgrade and MATE install will take a while, when asked select **lightdm** as display manager.

Then reboot and login through the **lightdm** interface, selecting MATE as desktop. Then configure the desktop environment to your liking. Make sure that **autologin** is enabled for the user account. Also, it is convenient to **disable the lockscreen** (under power settings).

Add the username in the file `/etc/lightdm/lightdm.conf`:

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

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
```

Now the RPi is running Ubuntu and MATE desktop environment, and it is ready for further configuration.

Install some handy utilities and remove some cloud crap:

```
sudo apt -y install chromium-browser
sudo apt -y install net-tools
sudo rm -rf /etc/cloud/
```

```
sudo rm -rf /var/lib/cloud/
```

**Note**: Instead of MATE, other desktop environments can be installed, like Kubuntu (KDE) or Gnome:

```
sudo apt -y install kde-plasma-desktop plasma-nm lightdm
sudo apt -y install gnome-shell-extension-dash-to-panel gnome-system-monitor lightdm
```

**Note**: If KDE was chosen as desktop environment, it may not start up. To repair, in the directory `/usr/share/lightdm/lightdm.conf.d` change the user session in the configuration **40-kde-plasma-kf5.conf**:

```
[SeatDefaults]
user-session=plasma
```
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 also be used to connect the virtual desktop device via the home LAN.

Basic network settings:

Two RPi network interfaces need to be configured, the Ethernet and the WiFi. First obtain the logical names given by Ubuntu:

```
ifconfig
```

This should give the local loopback (lo), an Ethernet interface name usually starting with ‘e’ and a WLAN interface name usually starting with ‘w’ (for example: eth0 and wlan0).

In the 18.04 build, netplan is the network managing mechanism; remove it since it doesn’t add anything to using ifupdown and wireless AP is badly supported:

```
sudo systemctl stop networkd-dispatcher
sudo systemctl disable networkd-dispatcher
sudo systemctl mask networkd-dispatcher
sudo apt-get purge nplan netplan.io
```

Also remove network-manager

```
sudo apt remove --purge network-manager-gnome network-manager
```

The alternative ifupdown can be installed, as well as bridge-utils, hostapd and dnsmasq:

```
sudo apt install ifupdown
sudo apt install -y bridge-utils dnsmasq hostapd
```

Now configure the network for a static interface addresses and a WiFi Access Point. The file /etc/network/interfaces needs to be changed to define the static addresses for wlan0 and eth0 interfaces:
sudo nano /etc/network/interfaces

The relevant contents of the file should look like (addresses need to be adapted):

```plaintext
# The loopback interface
auto lo
iface lo inet loopback

# The ethernet interface
auto eth0
iface eth0 inet static
derived-address 192.168.1.123/24
gateway 192.168.1.1
dns-nameservers 8.8.8.8 8.8.4.4

# The wireless interface
auto wlan0
iface wlan0 inet static
derived-address 192.168.3.1/24
```

The RPi can now be connected to the home LAN and accessed through e.g. SSH.

**Wireless Access Point:**

To be able to use the system in stand-alone mode, the RPi is configured as a Wireless Access Point.

Configure `hostapd` by creating `/etc/hostapd/hostapd.conf` with the following lines:

```plaintext
interface=wlan0
driver=nl80211
ssid=YOUR-SSID
hw_mode=g
channel=6
ieee80211n=1
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_passphrase=YOUR-PASSPHRASE
rsn_pairwise=CCMP
```

Then point the host accesspoint service to its configuration file by editing the following line in file `/etc/default/hostapd`:

```plaintext
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Likewise, edit the file `/etc/init.d/hostapd` to contain:

```plaintext
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Configure the DHCP server for the AP interface, so attached clients will be given an IP address.

First backup `/etc/dnsmasq.conf` and then replace or append the contents by:
```
interface=wlan0
listen-address=192.168.3.1

# Bind to the interface to make sure we aren't sending things elsewhere
bind-interfaces

# Forward DNS requests to Google DNS
server=8.8.8.8

# Don't forward short names
domain-needed

# Never forward addresses in the non-routed address spaces.
bogus-priv

# Address range for clients with 12h lease time
dhcp-range=192.168.3.50,192.168.3.100,12h
```

Finally start the services:

```
sudo systemctl unmask hostapd
sudo update-rc.d hostapd enable
sudo service hostapd start
sudo systemctl unmask dnsmasq
sudo service dnsmasq start
```

After a reboot you will be able to detect the RPi wireless network you just created, and to connect to it.

**DNS:**

To repair DNS issues, stop the `resolved` daemon:

```
sudo systemctl stop systemd-resolved
```

Then add the dns server addresses to `/etc/systemd/resolved.conf`:

```
DNS=8.8.8.8 8.8.4.4
```

Finally restart the `resolved` daemon:

```
sudo systemctl start systemd-resolved
```

The RPi is now accessible through either Ethernet or WiFi interface, but the interfaces are not yet linked together. For this IP forwarding needs to be enabled.

**IP forwarding:**

The default gateway (defined in `/etc/network/interfaces`) is added as an entry in the routing table as well as the forwarding rules are loaded from `iptables`. The IP forwarding itself must be enabled, so that packets will be actually forwarded from one interface to the other.

At the end of `/etc/network/interfaces` add the line that loads the forwarding rules:

```
post-up iptables-restore < /etc/iptables.rules
```

In the file `/etc/sysctl.conf` the following line must be uncommented and/or changed to:

