Video Stream

Once connected to the GoPro camera, the drone can send a video stream to an other device.

Prerequisites

Request First of all to request the data stream, you have to connect to the WiFi network of the drone and you need to initialize a TCP connection to the controller in order that it know the host.

WiFi To connect to the WiFi network, you only have to turn on the drone and the controller, after that connect your device to WiFi named "SoloLink-_____" and enter the password "sololink".

TCP To initialize the TCP connection you have two possibility : simulate it with an utility like netcat or open a socket with an handmade program. In both cases you have to connect to the port 5502 and the address "10.1.1.1" (IP address of the controller)

```
nc 10.1.1.1 5502
```

or

```
s = socket(PF_INET, SOCK_STREAM, 0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr ("10.1.1.1"); // Address of the server
serv_addr.sin_port = htons (5502); // Port of the server
memset (&serv_addr.sin_zero, 0, sizeof(serv_addr.sin_zero));
connect (s, (struct sockaddr *)&serv_addr, sizeof serv_addr);
```

Protocol

As previously said, the video stream is broadcast using the UDP protocol. In our case, we listen packets from 10.1.1.1 (controller IP address) on the port 5600. The format of a UDP frame is in the following form :

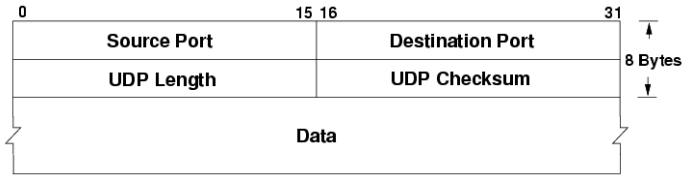

FIGURE 1 – Construction of an UDP frame

Encapsulated into this protocol we find the RTP/AVP protocol. This one is done for delivering audio and video. In our case we only deliver video, so we use a payload of type dynamic for only video (H264 AVC) which matches with the MPEG-4 format. More, the clock rate is of 90000 Hz.

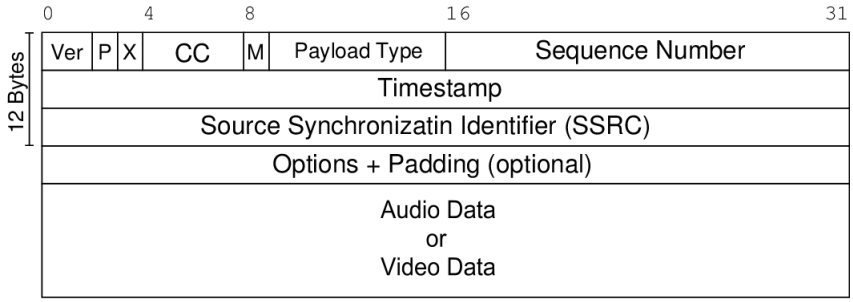

FIGURE 2 – Construction of an RTP frame

Get video stream

To get the video stream you have several possibilities :

. You can directly get it with a software like VLC, opening a file named "sololink.sdp" with the following lines :

```
c=IN IP4 10.1.1.1
m=video 5600 RTP/AVP 96
a=rtpmap:96 H264/90000
t=0 0
```

. An other way is to create a UDP server in an handmade program, listening on the port 5600 and interpret the received frames. This method remains the most complicated to set up.

```
s2 = socket (PF_INET, SOCK_DGRAM, 0);
locAddr.sin_family = AF_INET;
locAddr.sin_addr.s_addr = INADDR_ANY;
locAddr.sin_port = htons (5600); // Listening port
memset (&locAddr.sin_zero, 0, sizeof(locAddr.sin_zero));
bind (s2, (struct sockaddr *)&locAddr, sizeof locAddr);

memset(buf, 0, BUFFER_LENGTH);
recsize = recvfrom(s2, (void *)buf, BUFFER_LENGTH, 0, (struct sockaddr *)&serv_addr, &
    fromlen);
```

. Third possibility, use the library GStreamer and more specialy in this case his launch command. To install this library refer to the installation manuals in the sources. Thus with the command line below we obtain and display the video stream in the same way than with VLC.

```
gst-launch-1.0 -v udpsrc port=5600 caps = "application/x-rtp, media=(string)video, clock-
    rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! rtph264depay ! decodebin !
     videoconvert ! autovideosink
```

. Last solution, use the library GStreamer again but in this case in an handmade program. To understand each part of the program, we advise you to refer to the documentation in the sources but the main thing to know is that to get the same result than before we only have to call the function "gst_parse_launch" where we put the exactly same command line in parameter. The "gst_init" function must be call before all and can have "NULL" parameters. "gst_element_get_bus" is use

to put the pipeline on the bus, "gst_element_set_state" to start playing the video and the "loop" functions are used to initialize and run the video stream.

```
/* Initialize GStreamer */
gst_init(&argc, &argv);

/* Build the pipeline */
pipeline = gst_parse_launch("-v udpsrc port=5600 caps = \"application/x-rtp, media=(string)
    video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96\" ! rtph264depay
    ! decodebin ! videoconvert ! autovideosink", NULL);
bus = gst_element_get_bus(pipeline);

/* Start playing */
ret = gst_element_set_state(pipeline, GST_STATE_PLAYING);

/* Loop */
loop = g_main_loop_new(NULL, FALSE);
g_main_loop_run(loop);
```

Save video stream

In order to reuse or interpret video stream, we give the posibility to save it in three forms.

