FPV HAT manual V1.0

This manual is valid for the following SKUs:
FPVHAT433
FPVHAT868
FPVHAT915

PLEASE NOTE: We recommend that only Raspberry Pi 3's or 2's are used in connection with our FPV HAT. The original Raspberry Pi cannot be used with our HATs as its 26 pin GPIO connector does not fit the 40 pin GPIO connector mounted on the FPV HAT. Raspberry Pi Zero, A+ and B+ *can* be used with our HATs but at a reduced transfer rates. Raspberry Pi Zero e.g. approx. 420 kbit @ 4GFSK. When we say reduced transfer rates we mean a transfer rate where end-to-end low latency of continous data transfer is maintained. You may be able to set a somewhat higher data rate for discontinuous data / burst transfers on these slower boards. But for maximum data rate for continuous data transfer we recommend using only the Raspberry Pi 2 or 3.

PLEASE NOTE: Keep antennas attached to the SMA antenna connector on both transmitting and receiving Pi+HAT at all times when these are powered on!

FPV HAT Specifications:

- Built-in RF amplifier frequency ranges:
    - 433 Mhz version: 410-450 Mhz,
    - 868 Mhz version: 863-873 Mhz,
    - 915 Mhz version: 905-925 Mhz.
- 1 watt max transmit power.
- Max data rate 500 kbit.
- Software supports AES-256 symmetric key encryption option (via OpenSSL).
- HAT conforms to Raspberry Pi Foundation 2014 HAT standard, incl. EEPROM.
- SMA female antenna connector. 50 Ohms impedance.
- Powered from Rpi 40-pin GPIO and draws 550-600 miliamperes at max. transmit power (1 watt max). Can also be powered via +5VDC and GND pins on HAT. NOTE that you will power both HAT and Pi via this option and that you must provide stable 5V +-0.25V. No USB power cable must be attached to the Pi. This is an experimental option only.
- Software supports 2FSK, 2GFSK and 4GFSK modulations. 4FSK support is partial/experimental so far. Max data rates are approx. 300 kbit for 2FSK/2GFSK and 500 kbit for 4GFSK. 2FSK/2GFSK takes up approx 300 khz bandwith centered on the selected or default frequency. 4GFSK approx 600 Khz (including some of the side-band).
- Software has experimental support for 4/8 FEC (Forward Error Correction).
- Transmit and receive frequencies can be set in software in 500 Khz increments above base frequency (which will be 433/868/915mhz depending on version). Setting the frequency below base frequency is not implemented yet but is coming in future software releases.
- The FPV HAT hardware wireless RF link latency alone is <1 ms
    - BUT please note that raspivid h.264 minimum video latency is approx 100-150ms. This is mainly due to the design of the encoding pipeline in the Raspberry Pi Broadcom h.264 hardware video encoder. The latency is also strongly dependent on e.g. Raspivid h.264 bitrate, fps, quantization and other parameters and can increase dramatically above 150ms if too bandwith-demanding raspivid options are set. TFT displays also have varying latencies which can add to the final camera-to-screen latency as well.

Summary:

Thanks for purchasing our FPV HAT for Raspberry Pi.

This manual describes physical installation and software setup for our FPV HAT.

