mipi-gstreamer-3

February 10, 2021

1 Using MIPI Sensors with GStreamer

1.1 Part 3

In this tutorial, we want to display a live video image. This could be done easily with an `Image`-widget. Let's create one here so we could use it later:

[ ]:
```
import ipywidgets
from IPython.display import display

image_widget = ipywidgets.Image(format="jpeg")
```

We need the GStreamer module, as usual:

[ ]:
```
import gi
gi.require_version("Gst", "1.0")
from gi.repository import Gst

Gst.init(())
```

Create a pipeline that scales the video stream to 640x480 using the accelerated scaler/converter *nvvidconv*. Then encode the result as a jpeg using the accelerated video encoder *nvjpegenc*.

[ ]:
```
pipeline = Gst.parse_launch("nvarguscamerasrc name=src ! " \
                            "nvvidconv ! " \
                            "video/x-raw(memory:NVMM),width=640,height=480 ! " \
                            "nvjpegenc ! "\
                            "appsink max-buffers=1 name=sink")
# Start the pipeline, then wait for it to run
pipeline.set_state(Gst.State.PLAYING)
pipeline.get_state(Gst.CLOCK_TIME_NONE)

# Get our src and sink elements to work with
src = pipeline.get_by_name("src")
sink = pipeline.get_by_name("sink")
```

Let's check whether we get a valid jpeg image, as expected:

[ ]:
```
sample = sink.emit("try-pull-sample", 1 * Gst.SECOND)
print("Caps:", sample.get_caps().to_string())
buffer = sample.get_buffer()
print("Got a buffer of size: %d Bytes" % (buffer.get_size()))
```

When we get jpeg images at the appsink, we could feed them to the image widget we created earlier for display.

To do this, we connect to the `new-sample` signal of the appsink. We also need to enable the elements signals via the `emit-signals` property. Finally, we have to pull a sample from the appsink since the appsinks buffer queue is already full right now and new samples can only arrive if there is room in the buffer queue.

[ ]:
```
# Signal handler called when a new sample arrives at the appsink
def on_new_sample(sink, image_widget):
    sample = sink.emit("pull-sample")
    buffer = sample.get_buffer()
    result, mapinfo = buffer.map(Gst.MapFlags.READ)
    if result:
        image_widget.value = mapinfo.data
        buffer.unmap(mapinfo)
    return Gst.FlowReturn.OK

sink.connect("new-sample", on_new_sample, image_widget)
# Signals are not emmited unless enabled
sink.set_property("emit-signals", True)
# Flush the buffer currently in the appsink, so that new buffers can arrive
sink.emit("pull-sample")

display(image_widget)
```

Finally, let's add some controls to change camera properties:

[ ]:
```
exp_slider = ipywidgets.FloatSlider(min=-2, max=2,␣
 ↪description="ExposureCompensation")
exp_slider.observe(lambda change: src.set_property("exposurecompensation",␣
 ↪change.new), "value")
display(exp_slider)
```

[ ]:
```
sat_slider = ipywidgets.FloatSlider(min=0, max=2, default=1,␣
 ↪description="Saturation")
sat_slider.observe(lambda change: src.set_property("saturation", change.new),␣
 ↪"value")
display(sat_slider)
```

Don't forget to shut down the pipeline when we are done:

[ ]:
```
pipeline.set_state(Gst.State.NULL)
```

[ ]: