- Video Filter Graphs Using Gstreamer -

Goal:
With this exercise we want to get some experiences with video decoding and playback based on the Gstreamer framework.

Exercise 10.1: Initial test
Verify on host and target, that Gstreamer can create a window on X11 used as video sink. This requires to start up the window environment using **`startx`**:

```
gst-launch-0.10 videotestsrc ! video/x-raw-rgb, \
     framerate=25/1, width=640, height=360 ! ximagesink
```

Exercise 10.2: Video playback with gst-launcher CLI
Get yourself some ogg files with video content (format: theora) from the net. Then you create on host and target an appropriate filter graph for video, e.g. using:

```
gst-launch-0.10 -v filesrc \
location="./big_buck_bunny_240p.ogv" ! oggdemux \
name="demux" ! queue ! theoradec ! ffmpegcolorspace ! \
videoscale ! autovideosink demux. ! queue ! vorbisdec ! \
audioconvert ! autoaudiosink
```

Exercise 10.3: Video playback as a stand-alone C program
After working on a stand-alone audio player in the previous exercise, now we want to extend the program to build a video player.

For this pls. execute the following steps: first you copy the audio player project into a new project within Eclipse. Now you extend the program by a simple video pipeline. Based on exercise 10.2 we now, that this pipeline should consist of connected Gstreamer filters **`ffmpegcolorspace ! videoscale ! autovideosink`** . Why? Because the **`uridecodebin`** filter will deliver in case of video content not only a raw-audio stream, but also a raw video stream *on another, dynamically created output pad*.

If you start your audio-only player with a ogg/theora file, e.g. **`big_buck_bunny_480p_stereo.ogg`** , then you will know, which data type this output pad has. Extend your **`pad-added-handler()`** in a way that it connects either the audio pipeline or the video pipeline depending on the type of **`new_pad`** . Additional information can be found on
http://docs.gstreamer.com/display/GstSDK/Basic+tutorial+3%3A+Dynamic+pipelines

Now create a run config and a remote run config in Eclipse to test your video player on the host and on the target.

Exercise 10.4: What else?

Now test the performance of your target: What is the maximum resolution which can be decoded and played back without interruptions?

Additional note:

Prerequisite for serious video decoding on the target based on GStreamer ist to install the hardware accelerated video decoders for OMX. In our standard image on the SD card we use with software-only decoders. The target is powerful enough to allow reasonable video decoding of standard formats in standard resolution, but NOT the flawless decoding of high definition video. See: https://github.com/matthiasbock/gstreamer-phone/wiki/Streaming-H.264-via-RTP