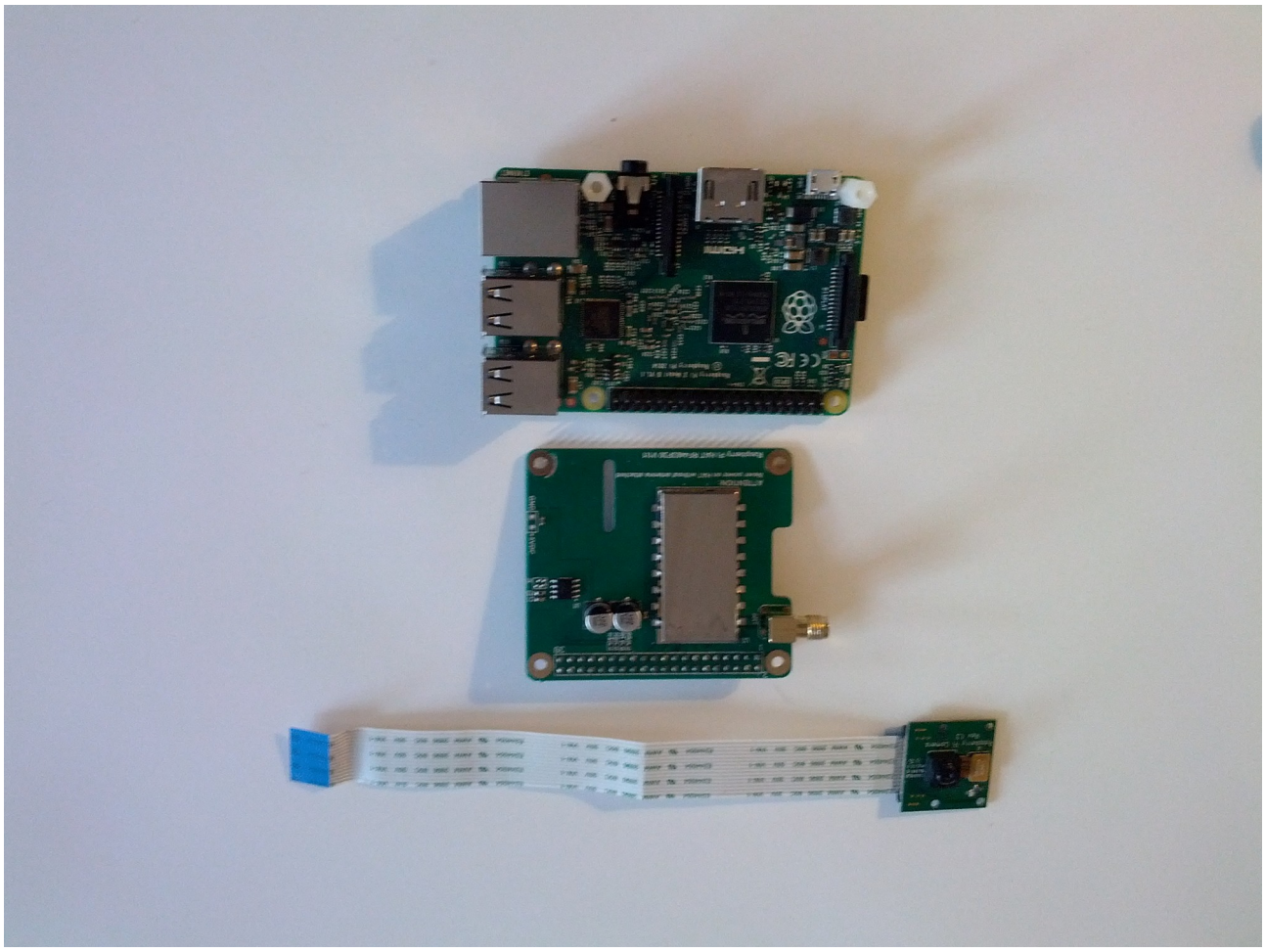
Please note that the software in its current form is **a one-way system**. I.e. data is transmitted from sender Pi to receiving Pi and no data goes the other way. This has enabled us to remove almost all protocol overhead and any possible delays in switching from RX to TX in hardware and software.

All publicly available software related to this project is GPLv3 licensed.

Thanks to Befinitiv and his wifibroadcast system for inspiration!

