Recorder Development Manual

Cliff Swafford, Jonathan Hughes, Satyaraj Ryali

April 3, 2011

Recorder Program Flow

1. Configure the audio and video encoding

Using av_guess_format we configure the audio and video encoding for Mpeg4 video encoding and MP2 audio encoding, respectively.

2. Camera Initialization

Open up file descriptors to the webcam for video and audio recording.
Initialize formats for the data that we receive.

(a) Using OSS, we open a file descripter to /dev/dsp/ for audio recording. We sample at a rate of 44kHz with 16 bits per sample. Our audio encoding format is MP2.

(b) video record init Open the camera at /dev/video for communication. Set the format of the video frames received from the camera to be YUY2 encoded. Initialize the width and height of the frame to 640 by 480.

(c) Since the webcam doesn't enable read() access, we need to mmap the webcam frame buffers into our programs address space. Currently, we are using 10 buffer frames. Our buffer structure is shown as described below.

```
struct buffer {
        void * start;
        int length;
};
struct buffer * buffers;
```

(d) We then instruct the webcam to queue 10 buffer frames to be ready for retrieval.

(e) SDL Initialize

- In order to display the webcam's image to the screen, we use SDL to display the images we receive from the webcam buffer. The SDL window is initialized to match our image format paramters and create a YUYOverlay to present the frame.
- To allow for pan/tilt control, we listen for key presses in the SDL window. The up/down arrow keys are mapped for tilt functions and the left/right arrow keys are mapped for pan functions. Note: We have to wrap the pan/tilt ioctl calls in a while loop since they don't necessarily work the first time we try to control the camera.

3. Video and Audio encoding

At this point we split off separate threads for t video and audio encoding. We pull data from the camera and microphone, and encode the data.

4. Name server connection

We connect to the name server, tell it the information player clients will need to connect to our

recorder server, and then wait for a connection.

5. Video and Audio streaming

Once we receive a connection from a player client, we begin to start queuing up encoded video and audio. We split off 4 threads, one for video transmission, one for audio transmission, and one for sending control packets, and one for receiving control packets.

This is the struct for our packetqueue

```
typedef struct RecorderPacketQueue {
        AVPacketList *first_pkt, *last_pkt;
        int nb_packets;
        int size;
        pthread_mutex_t mutex;
} RecorderPacketQueue;
```

The video and audio transmission threads pull data from their respective queues and send them through the network. These can be a TCP or UDP connection, but only TCP is implemented.

The control packet transmission thread is unused, but could be utilized for bandwidth management or other things. This is a TCP connection.

The control packet receiving thread receives control packets. Here we get our remote control packets allowing the player to pan and tilt the camera using the arrow keys. This is a TCP connection.

2 Sources

We used example usage of ffmpeg and the v4l2 api to help drive our development.
The resources we used are linked here.

• V4l2 API: http://v4l2spec.bytesex.org/spec/book1.htm
• V4L2 Capture Example: http://v4l2spec.bytesex.org/spec/captureexample.
html
• Pan/Tilt Example: http://www.zerofsck.org/2009/03/09/example-codepan-
and-tilt-your-logitech-sphere-webcam-using-python-module-lpantilt-linuxv4l2/
• FFMPEG Example: http://recapstudio.googlecode.com/hg/src/common/AVEncoder.cpp?
r=4c707b0b25c2499b86aaeb3d096e16bd7dc441a6