GStreamer Buster Version 1.14.4 on RaspberryPi 4

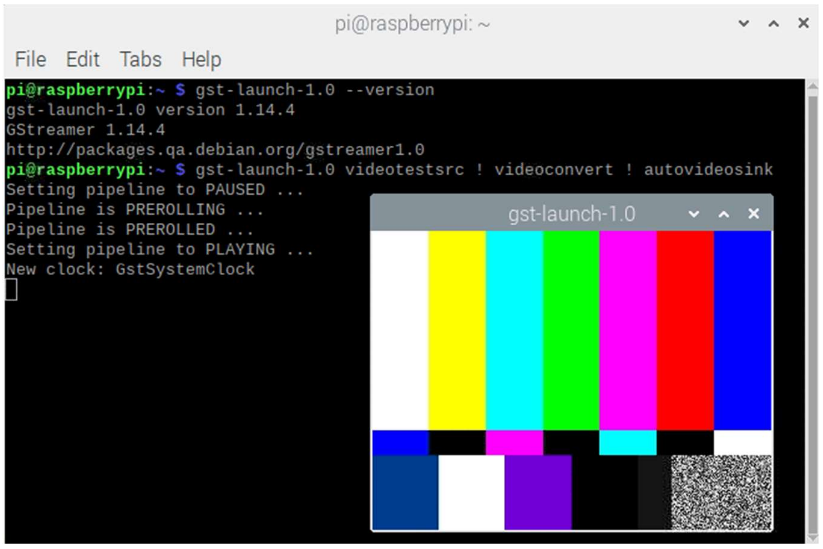
Installation :

A few of the packages required for the gstreamer are already installed raspberry pi, we need to install some additional plugins

```
# install a missing dependency
$ sudo apt-get install libx264-dev libjpeg-dev
# install the remaining plugins
$ sudo apt-get install libgstreamer1.0-dev \
     libgstreamer-plugins-base1.0-dev \
     libgstreamer-plugins-bad1.0-dev \
     gstreamer1.0-plugins-ugly \
     gstreamer1.0-tools
# install some optional plugins
$ sudo apt-get install gstreamer1.0-gl gstreamer1.0-gtk3
# if you have Qt5 install this plugin
$ sudo apt-get install gstreamer1.0-qt5
# install if you want to work with audio
$ sudo apt-get install gstreamer1.0-pulseaudio
```

Testing Installation :

```
$ gst-launch-1.0 videotestsrc ! videoconvert ! autovideosink
```