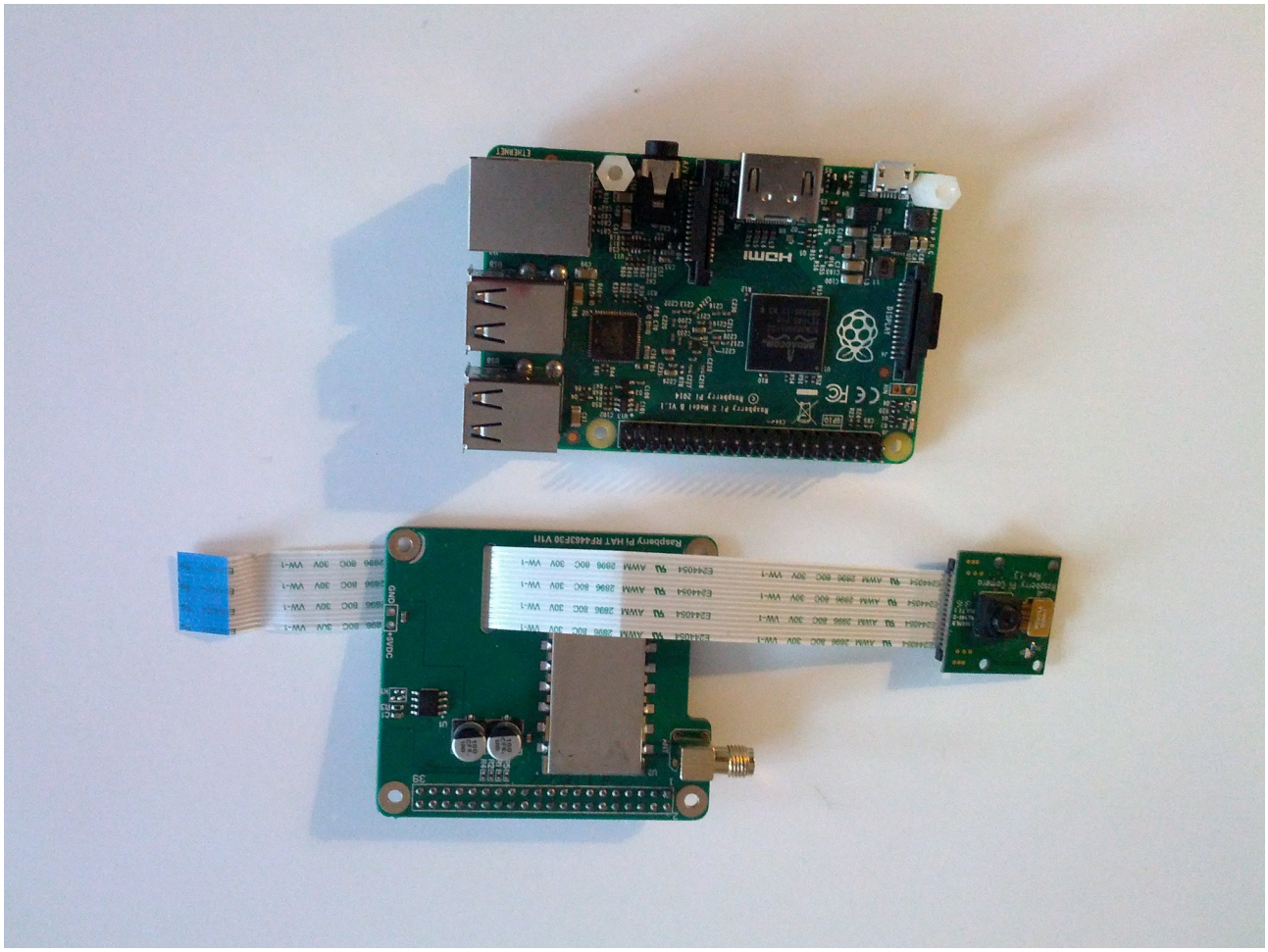
Physical assembly:

On (video) transmitter side you should add both our FPV HAT and the Raspberry Pi Camera (or equivalent compatible camera) to a Raspberry pi board:



Please note that due to the lack of flow-control and protocols, the receive Pi should be equal to or faster than the sending Pi to avoid data buffer "overfill" on receive Pi side. E.g. use a Pi2 as sender and a Pi3 as receiver. Or just use two Pi3's (the built-in WIFI on the Pi3 is very useful for remote setup via ssh). Future versions of the FPV HAT will probably require only Pi3's anyway.

