Libcamera GSoC Proposal

Introduction

Name	Rishikesh Donadkar
IRC nick	Rishikesh
Email	rishikeshdonadkar@gamil.com
GitHub	https://github.com/RISHI27-dot
Time Zone	IST (UTC+5:30)
Time Commitments	No time commitments.
Blog	https://rishi27-dot.github.io/GSoC_2022

Academic/ Technical background -

1) I am a Second Year Bachelor of Technology student at Veermata Jijabai Technological Institute, Mumbai. I study Electronics and Telecommunication Engineering.

2) My first hands-on experience with programming was after the engineering college entrance exams when I decided to do a free course offered by Harvard University. In college I completed a course on Object Oriented Programming using C++.

3) Projects that I have done from the beginning of the first year are listed below.
    - ESP32-chat-communication-over-wifi -
      The aim of the project is to build a chat system using two esp32 development boards. The established communication is peer to peer

and happens without internet connection. The communication protocol used is ESP-NOW.

- Functional-Weeder -
  This project was made as a part of international Robotics competition E-Yantra conducted by IIT, Bombay.
  The aim of this project was to build a prototype model of robot that would help farmers in tasks like sowing and weeding. The Problem statement was to use two or more Alpha-Bots to perform sowing and weeding of two separate arenas (arena was an abstraction of a farming field). The path planning and path traversal algorithm are written in a functional programming language elixir. For efficient path planning and to cover the objective in least amount of time, the robots need to exchange data about each other's status. Communication was established between the bots using the Phoenix Framework**.** The code to complete the objective was run on raspberry pi where the robots communicated over the internet.
  We stood among the **top 10** in the competition.

4) I am pretty familiar with C and C++. I have implemented and practiced OOP with C++ and am well acquainted with basic OOP concepts.I have been using Linux (ubuntu 20.04) for 1.5 years and I am familiar with Linux and can do all basic tasks using CLI. During working on the projects mentioned above I used git extensively and am familiar with all basic git concepts. Talking about the patch submission process,I did not have much experience of it but got familiar with it when I submitted my first patch. The patch was trivial but I was happy for my first contribution in open source.

5) I am an active member of Society of Robotics and Automation, VJTI.Here we work on projects in Embedded Systems, Robotics, OpenCV, etc. I have contributed in giving various demos and lectures on topics like motor drivers, controlling robotic arm using emg sensors,Python,Git/GitHub, OpenCV etc.

Why did you apply to libcamera? What looked interesting/exciting?

Contributing in open source projects was always a dream for me. I love how open source projects provide excellent technologies to the consumer along with freedom to use, study, change, and distribute the source code.I wanted to apply for GSoC, while browsing through the organizations I came across libcamera. I liked how libcamera is making the application development easy by providing an open source camera stack that can overcome the complication caused by the device/hardware specific constraints. Also, studying an open source C++ code base and contributing to it would help me in developing C/C++ skills, communication skills and get me familiarized with how things work in the Open Source community which I would gladly like to be part of.

The Proposal

Project: Improve GStreamer element to add support for properties

GSoC WarmUP Tasks:

1) Built libcamera with the GStreamer element enabled and logged my observations here.

2) Streamed using GStreamer from the libcamera element. I did this task with the gst-launch-1.0 tool and logged my results in this blog.
   I was suggested by mentors to run the same gstreamer examples using code and not the gst-launch-1.0 CLI tool to get more acquainted with gstreamer, here is the code I have written.

3) Next, I built a test application that uses libcamera to learn more about libcamera controls. The code for the application can be found here. While writing this application I referred to simple-cam, which cleared the workflow of how libcamera bridges the gap between the application and the hardware camera device.

4) Read about the GStreamer properties from the GStreamer tutorials. First I read about GLib, GObject instantiation and GObject properties. Next, I went through the code in **libcamera/src/gstreamer** and understood how the camera-name property is defined in the **gst_libcamera_src_class_init()** function and how the **gst_libcamera_src_set_property()** and **gst_libcamera_src_get_property()** functions manage the property.