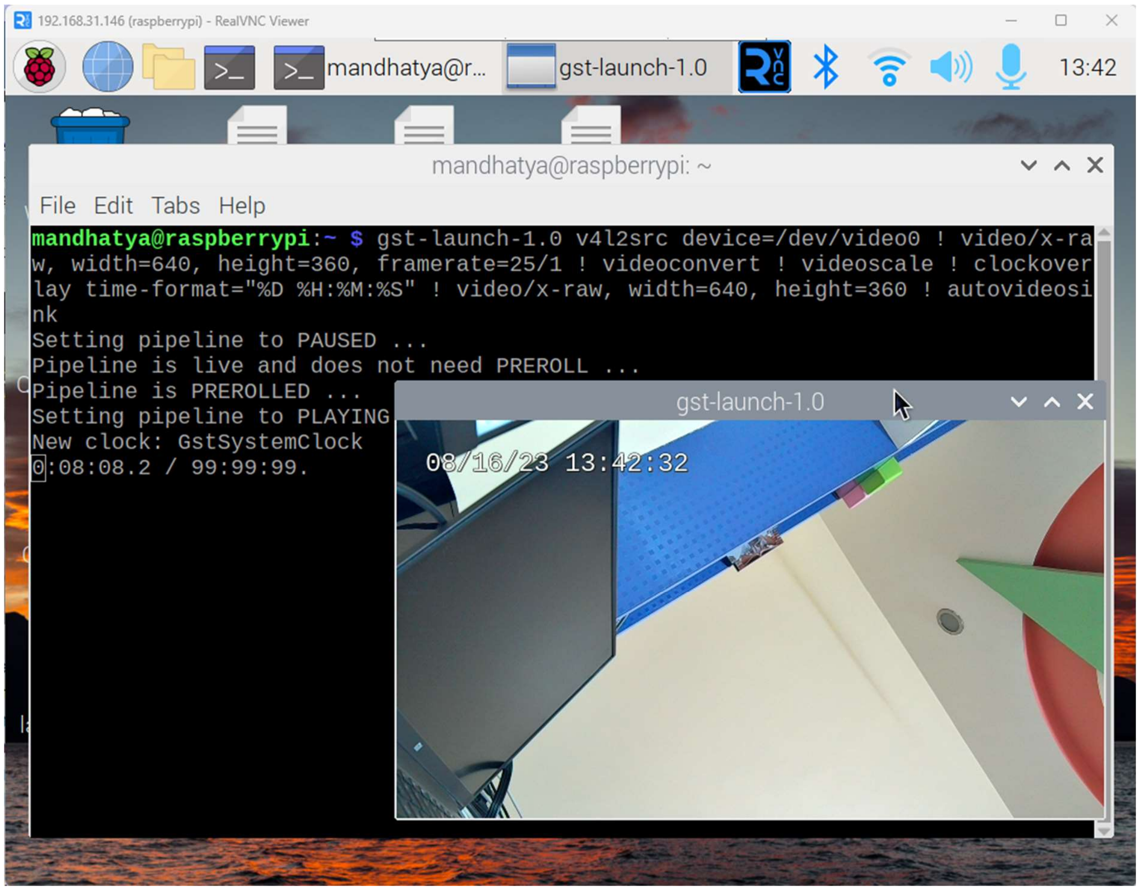


```
pi@raspberrypi:~ $ gst-launch-1.0 --version
gst-launch-1.0 version 1.14.4
GStreamer 1.14.4
http://packages.qa.debian.org/gstreamer1.0
pi@raspberrypi:~ $ gst-launch-1.0 videotestsrc ! videoconvert ! autovideosink
Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

Testing Streaming with Camera :

```
$ gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-raw, width=1280,
height=720, framerate=30/1 ! videoconvert ! videoscale ! clockoverlay
time-format="%D %H:%M:%S" ! video/x-raw, width=640, height=360
! autovideosink
```

(Follow the instructions below the picture before executing the above command to use the correct configuration available for your camera)



```
mandhatya@raspberrypi:~ $ gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-ra
w, width=640, height=360, framerate=25/1 ! videoconvert ! videoscale ! clockover
lay time-format="%D %H:%M:%S" ! video/x-raw, width=640, height=360 ! autovideosi
nk
Setting pipeline to PAUSED ...
Pipeline is live and does not need PREROLL ...
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
0:08:08.2 / 99:99:99.
```

```
08/16/23 13:42:32
```

V4l2src device=/dev/video0 is used to capture the video from the camera

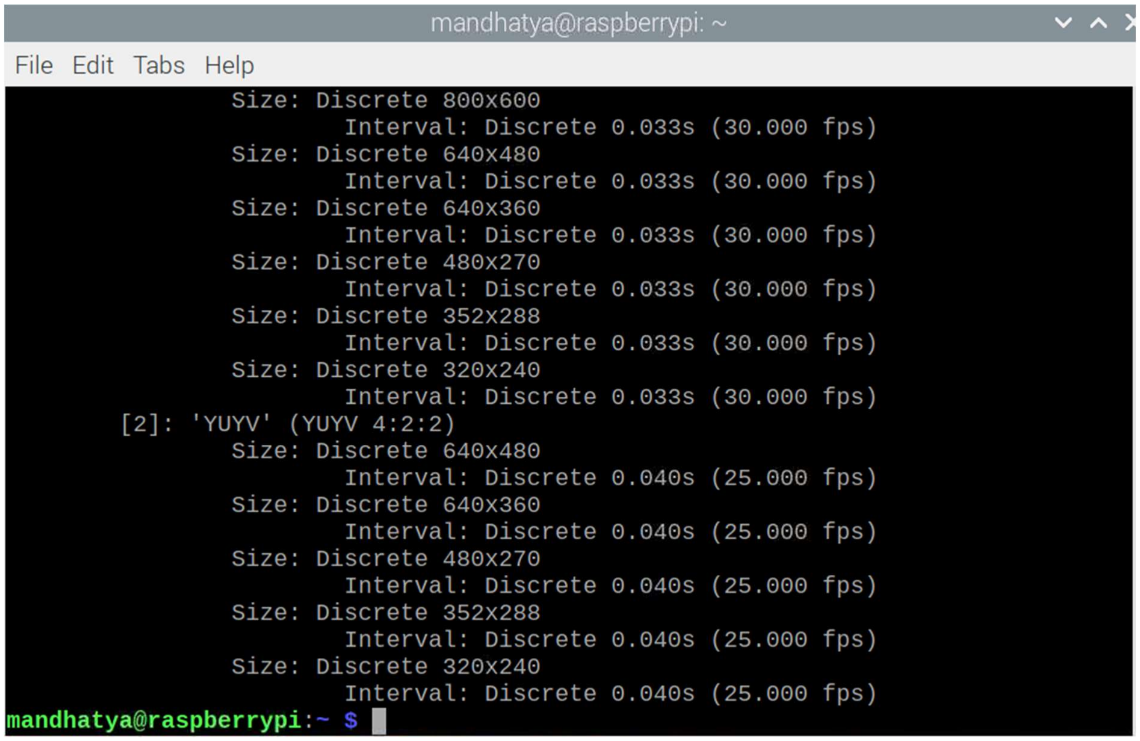
- If you are using raspi camera(cli) then it is video0
- If there is no raspi on the raspberry pi and you are using a usb camera, even then it’s video0

If there is raspi camera and there is also a usb camera connected, if you want to stream through raspi its video0, if usb camera its video1.

The width and height are your resolution, the width, height and your framerate are dependent on your camera, to check your camera specifications type :

```
$ v4l2-ctl --device=/dev/video0 -D --list-formats-ext
```

Use only YUVY Specification