Video file First solution, register it in a video file (.avi in this case). To be done, we use again the library gstreamer where we redirect the input flow (udpsrc) to a file with the following sequence : "! queue ! avimux ! filesink location=capture.avi". Thus the string to put in gst_parse_launch become :

```
/* Build the pipeline */
pipeline = gst_parse_launch("-v udpsrc port=5600 caps = \"application/x-rtp, media=(string)
    video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96\" ! rtph264depay
    ! decodebin ! tee name=t ! queue ! videoconvert ! autovideosink t. ! queue ! avimux ! filesink
    location=capture.avi", NULL);
```

Pictures files Second possibility which completed the first one, save each frame of the video stream on pictures. To do this, we redirect again the input flow (udpsrc) with gstreamer but in this case in multiple files as shown this code : "! queue ! videorate ! jpegenc ! multifilesink location="frame/jpg/frame%06d.jpg"". Note that in this example we write frames on .jpg files but we can change the format to .bmp for example replacing "jpegenc" by "avenc_bmp". Thus the final string to put in gst_parse_launch become :

```
/* Build the pipeline */
//.jpg files
pipeline = gst_parse_launch("-v udpsrc port=5600 caps = \"application/x-rtp, media=(string)
    video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96\" ! rtph264depay
    ! decodebin ! tee name=t ! queue ! videorate ! jpegenc ! multifilesink location=\"frame/jpg/
    frame%06d.jpg\" t. ! queue ! videoconvert ! autovideosink t. ! queue ! avimux ! filesink location=
    capture.avi", NULL);
```

Memory buffer Last possibility which replace the second one, register each frames of the stream in a memory during the program execution. To do this, we redirect again the input flow (udpsrc) with gstreamer but in this case in a char table. To do this we have first to change the string to parse adding the following fileds : "! queue ! appsink name=sink emit-signals=true sync=false max-buffers=1

drop=true".
Then we have to get the sink before playing the pipeline. After that we do the following instructions :

1. Get the sample and the buffer
2. Get the capture, the structure, and so the width and the height
3. Get the map and the image matrix

The code of previous instructions is :

```
/* Build the pipeline */
pipeline = gst_parse_launch("-v udpsrc port=5600 caps = \"application/x-rtp, media=(string)
    video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96\" ! rtph264depay
     ! decodebin ! tee name=t ! queue ! videoconvert ! autovideosink t. ! queue ! avimux ! filesink
    location=capture.avi t. ! queue ! appsink name=sink emit-signals=true sync=false max-buffers
    =1 drop=true", NULL);

/* Get sink */
sink = gst_bin_get_by_name(GST_BIN(pipeline), "sink");

/* Get sample and buffer */
sample = gst_app_sink_pull_sample ((GstAppSink *)sink);
buffer = gst_sample_get_buffer(sample);

/* Get capture, structure, width and height */
caps = gst_sample_get_caps(sample);
structure = gst_caps_get_structure(caps, 0);
width = g_value_get_int(gst_structure_get_value(structure, "width"));
height = g_value_get_int(gst_structure_get_value(structure, "height"));
//printf("%d, %d\n", width, height); //Width and height of the video

/* Get map and image matrix */
gst_buffer_map(buffer, &map, GST_MAP_READ);
gst_image = malloc(map.size * sizeof(unsigned char));
memcpy(gst_image, (char*)map.data, map.size);
```

Sources

- Streaming live video from the 3DR Solo : https ://www.markturner.net/2017/02/06/streaming-live-video-from-the-3dr-solo/
- Install GStreamer on macOS : https ://gstreamer.freedesktop.org/documentation/installing/on-mac-osx.html#InstallingonMacOSX-Build
- Install GStreamer on linux : https ://gstreamer.freedesktop.org/documentation/installing/on-linux.html
- Install GStreamer on windows : https ://gstreamer.freedesktop.org/documentation/installing/on-windows.html
- GStreamer theory : https ://nicolargo.developpez.com/tutoriels/audio/gstreamer-theorie/
- GStreamer documentation : https ://gstreamer.freedesktop.org/documentation/index.html
- Get video stream with GStreamer in the command line : https ://gist.github.com/esrever10/7d39fe2d4163c5b
- Get video stream with GStreamer in a program c : https ://stackoverflow.com/questions/41698656/convert-gstreamer-command-to-c-code
- To register video stream in a file with gstreamer : https ://www.raspberrypi.org/forums/viewtopic.php ?t=65
- To register video stream frames in files with gstreamer : http ://www.tal.org/tutorials/timelapse-video-gstreamer