```
net.ipv4.ip_forward=1
```

Then commit the `sysctl` change with:
sudo sysctl -p

Finally, the forwarding behavior itself must be defined in **iptables**:

```
sudo iptables --policy INPUT ACCEPT
sudo iptables --policy FORWARD ACCEPT
sudo iptables --policy OUTPUT ACCEPT
sudo sh -c "iptables-save > /etc/iptables.rules"
```

Normally these policies are already defined. They make the forwarding behavior fully transparent; the rules need to be further tweaked as required. The line that loads these rules into the routing table was already added to `/etc/network/interfaces`.

**Note:** Local names can be saved in `/etc/hosts` for easier resolution, so e.g. add lines like:

```
192.168.3.1  this-pi
```

**Note:** In order to make the RPi wireless network accessible and enable internet access from it, the home LAN router must be configured as well. A static route must be added, pointing to the RPi wireless network `192.168.3.0`, where the next hop node is the wired interface of the RPi `192.168.1.123`.

**Testing:**

From a laptop connected to the wireless RPi network try to **ping google.com** from a command terminal; if this works routing is okay as well as the DNS name resolution.
4 Virtual desktop

At this moment the RPi board should be running Ubuntu, have a nice desktop environment, provide a WLAN access point and also allow access to the internet. Natively, Ubuntu has an SSH server, but we also need to set up a remote GUI on the workstation PC. For this purpose, the VNC server `x11vnc` is used:

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

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. Together with the autologin it makes sure VNC environment is set up.

Then update `/boot/firmware/usercfg.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
```

In my case this is a working setting for a standard 1280x768 display, but you may need to experiment with video modes.

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 TightVNC successfully in the past. Now with Ubuntu 19.10.1 and MATE desktop Qt applications tend to get corrupted graphics, except with RealVNC viewer, so that’s what I use now. From Android I have used VNC 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:

![Image of VNC connection]

*RPi 4 and Ubuntu (19.10.1) + MATE desktop in TightVNC*

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

During boot the file `config.txt` is executed, and this file as supplied with the Ubuntu 19.10.1 image includes a number of other files, notably `syscfg.txt` and `usercfg.txt`. In turn, `syscfg.txt` includes `nobtcfg.txt` and so on. I think it overcomplicates things, but it is what it is.

We need to achieve that UART0 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.

After loading the kernel in `config.txt`, the following sequence is parsed:

```
# The following settings are "defaults" expected to be overridden by the
# included configuration. The only reason they are included is, again, to
# support old firmwares which don't understand the "include" command.
enable_uart=1
cmdline=nobtcmd.txt
include syscfg.txt
include usercfg.txt
```

First, we will change the commandline file `nobtcmd.txt` content to not link console to UART0:

```
net.ifnames=0 dwc_otg.lpm_enable=0 console=ttym1 root=LABEL=writable
rootfstype=ext4 elevator=deadline rootwait fixrtc
```

Then we will load the right overlays in `syscfg.txt` and disable the other assignments:

```
dtoverlay=disable_bt
dtoverlay=pps-gpio,gpiopin=18
dtoverlay=uart0
dtoverlay=uart5
#include nobtcfg.txt
```

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

```
pps-gpio
```

After a reboot UART0 and UART5 will be visible as `/dev/ttyAMA0` and `/dev/ttyAMA1` respectively. Also there is now a and `/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
```

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

**Connecting a GPS receiver**

Since the system will run without internet connection, a separate time source needs to be used (or you would have to set the time manually). A U-Blox Neo-8M GPS module is connected to the HW UART0 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 are delivered with Ubuntu.
The `/dev/pps0` as well as the `/dev/ttyAMA0` devices must be available and free to use. The `ttyAMA0` device is connected to HW `UART0`, and available on pins 8 and 10 (GPIO14 and 15). 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):

```bash
sudo stty -F /dev/ttyAMA0 9600 -echo
```

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

```bash
cat </dev/ttyAMA0
```

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 reference. First install `gpsd` and clients:

```bash
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/ttyAMA0 /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. It can 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:

```bash
sudo stty -F /dev/ttyAMA0 -9600 -echo
sudo stty -F /dev/ttyAMA1 -9600 -echo
sudo service gpsd start
```

### 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`.

```bash
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.ttyAMA0.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` 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```

**Note:** You can also use `gpsd` as direct source of location and time for EKOS, see further down.

**NEQ6 control**

The `ttyAMA1` device is connected to HW UART5, and available on pins 32 and 33 (GPIO12 and 13). This serial port needs to be configured in EKOS as NEQ6 mount. (See also HW section)

**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.
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:

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

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

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

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 DSLR 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 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.

```bash
sudo apt-get install astrometry-data-tycho2
sudo apt-get install astrometry-data-2mass-00-19
sudo apt-get install astrometry-data-2mass-06
sudo apt-get install astrometry-data-2mass-05
```

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:

```bash
sudo apt-get install astrometry.net
```

PHD2:

PHD2 is maintained by [http://openphdguiding.org/](http://openphdguiding.org/) but only for Windows and MacOS. The Linux variants are maintained by Patrick Chevalley on his launchpad:

[https://launchpad.net/~pch/+archive/ubuntu/phd2](https://launchpad.net/~pch/+archive/ubuntu/phd2)

To install, enter the following:
**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:

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

This test package shows the numbers for the different buttons, to which KStars will refer.
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 19.10.1 for the RPi4:**
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