5) Now, I was clear about the gstreamer properties and libcamera controls. I asked for help on IRC on how to connect them to each other. To connect them we need to map the property form gstreamer to the appropriate control in libcamera. I understood how the mapping can be done by backtracking the function **gst_libcamera_stream_configuration_to_caps()**, for now the gstreamer **GstVideoFormat** is mapped to libcamera **PixelFormat**, this function can be further extended to map libcamera colorspace ot gstreamer colorimetry. Similarly, **gst_libcamera_stream_formats_to_caps()** could be extended to map the **framerate** form gstreamer and the **FrameDuration** (or **FrameDurationLimits**) from libcamera, and probably set it to be configured when the camera is started.

Goal

1) **Adding support for framerate property.**
   The aim will be that the framerate aspect of the stream can be controlled. The **framerate** property will be retrieved if set and will be mapped to the **FrameDurationLimits** control to be set when the camera is started. Also, appropriate FrameDurations needs to be calculated to report the framerate to gstreamer.

2) **Adding colorimetry support.**
   The aim will be to map the libcamera colorspace and the gstreamer colorimetry. After mapping the requested colorspace can be set when the camera is started. And also the applied colorspace can be reported to gstreamer.

3) **Adding support for properties that affect the configuration.**
   Some properties affect how the camera or stream is configured.
   The aim here will be to retrieve those properties and incorporate them into the configuration of camera or stream.

Plan

1) **Adding support for framerate property.**

- The property for framerate needs to be installed in **gstlibcamerasrc.cpp** so that the framerate set by gstreamer can be retrieved.

```
GParamSpec *spec_framerate = g_param_spec_int("framerate", "Framerate",
                             "specify the framerate.", FRAMERATE_MIN,
                             FRAMERATE_MAX,0,
                             (GParamFlags)(G_PARAM_READWRITE | G_PARAM_STATIC_STRINGS));
g_object_class_install_property(object_class, PROP_FRAMERATE, spec_framerate);
```

- The next step would be to add the **ControlList** attribute in the **StreamConfiguration** so that the FrameDurations controls can be accessed.
- Currently there is not any way in libcamera where the current framerate(or FrameDuration) can be determined. In this patch attempts are made to calculate and set the **FrameDurations** *(which was split into **FrameDuration** and **FrameDurationLimits** in this patch.)* control at the time of generation of the **CameraConfiguration** in SimplePipelineHandler, This might give misleading values of the frame duration according to the comment form **Jacopo Mondi**.Thus, when and where to calculate FrameDurations need to be determined.
- Next, **gst_libcamera_stream_configuration_to_caps()** function can be further extended to retrieve the **FrameDurations**, convert it to framerate and update the framerate data into gst caps.
- **gst_libcamera_stream_formats_to_caps()** can also further be extended to retrieve the framerate if set in the gstreamer pipeline, convert the framerate to **FrameDurationLimits** and set it into the **ControlList** when the camera is started.

2) Adding colorimetry support.

- The **ColorSpace** class of libcamera has the following attributes.
    1. primaries
    2. transferFunction
    3. ycbcrEncoding
    4. range

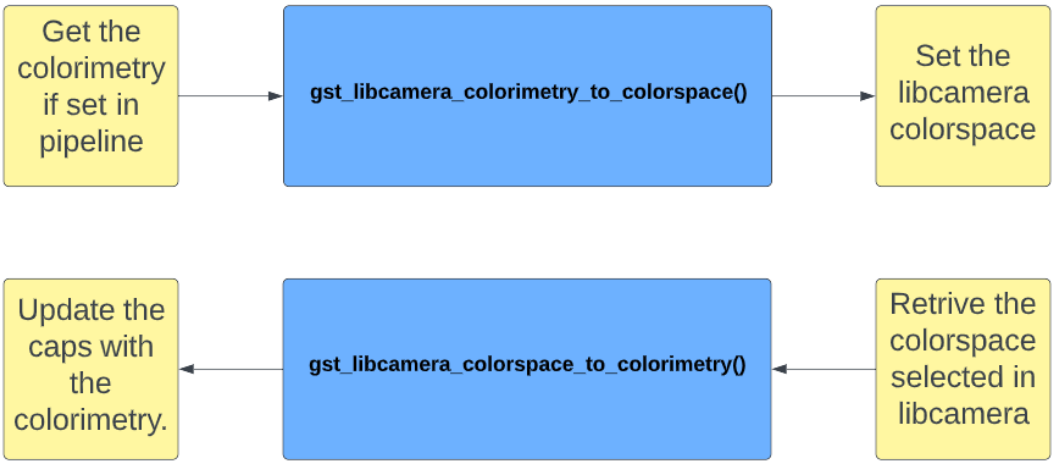
    Certain combinations of these fields forms a colorspace