First, the camera ribbon cable is passed though the rectangular hole in the FPV HAT:

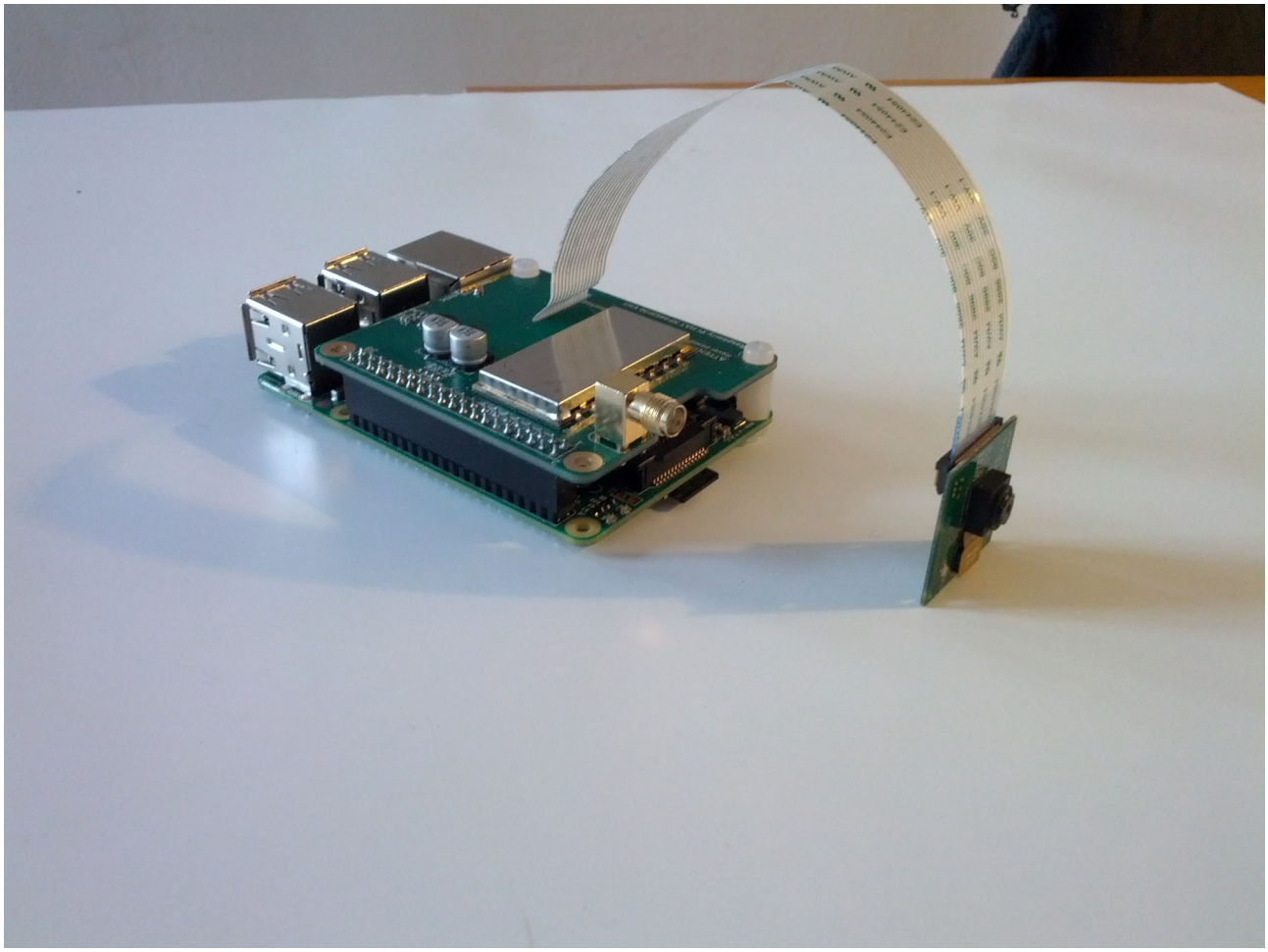


Then the camera ribbon cable is attached to the camera CSI connector on the Raspberry Pi board while at the same time holding the FPV HAT board.

