Python RTTY Module - rtty.py

This library provides access to the PITS+ / PITS Zero RTTY (Radio TeleTYpe) module.

Basic Usage

```
from rtty import *
from time import sleep

print("Create RTTY object")
rtty = RTTY()

print("Send RTTY Sentence")
rtty.send_text("$$PYSKY,1,10:42:56,51.95023,-2.54445,145,8,21.6*EB9C\n")

while rtty.is_sending():
    sleep(0.1)
print("FINISHED")
```

The RTTY object is **non-blocking**, which means that the calling code does not have to wait for slow operations (i.e. sending an RTTY message) to complete. This is vital, since otherwise the tracker program would not be able to do other things (e.g. send a LoRa packet, or take a photograph) whilst an RTTY transmission (which can take 20 seconds or more) to complete. The calling code can either poll to find out when the transmission has completed, or can be notified via a callback.

Reference

Object Creation

```
RTTY(frequency=434.250, baudrate=50)
```

The **frequency** is in MHz and should be selected carefully:

- If you are using LoRa also, it should be at least 25kHz (0.025MHz) away from the LoRa frequency
- It should be different to that used by any other HAB flights that are airborne at the same time and within 400 miles (600km)
- It should be legal in your country (for the UK see https://www.ofcom.org.uk/*data/assets/pdffile/0028/84970/ir*2030-june2014.pdf)

The **baudrate** should be either **50** (best if you are not sending image data over RTTY) or **300** (best if you are).

When setting up your receiver, use the following settings:

- 50 or 300 baud
- 7 data bits (if using 50 baud) or 8 (300 baud)
- no parity
- 2 stop bits

Functions

```
send_packet(packet, callback=None)
```

Sends a binary packet **packet** which should be a **bytes object**. Normally this would be a 256-byte SSDV packet (see the camera.py module).

callback, if used, is called when the packet has been completely set and the RTTY object is ready to accept more data to transmit.

```
send_text(sentence, callback=None)
```

Sends a text string **sentence**. Normally this would be a UKHAS-compatible HAB telemetry sentence but it can be anything. See the telemetry.py module for how to create compliant telemetry sentences.

callback is as for **send_packet()**

```
is_sending()
```

returns **True** if the RTTY module is busy sending data, or **False** if not.

Dependencies

- Needs **pigpio** to be installed
- Needs the standard serial port **/dev/ttyAMA0** to be present and available
- On a Pi 3B, or any other future Pi with the Bluetooth module, needs an appropriate devtree module installed to remap the main serial port to the GPIO pins.

Behind the scenes

RTTY is essentially a radio version of serial (e.g. RS232) communications. On a Pi the easiest way of generating RTTY is to send data to the serial port, and have that control the frequency of a suitable radio transmitter. PITS uses the MTX2 radio transmitter which accepts an input voltage to control (modulate) the radio frequency. The serial port and MTX2 are connected together via a simple resistor network which means that the change of frequency is approx 830Hz.

Testing

Run the supplied test program:

```
python3 test_rtty.py
```