```
                Size: Discrete 800x600
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 640x480
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 640x360
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 480x270
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 352x288
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 320x240
                        Interval: Discrete 0.033s (30.000 fps)
        [2]: 'YUYV' (YUYV 4:2:2)
                Size: Discrete 640x480
                        Interval: Discrete 0.040s (25.000 fps)
                Size: Discrete 640x360
                        Interval: Discrete 0.040s (25.000 fps)
                Size: Discrete 480x270
                        Interval: Discrete 0.040s (25.000 fps)
                Size: Discrete 352x288
                        Interval: Discrete 0.040s (25.000 fps)
                Size: Discrete 320x240
                        Interval: Discrete 0.040s (25.000 fps)
mandhatya@raspberrypi:~ $
```

UDP Streaming:

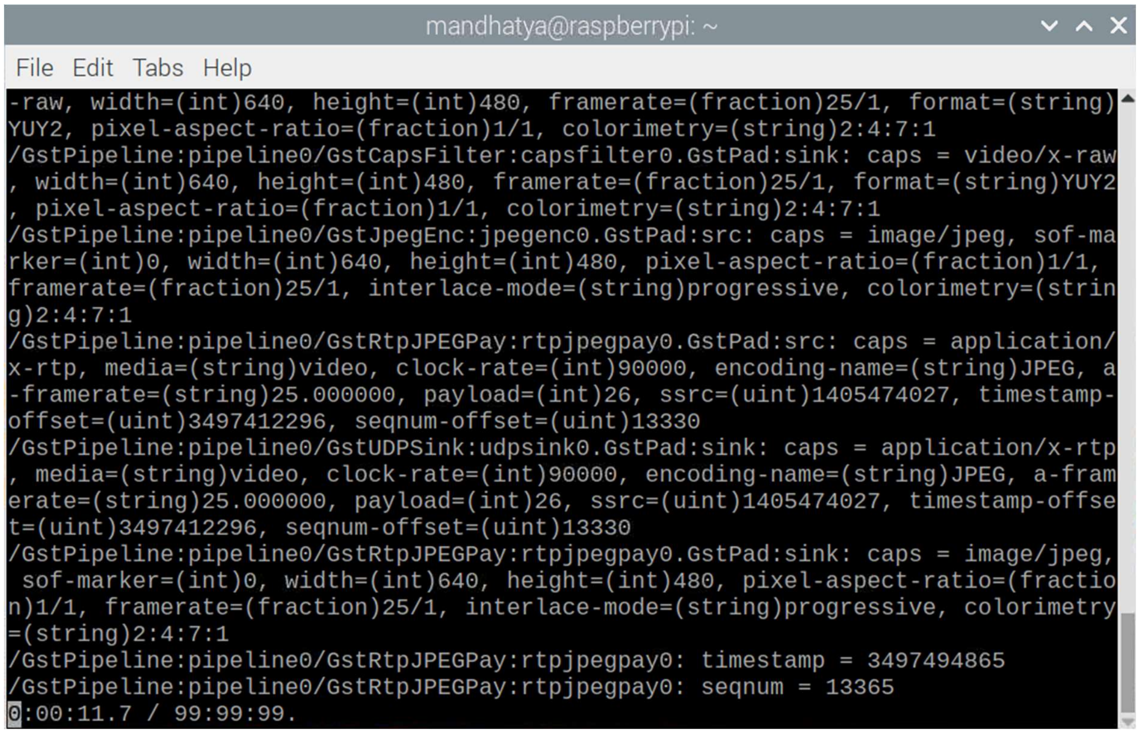
We use two Raspberry Pis, both connected to the same home network. However, it could just as easily be an RPi and a laptop on the other side of the world. You need to know the address of the receiving Raspberry Pi on forehand.

Raspberry Pi 32 or 64-bit OS

```
# get the IP address of the recieving RPi first
$ hostname -I
# start the sender, the one with the camera
$ gst-launch-1.0 -v v4l2src device=/dev/video0 num-buffers=-1 ! video/x-raw, width=640, height=480, framerate=30/1 ! videoconvert ! jpegenc ! rtpjpegpay ! udpsink host=192.168.31.146 port=5200
# start the reciever, the one with IP 192.168.31.146
```

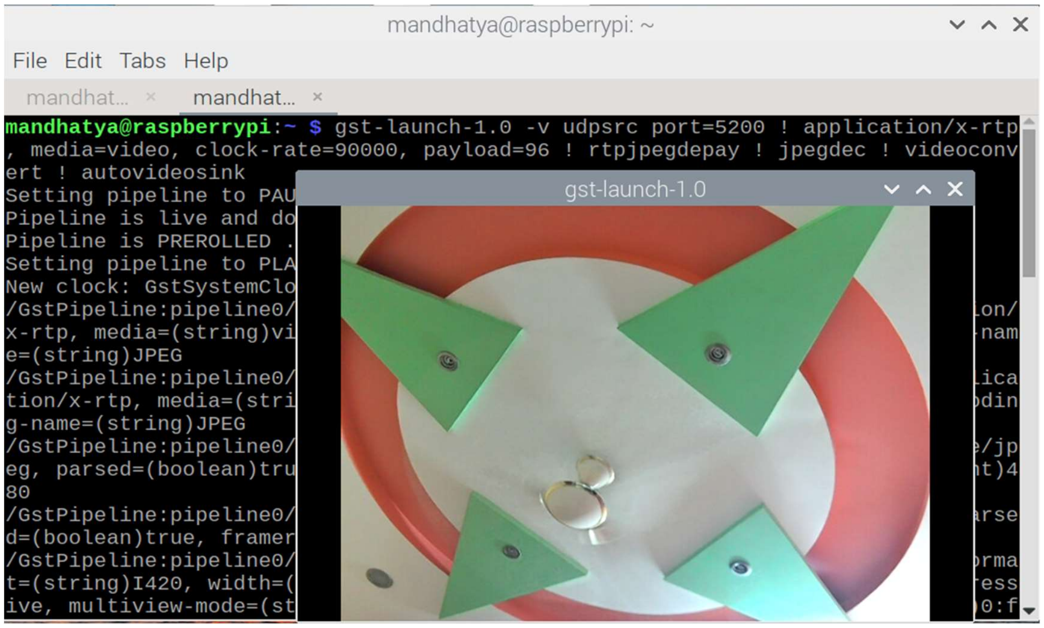
```
$ gst-launch-1.0 -v udpsrc port=5200 ! application/x-rtp, media=video,
clock-rate=90000, payload=96 ! rtpjpegdepay ! jpegdec ! videoconvert !
autovideosink
```

Sender