Make sure the the ribbon cable is fully inserted and that the ribbon cable-end is level with the surface of the Raspberry Pi board before pressing hard on the camera connector fastener.

After fastening the camera ribbon cable please align the FPV HAT 40 pin connector with the 40 GPIO pins on the Raspberry Pi board. Then carefully but firmly press the FPV HAT onto the Pi board.

After which it should look like this:



PS: The plastic stand-offs visible in the pictures are not included but are recommend for long term use. Especially if any kind of vibration or physical movement of the transmitter board+pi is expected. I.e. when in use as a FPV solution on RC plane/car/robot/etc or similar.

Repeat the same process on receive Pi side, except for not adding a camera.

Lastly, please attach you antennas to the SMA connectors on the FPV HATs.
DO NOT turn on the Pi's before attaching the antennas!

Software installation:

This software installation guide assumes that you have 2 working Rasberry Pi's already set up, each with internet access. One of these Pi's should also have a Raspberry Pi Camera attached. The Pi with the camera will be the "transmitting pi" and the one without will be the "receiving Pi". You should also have some sort of local display connected directly to the recieving Pi. A HDMI display is recommended as you will then have a fully digital signal path from camera to screen.

Make sure that you have some additional free space on your Pi's SD card ('df') and that your Rasbian installation is up-to-date (by running 'sudo apt-get update' and 'sudo apt-get install') Make sure that you have the full version Rasbian installed and not e.g. the lightweight version. The current full versions of Rasbian have all necesary libraries etc., such as wiring pi, included.

Enter Raspi-config:
sudo raspi-config
Make sure that you have SPI enabled on both Pi's.
Enable camera on the Pi which will used as video transmitter.
Disable any overclocks you might have previously set. Especially on transmitting Pi as the FPV HAT board will emit some extra heat to the air in close proximity to the Pi CPU during transmit.

- While you're at it in raspi-config, it is recommended to enable sshd so that remote login on your Pi is possible. It is highly convenient when experimenting with- and setting up our FPV HAT to work on both Pi's via SSH over WIFI from terminals on e.g. a desktop og laptop PC.

Install the necessary OpenSSL library:
sudo apt-get install libssl-dev

If you haven't already have done so, please install the git utility package:
sudo apt-get install git

Download the software for the FPV HAT onto you Pi:
git clone https://github.com/FernRF/FPV-HAT.git

Change directory to the video-transfer directory:
cd video-transfer

Set correct version to be compiled – this depends on which version you have purchased.
For 868 Mhz version it would be:
git checkout 868mhz
For 433 Mhz version it would be:
git checkout master
For 915 Mhz version it would be:
git checkout 915mhz

Compile:
make

(Or if you compiled already and made some mistake, e.g. forgot to checkout a branch, and must redo this step:)
make clean all

While you are in the video-transfer directory you should download the older binary versions of raspivid and hello_video needed for video transfer (see below for explanation):
wget https://github.com/FernRF/FPV-HAT/blob/master/raspivid_old.bin
wget https://github.com/FernRF/FPV-HAT/blob/master/hello_video_old.bin

Here are the MD5 sums for raspivid_old.bin and hello_video_old.bin:
2052276a7483c0971280901433d69b38 *raspivid_old.bin
a797b0c19a6998582642c86f8dfdb603 *hello_video_old.bin

Check to see that the two binaries downloaded correctly ('ls') and make them executable:
chmod +x raspivid_old.bin hello_video_old.bin

Please change to test dir:
cd test

Test to see if FPV HAT is alive and well:
sudo ./info
This command prints version info from the radio module on the HAT. If you have attached and installed everything correctly in previous steps - and your HAT is working correctly and/or is undamaged - you should always be able to print this info via the ./info command.