- colorimetry property can be installed along with camera-name and framerate in gstlibcamerasrc.cpp
- The main task here will be to map the colorimetry from gstreamer(which can be found here) to the corresponding colorspace in libcamera.
- The mapping needs to be such that the closest requested colorspace can be determined and applied to the stream. Colorspace can be applied to the stream by setting it in the **colorSpace** attribute of the StreamConfiguration.
- **gst_libcamera_stream_configuration_to_caps()** function can be further extended to map the libcamera colorspace to gstreamer colorimetry.
- These functions need to be added to **gstlibcamera-utils.h** -
    - ➔ **gst_libcamera_colorspace_to_colorimetry()** ,
    - ➔ **gst_libcamera_colorimetry_to_colorspace()**

    gst_libcamera_colorspace_to_colorimetry() can be called inside **gst_libcamera_stream_configuration_to_caps()** to map the colorspace with colorimetry, update the caps exposed by the element with the current colorimetry.

    I suggest that **gst_libcamera_colorimetry_to_colorspace()** should retrieve the colorimetry if set, convert it to colorspace and apply that to stream configuration.

    .



3) Adding support for properties that affect the configuration.

- Apart from colorimetry and framerate, some controls also can be hooked up to gstreamer properties.
- Some of such properties that control the capture are Brightness, Exposure etc.
- If these properties are set by the gstreamer then they can be retrieved and the stream can be configured accordingly.

The Timeline

May 20	Results announced	Done with semester exams by May 20
May 20 - June 12	Community Bonding period	• Study the libcamera code base thoroughly. • Read more about gstreamer pipeline negotiation with libcamera. • Discuss implementation ideas for adding support to framerate and colorimetry.
June 13	Prerequisite completed	Begin with coding!
June 13 - 19	Milestone #1	Goal 1 • Declaring functions and gstreamer properties that need support. • Adding the necessary attributes in pre-existing classes to hook up the controls and properties.
June 20 - 26	Milestone #2	Goal 1 • Figuring out when and where the current frameduration can be calculated. • Writing code to calculate frameduration and setting the control.
June 27 - July 3	Milestone #3	Goal 1 • Retrieving the frameduration which was set, converting it to framerate and updating it into gst caps. • Extending the **gst_libcamera_stream_formats_to_caps()** function to get the framerate gstreamer property if set and converting it to

		frameduration and setting the control to run the stream at requested framerate.
July 4 - 10	Milestone #4	Goal 1 ● Making sure that the stream runs with the framerate requested and pipeline negotiation is good. ● Writing tests to check if framerate is set successfully.
July 11 - 17	Milestone#5	Goal 2 ● Declaring the functions required to map gstreamer colorimetry and libcamera colorspace ● Decide how the mapping needs to be done considering that platforms will typically have different constraints on what color spaces can be supported and in what combinations.
July 18 - 21	Milestone#6	Goal 2 ● Map the libcamera colorspace with gstreamer colorimetry, mapping must be such that the closest colorspace corresponding to the gstreamer colorimetry can be selected and the stream can be configured accordingly. ● Update the colorimetry in caps exposed by the libcamera element.
July 25 - 29	Phase 1 Evaluation	Sum up all the work done till now in the blog and submit it for evaluation.
July 30 - Aug 7	Milestone#7	Goal 2 ● Testing to check if the mapping done is acceptable and working.

		Goal 3 ● Declare the properties like Brightness and Exposure and map them to corresponding controls from libcamera.
Aug 8 - 14	Milestone#8	Goal 3 ● Pass the mapped controls either on camera starting or queuing it as a part of request so that the requested controls can be seen in the stream.
Aug 15 - 21	Final Week	● Summarize the work in the blog. ● Document the code written.
Aug 22 - 28	End of session	● Mentors submit final GSoC contributor evaluations. (standard coding period)