```
-raw, width=(int)640, height=(int)480, framerate=(fraction)25/1, format=(string)
YUY2, pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)2:4:7:1
/GstPipeline:pipeline0/GstCapsFilter:capsfilter0.GstPad:sink: caps = video/x-raw
, width=(int)640, height=(int)480, framerate=(fraction)25/1, format=(string)YUY2
, pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)2:4:7:1
/GstPipeline:pipeline0/GstJpegEnc:jpegenc0.GstPad:src: caps = image/jpeg, sof-ma
rker=(int)0, width=(int)640, height=(int)480, pixel-aspect-ratio=(fraction)1/1,
framerate=(fraction)25/1, interlace-mode=(string)progressive, colorimetry=(strin
g)2:4:7:1
/GstPipeline:pipeline0/GstRtpJPEGPay:rtpjpegpay0.GstPad:src: caps = application/
x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)JPEG, a
-framerate=(string)25.000000, payload=(int)26, ssrc=(uint)1405474027, timestamp-
offset=(uint)3497412296, seqnum-offset=(uint)13330
/GstPipeline:pipeline0/GstUDPSink:udpsink0.GstPad:sink: caps = application/x-rtp
, media=(string)video, clock-rate=(int)90000, encoding-name=(string)JPEG, a-fram
erate=(string)25.000000, payload=(int)26, ssrc=(uint)1405474027, timestamp-offse
t=(uint)3497412296, seqnum-offset=(uint)13330
/GstPipeline:pipeline0/GstRtpJPEGPay:rtpjpegpay0.GstPad:sink: caps = image/jpeg,
 sof-marker=(int)0, width=(int)640, height=(int)480, pixel-aspect-ratio=(fractio
n)1/1, framerate=(fraction)25/1, interlace-mode=(string)progressive, colorimetry
=(string)2:4:7:1
/GstPipeline:pipeline0/GstRtpJPEGPay:rtpjpegpay0: timestamp = 3497494865
/GstPipeline:pipeline0/GstRtpJPEGPay:rtpjpegpay0: seqnum = 13365
0:00:11.7 / 99:99:99.
```

Receiver