If you feel that the radio module on the FPV HAT board gets too hot while idling, you can use this command to put the board to sleep (it will automatically wake up when receiving or transmitting):
sudo ./sleep

Transmitting / receiving video.

Assuming that all steps above were carried out correctly you should now be ready to send and receive video using raspivid and hello_video apps.

We will be using slightly older binary versions of raspivid and hello_video called hello_video_old.bin and hello_video_old.bin.

- This is due to the fact that some newer versions of raspivid have bugs related to displaying low bitrate video without adding huge latency.
  We also use an older version of hello_video since newer versions does not implement receving video data via a simple unix pipe.
  NB: If you want to use the newest version of hello_video you could pipe video output to a fifo buffer and then tell hello_video to display that buffer. The details of that setup falls outside the scope of this guide but this page should give you what you need to set it up: http://archive.is/pb9f4

Before running the sender or receive apps you should shut down any graphical interface (X11/Wayland) running on the Pi to free up CPU and GPU ressources.

Transmitting video:
The command line on the transmitting Pi could be something like this:

```
sudo stdbuf -o 0 ./raspivid_old.bin -n -t 0 -vf -hf -w 512 -h 288 -fps 49 -g 0 -if cyclic -b 460000 --flush -a 512 -pf high -ih -o - | sudo ./sender -m 4 -P 20 -f -
```

To explain each part of this rather long command line:

sudo	executes the following commands with as superuser
stdbuf	
-o 0	sets stream buffering to zero (to avoid possible latency due to Pi system buffers)
raspivid_old.bin	
-n	No preview
-t 0	No timeout on video stream
-vf	Vertically flip image – depends on your physical camera position/set-up.
-hf	Horizontally flip image.
-w 512	512 pixels video with. This may seen very low but remember that our bandwith is limited to max. 500 kbit on the FPV HAT. This number should also be divisable by 16 due to h.264 macroblock size of 16 and should scale nicely on your receive Pi display. YMMW.
-h 288	288 pixels video height. Same considerations as above. 512x288 pixels also corresponds to the 16/9 aspect ratio of most modern displays.

-fps 49	Frame rate set to 49 frames per second. The higher the frame rate the lower the (camera) latency. It also makes motion seem more fluid and pleasant to watch. Pls note that image quality is reduced above 49 fps.
-g 0	This sets the intra-refresh rate of the Pi h.264 hardware encoder to zero. Normally this is not recommended because no intra-refresh means that the image degrades to garbage in the presence of errors. This option has to be set to 0 to avoid a bug in the Pi h.264 hardware encoder firmware which results in very noticeable stutter at the low data rates the FPV HAT is capapable of, everytime an I-frame is sent. BUT in combination with the undocumented -if cyclic option, an I-frame will still be sent every approx. 0.8 sec, thereby compensating for any previous transmit errors. This combination has the added benefit of smoothing out the peaks in the data stream because the -if cyclic option splits the I-frames over a number of P- or B-frames. This is especially useful at low bit rates where all available bandwith must be fully utilized.
-if cyclic	See above.
-b 460000	This is the selected bit rate. The FPV HAT can transfer a maximum of approx. 500 kbit (500000). Please note that the higher the bit rate the shorter the range, and vice versa. A good high data rate which will also give some range is 440-460kbit. NB: The spec sheet for the si4463 module mention 1 mbit as max. bit rate. It may be that the hardware is capable of this but SiLabs EZRadioPro software (used to generate initialization bytes for radio module) only allows a max. setting of 500 kbit anyhow.
-flush	Flush h.264 encoder buffers. May reduce latency.
-a 512	Adds a frame counter to video output.
-pf high	Video quality and compression profile. High is best in most cases.
-ih	Extra headers on frames. Makes sync in case of lost signal easier.
-o -	Send output video stream to stdout (and pipe).
|	Unix pipe
sender	Our sender app which takes video stream input from pipe and sends it to/via radio module on board
-m 4	4GFSK (=2x2GFSK). This is the fastest modulation type available on the FPV HAT. For really long range projects -m 1 (2FSK) or -m 2 (2GFSK) should be set. Max. data rate for -m 1 and -m 2 is approx. 300 kbit. Again, set raspivid -b bit rate option a tad lower at, say, 280 kbit, for better reception/range. -m 3 (4FSK) option is experimental (i.e. works but with errors).
-P 20	Transmit power option. Range between 1 and 127. Above 80-90 produces a lot more heat, relatively speaking, as the rf power output curve is not flat. 127 should give around 1 watt at 5V.
-f -	Sends video stream to stdin/stdout

