I'm finally writing up the steps for anyone wanting to monitor their Renogy Rover or SRNE (ML2420/ML2430/ML2440) with a Raspberry PI.
With the correct port forwarding on your router and a noip.com (or similar) account, you should be able to see what's happening on your solar controller from anywhere and also download and analyse past data as this is logged to a database.

Please note that I have only tested this on my SRNE ML2440, but I believe it should also work for the Renogy Rover series. That said, even though I have had no problems running this code for the past few months, I make no guarantee that it will work with your controller and by using this software and following these instructions you agree to not hold me accountable in the unlikely event that this causes damage to you or your solar controller, Raspberry PI or any other attached devices.

What this code does:

- Reads the registers from the solar controller every few seconds.
- Read current from an external shunt every few seconds using an external ADS1115 (Adafruit) board. This part of the code could be substituted for an estimate instead if you have anything directly connected to the batteries and don't have a shunt and ADS1115 board. I also have included a reed relay controlled from the Raspberry PI that disconnects the inputs of the ADS1115 from the shunt when the power to the Raspberry PI is off. This fixes an issue where the ADS1115 would sometimes burn out if the power to the Raspberry is disconnected and the ADS1115 inputs were still connected to the shunt and sensing a voltage.
- Writes these values to a SQLite database a couple of times a minute.
- Builds a webpage hosted on the Raspberry PI which shows this data. This is formatted to be basic (with low data use), but look decent on a cellphone (tested in Chrome) and to automatically refresh 2-3 times a minute with updated data. This also allows you to download the hourly backup of the database.
- Updates some LED's attached to the Raspberry PI to indicate how close the batteries are charged in relation to the "daily" usage (Ah) from the shunt and load of the solar controller. At the end of the day these LED's are then kept static overnight and are reset when the solar controller considers it to be daytime the next morning. Note that the daily registers and the shunt's daily usage are cleared around 6-8pm in the evening when the solar controller considers it nighttime. This also doesn't take into account multiple days where the batteries do not achieve full charge, however I have the blue LED set to indicate that a substantial amount of extra charge has been gained, which would then give some indication that the batteries are recovering their charge after any previous cloudy days.
- Uses a relay to switch my router off for a few seconds and then back on again if the internet has gone down for more than a few minutes. My router is powered on the solar controller load terminals via a 24-12V step down DC-DC converter.

Required skills:

- Basic knowledge of Raspbian, using SSH and Nano etc to edit files.
- Soldering up cables and connectors as well as LED's (if you choose to include this part).
- Port forwarding on your router if you want to access the webpage from outside.

Required components:

- Raspberry PI (I used the Raspberry PI 2B, but I believe the Zero W would also work)
- Max3232 board to convert 15V serial from the controller to 3.3V that the PI can handle. I was unable to interface with the solar controller using a cable from it to the USB on the Raspberry PI, so chose to use the header on the PI instead, via the Max3232 and a serial cable. (DO NOT try to wire the serial from the solar controller direct into the PI as it cannot handle 15V!). See https://www.netram.co.za/2837-rs232-to-ttl-converter-max232-ic.html
- Serial cable to connect the solar controller to the Raspberry PI. See RS232.png for the wiring diagram based on the SRNE manual. Rx to Rx and Tx to Tx doesn't seem correct since transmit should go to receive on each end, however this seemed to work for me and it may be that the SRNE manual is incorrect. If it doesn't work for you then try change Rx to Tx and Tx to Rx.
- Various wires and connectors (6-pin RJ11, DB9M etc) to connect to the solar controller, the Max3232 board and the Raspberry PI header. See PiHeader.png.

Optional components:

- Charge status indicated by LED's. Soldered the LED's and resistors to a "Veroboard". See PiHeader.png for how this is wired. Red, Yellow, White, Green, Blue in order from least charged to most.
- ADS1115 (Adafruit), D32A2100 (or similar) reed relay, 2N2222 transistor, 3.9k resistors, wire, veroboard etc to drive reed relay from Raspberry (see included diagram). If you are only powering devices off the load terminals from the solar controller then you don't need this as this is used to monitor the power usage of devices attached to the battery through a shunt (devices that have high power usage). See: http://www.communica.co.za/Catalog/Details/P1007158345
- 2 Relay board (a single relay board should also work). I only have this because my router sometimes loses the internet and stops trying to reconnect. A hard reboot for it is needed. Something like this should work: https://www.netram.co.za/3817-1-channel-5v-relay-module.html

Required Python libraries:

- Pymodbus, required to talk to the controller. Instructions here: https://github.com/riptideio/pymodbus/blob/master/README.rst
- SQlite, required database to store values. Instructions here: http://www.geothread.net/using-sqlite-on-a-raspberry-pi/
- Apache, required web server. Instructions here: https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md

Optional Python libraries:

- ADS1x15, required for the ADS1115. Instructions here: https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/ads1015-slash-ads1115

Instructions:

1. Buy at least the required components listed above.
2. Create the cable shown in RS232.png and wire up the Raspberry header as shown in PiHeader.png with the components you will use.

3. Install Raspbian Stretch (https://www.raspberrypi.org/downloads/raspbian/) and at least the required Python libraries listed above.
4. Copy the attached SolarMonitor.py into your home directory, usually /home/pi/. You will notice that the code is not particularly neat, and maybe not as efficient as it could be, but it works. I have tried to add enough comments that it is not too hard to understand.
5. If you are not using an ADS1115 (and the reed relay and driving components) and want to estimate your loads connected directly to the battery you can modify the "directAmps" variable and comment out anything referring to the ADS1115. If you leave "directAmps" as 0 then some values in the webpage will always show 0.
6. You may also want to comment out the ping command that checks the internet and switches the relay board if you are not using this functionality.
7. In order to get the script to run after booting, run "sudo nano /etc/rc.local" and add "python /home/pi/SolarMonitor.py" (without quotes) above the line "exit 0".
8. After rebooting the code should start automatically but will first flash the charge status LED's (if installed) one by one so you can be sure they are connected correctly.
9. It should then create a database named SolarLog.db in /var/www/html as well as a html/webpage file, solar.html. Note that I have not tested that the create database command in SolarMonitor.py works. Worst case copy it out of SolarMonitor.py and run it manually from the sqlite prompt to create the database.
10. Browsing to http://<ip address>/solar.html should show you the registers from the solar controller (at least the ones I could figure out!), values from the shunt and when the router was last rebooted. It also has a link to download a backup of the database (within the last hour) and view older data. Browsing to http://<ip address>/SolarLog.db will allow you to download the most recent database, but I found that sometimes this is locked and doesn't download properly, which is why the code creates a backup which can then be downloaded with the link.
11. If you plan to set up port forwarding to make this accessible from outside your network, you probably want to make the files in /var/www/html read-only for anonymous access.
12. The code is an infinite loop, where it reads from the registers of the solar controller and ADS1115, updates the webpage and every few loops writes all the values to the database and creates a backup of the database every 600 loops. It will also keep a log of how many loops it's done where it hasn't been able to connect to the internet (tested with a ping). If there is no internet for a few minutes it will hold the relay on for a few seconds, effectively switching the router off and then back on again.
13. I have included a text file with the SQL to select values from the database with "fairly" descriptive column names and also transforming specific status codes (such as the charging mode and temperature) into readable values.

Let me know if you get it working and feel free to ask any questions :)