```
mandhatya@raspberrypi:~ $ gst-launch-1.0 -v udpsrc port=5200 ! application/x-rtp
, media=video, clock-rate=90000, payload=96 ! rtpjpegdepay ! jpegdec ! videoconv
ert ! autovideosink
Setting pipeline to PAU
Pipeline is live and do
Pipeline is PREROLLED .
Setting pipeline to PLA
New clock: GstSystemClo
/GstPipeline:pipeline0/                                                    ion/
x-rtp, media=(string)vi                                                    -nam
e=(string)JPEG
/GstPipeline:pipeline0/                                                    lica
tion/x-rtp, media=(stri                                                    odin
g-name=(string)JPEG
/GstPipeline:pipeline0/                                                    e/jp
eg, parsed=(boolean)tru                                                    nt)4
80
/GstPipeline:pipeline0/                                                    arse
d=(boolean)true, framer
/GstPipeline:pipeline0/                                                    orma
t=(string)I420, width=(                                                    ress
ive, multiview-mode=(st                                                    0:f
```

TCP Streaming:

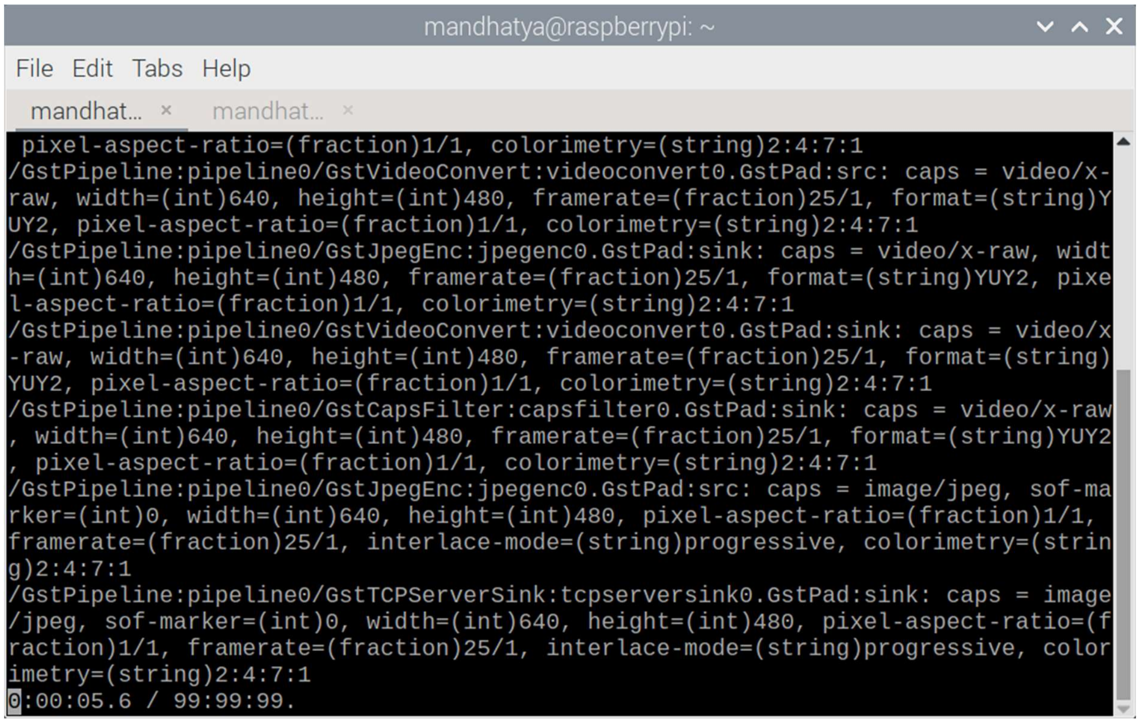
The other method of streaming is with TCP. The difference with UDP is the latency. UDP is faster.

The commands as listed below. Note the different IP addresses. With TCP streaming, you use the server address, the sender, instead of the receiver, as we saw with the UDP streaming.

Raspberry Pi 32 or 64-bit OS

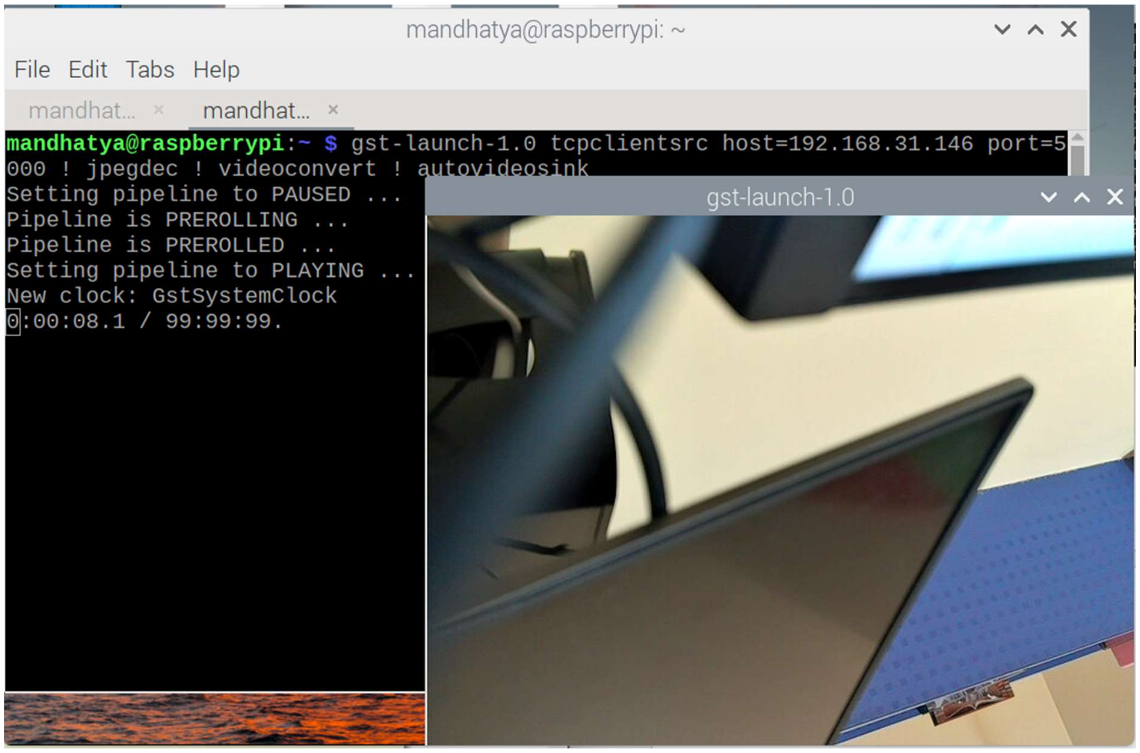
```
# get the IP address of the sending RPi first
$ hostname -I
# start the sender, the one with the Raspicam and IP 192.168.178.32
$ gst-launch-1.0 -v v4l2src device=/dev/video0 num-buffers=-1 ! video/x-raw,width=640,height=480, framerate=30/1 ! videoconvert ! jpegenc ! tcpserversink  host=192.168.31.146 port=5000
# start the reciever and connect to the server with IP 192.168.178.32
$ gst-launch-1.0 tcpclientsrc host=192.168.31.146 port=5000 ! jpegdec ! videoconvert ! autovideosink
```

Sender



```
 pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)2:4:7:1
/GstPipeline:pipeline0/GstVideoConvert:videoconvert0.GstPad:src: caps = video/x-
raw, width=(int)640, height=(int)480, framerate=(fraction)25/1, format=(string)Y
UY2, pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)2:4:7:1
/GstPipeline:pipeline0/GstJpegEnc:jpegenc0.GstPad:sink: caps = video/x-raw, widt
h=(int)640, height=(int)480, framerate=(fraction)25/1, format=(string)YUY2, pixe
l-aspect-ratio=(fraction)1/1, colorimetry=(string)2:4:7:1
/GstPipeline:pipeline0/GstVideoConvert:videoconvert0.GstPad:sink: caps = video/x
-raw, width=(int)640, height=(int)480, framerate=(fraction)25/1, format=(string)
YUY2, pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)2:4:7:1
/GstPipeline:pipeline0/GstCapsFilter:capsfilter0.GstPad:sink: caps = video/x-raw
, width=(int)640, height=(int)480, framerate=(fraction)25/1, format=(string)YUY2
, pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)2:4:7:1
/GstPipeline:pipeline0/GstJpegEnc:jpegenc0.GstPad:src: caps = image/jpeg, sof-ma
rker=(int)0, width=(int)640, height=(int)480, pixel-aspect-ratio=(fraction)1/1,
framerate=(fraction)25/1, interlace-mode=(string)progressive, colorimetry=(strin
g)2:4:7:1
/GstPipeline:pipeline0/GstTCPServerSink:tcpserversink0.GstPad:sink: caps = image
/jpeg, sof-marker=(int)0, width=(int)640, height=(int)480, pixel-aspect-ratio=(f
raction)1/1, framerate=(fraction)25/1, interlace-mode=(string)progressive, color
imetry=(string)2:4:7:1
0:00:05.6 / 99:99:99.
```

Receiver



```
mandhatya@raspberrypi:~ $ gst-launch-1.0 tcpclientsrc host=192.168.31.146 port=5
000 ! jpegdec ! videoconvert ! autovideosink
Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
0:00:08.1 / 99:99:99.
```

Reference Link :

https://qengineering.eu/install-gstreamer-1.18-on-raspberry-pi-4.html

Raspberry pi to AWS Kinesis Video Streams

About Kinesis Video Streams-
ppt link :- https://docs.google.com/presentation/d/1gP1cU0VE-0EHW7_Ou58qu5EtRfE1i1up/edit?usp=sharing&ouid=112744654845453628982&rtpof=true&sd=true

Creating Kinesis video stream in aws :-

- Go to aws management console
- Go to kinesis video streams
- Go to video streams
- Set region before proceeding
- Name and create video stream
- Go to IAM User
- Create IAM User
- Create access key and secret access key id and save them as csv file

Github link(for installation) :- https://github.com/awslabs/amazon-kinesis-video-streams-producer-sdk-cpp

Some steps to be done before following github link(not included in github) :-

In ubuntu or raspberry pi terminal :-

```
sudo apt update && sudo apt upgrade
sudo apt install pkg-config m4 cmake autconf g++ default-jdk
```

Some steps to be modified between the installation that were given incorrect :-
Instead of directly adding GST_PLUGIN_PATH and other paths in the normal terminal follow the following steps

- sudo su
- sudo nano ~/.bashsrc
- Go to end of the the line and add the following
- export GST_PLUGIN_PATH=home_directory_of_amazon_kinesis/build
- export LD_LIBRARY_PATH=home_directory_of_amazon_kinesis/open-source/local/lib
- execute the gst commands in the root directory, i.e, in sudo su
- Use the height width and framerate supported by the USB or the CLI camera, you can check the compatibility in the gstreamer on raspberry pi doc

Working Video : -
https://drive.google.com/file/d/18n7no6bw5tTKEVXFqp2OtL33swLzCwW7/view?usp=sharing

OCR in Raspberry Pi

Steps : -

- sudo nano ocr_camera.py
- sudo apt install pytesseract opencv-python
- Code :-

```
import cv2
import pytesseract
from pytesseract import Output

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

while True:
   # Capture frame-by-frame
   ret, frame = cap.read()

   d = pytesseract.image_to_data(frame, output_type=Output.DICT)
   n_boxes = len(d['text'])
   for i in range(n_boxes):
      if int(d['conf'][i]) > 60:
         (text, x, y, w, h) = (d['text'][i], d['left'][i], d['top'][i], d['width'][i],
d['height'][i])
         # don't show empty text
         if text and text.strip() != "":
            frame = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            frame = cv2.putText(frame, text, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 255), 3)

   # Display the resulting frame
   cv2.imshow('frame', frame)
   if cv2.waitKey(1) & 0xFF == ord('q'):
      break

# When everything done, release the capture
cap.release()
```

```
cv2.destroyAllWindows()
```

- python3 ocr_camera.py