Receiving video:
On receive side a command compatible with the above transmit command could be:

```
sudo stdbuf -o 0 ./receiver -m 4 -f - | ./hello_video_old.bin
```

Almost same options except for receiver app used. No points for guessing what it does ;)

It is important to specify the same -m (modulation) parameter for both sender and receiver apps.

PS: You may need to start the receiving command line first, before starting the transmitting command line, due a bug in the receving app. This should be fixed soon.

All sender app options:

915 MHz version, usage:
./sender [cspgizhrmfFESkK]
SPI configuration:
-c [0|1] defines SPI channel
-s [500..10000] defines SPI bus speed, kHz
GPIO configuration:
-p PIN defines RPi pin connected to power down control GPIO pin
-g PIN defines RPi pin connected to GPIO1 of Si4463
-i PIN defines RPi pin connected to interrupt output pin
Radio configuration:
-r CHANNEL defines radio channel
-m [1,4] defines modulation type, supported 2fsk (1), 2gfsk (2), 4fsk(3) or 4gfsk (4)
-P level defines power level (1..127)
Encryption:
-E use encryption
-S session ID
-k define key as hex string, key should be 256 bits
-K define salt as hex string
Miscellaneous:
-F use FEC
-q COUNT defines max available packets in incoming queue, packets exceeding the queue size will be dropped
-f FILE defines input/output file, use '-' for stdin/stdout
-u FILE defines console input file
-U FILE creates FIFO to be used as console input file
Help:
-h shows this help

NB: The -r channel option is only usable above default frequency (433/868/915 Mhz). It is set in increments of 500 Khz. E.g. -r 10 adds 5 Mhz to default frequency. On a 433 Mhz version this would result in setting a frequency of 438 Mhz.
It is not possible to go below the default frequency at the moment.

NB: The -F FEC (Forward Error Correction) option uses 4/8 FEC. Meaning 4 FEC bits for every 8 data bits. I.e. using 33 pct of available bandwith for error correction.

NB: -u and -U options can be used to create a separate data stream, alongside the video stream. I.e. for autopilot telemetry or similar. Please note that this is also a one-way stream from transmitter to receiver so it cannot replace a conventional two-way telemetry link.

All receiver app options:

915 MHz version, usage:
./receiver [kKcsSpgirfmFhE]
SPI configuration:
-c [0|1] defines SPI channel
-s [500..10000] defines SPI bus speed, kHz
GPIO configuration:
-p PIN defines RPi pin connected to power down control GPIO pin
-g PIN defines RPi pin connected to GPIO1 of Si4463
-i PIN defines RPi pin connected to interrupt output pin
Radio configuration:
-r CHANNEL defines radio channel
-m [1,4] defines modulation type, supported 2fsk (1), 2gfsk (2), 4fsk (3) or 4gfsk (4)
Encryption:
-E use encryption
-S session ID
-k define key as hex string, key should be 256 bits
-K define salt as hex string
Miscellaneous:
-F use FEC
-f FILE defines input/output file, use '-' for stdin/stdout
-u FILE defines console output file
-U FILE creates FIFO to be used as console output file
Help:
-h shows this help

Additional documentation for other raspivid options:

https://www.raspberrypi.org/documentation/raspbian/applications/camera.md

H.264 video compression:

https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC