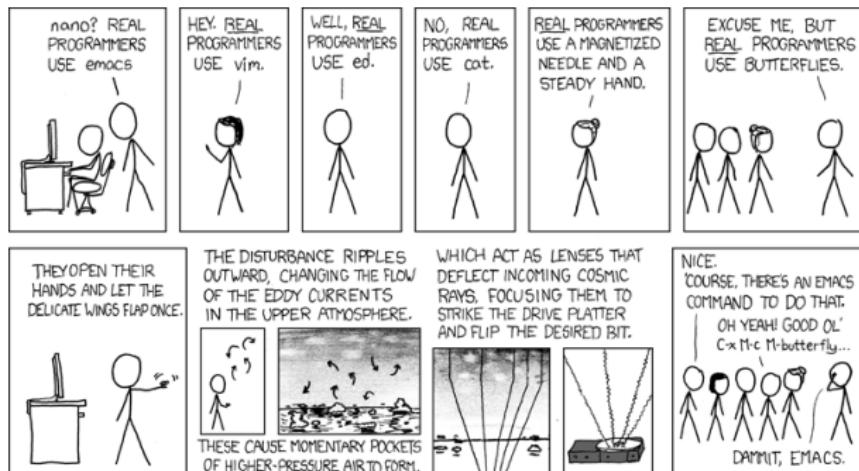




Gstreamer Pipeline für Kamera zu DNN in Python

Ingo Clever





- ① Specs und Implementation des Projekt
- ② Gstreamer ⇒ Applikation
- ③ Bsp. OpenCV ⇒ Tensorflow
- ④ dynamische Pipelines mit GstInterpipe & GstInference
- ⑤ Wie geht's weiter? (Quellen, Links, Fragen)



Der Xavier AGX eignet sich als mobile Lösung im Batteriebetrieb

NVIDIA Jetson AGX Xavier Module	
CPU	8-core NVIDIA Carmel 64-bit ARMv8.2 @ 2265MHz
GPU	512-core NVIDIA Volta @ 1377MHz with 64 TensorCores
DL	Dual NVIDIA Deep Learning Accelerators [DLAs]
Memory	16GB 256-bit LPDDR4x @ 2133MHz 137GB/s
Storage	32GB eMMC 5.1
Vision	[2x] 7-way VLIW Vision Accelerator
Encoder*	[4x] 4Kp60 [8x] 4Kp30 [16x] 1080p60 [32x] 1080p30 Maximum throughput up to [2x] 1000MP/s – H.265 Main
Decoder*	[2x] 8Kp30 [6x] 4Kp60 [12x] 4Kp30 [26x] 1080p60 [52x] 1080p30 Maximum throughput up to [2x] 1500MP/s – H.265 Main
Cameras†	[16x] MIPI CSI-2 lanes, [8x] SLVS-EC lanes; up to 6 active sensor streams and 36 virtual channels
Display	[3x] eDP 1.4 / DP 1.2 / HDMI 2.0 @ 4Kp60
Ethernet	10/100/1000 BASE-T Ethernet + MAC + RGMII interface
USB	[3x] USB 3.1 + [4x] USB 2.0
PCIe††	[5x] PCIe Gen 4 controllers 1x8, 1x4, 1x2, 2x1
CAN	Dual CAN bus controller
Misc I/Os	UART, SPI, I2C, I2S, GPIOs
Socket	699-pin board-to-board connector, 100x87mm with 16mm Z-height
Thermals‡	-25°C to 80°C
Power	10W / 15W / 30W profiles, 9.0V-20VDC input





- Raspberry Pi 2
Eine große Anzahl preiswerter und kompatibler Sensoren kann erstanden werden.
- MIPI CSI Raspberry Camera V2 "Noir"
Das Camera Serial Interface ermöglicht es die Kamera direkt anzusteuern.





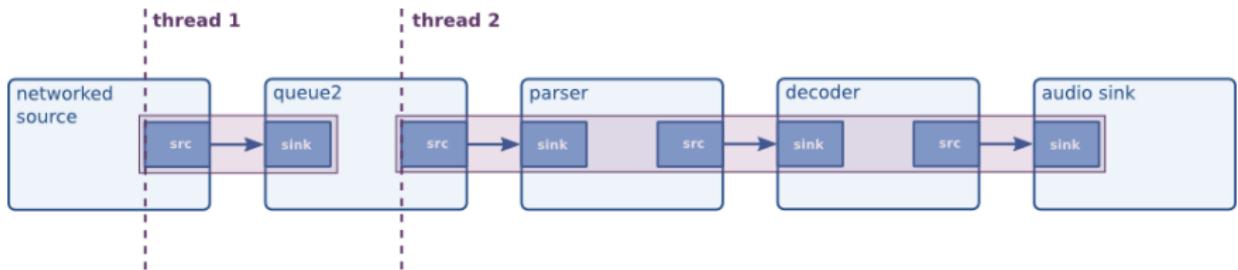
- ① In den Raspberry Pi Einstellungen den ssh Server, CSI Kameraport und X11-Forwarding aktivieren.
- ② In eigenem Subnetz eindeutige IP4 Adressen für beide Geräte festlegen.
- ③ Ad-Hoc Netzwerk bzw. Direktverbindung (ohne DHCP) auf beiden Geräten einrichten.
- ④ in Remote-Terminal einloggen mit "ssh -X pi@<chosen_ip4_address>"



- GStreamer ist ein großes Open-Source Framework, für die Bearbeitung und das Erstellen von streaming media applications.
- Eine Gstreamer Pipeline wird modular aufgebaut
- Plugins teilen sich auf in die "good", "bad" und die "ugly".
- Es gibt viele spezielle Plugins, mit Unterstützung von Python, C, C++ und Rust.



- Prinzipieller Aufbau einer Gstreamer Pipeline
- Auf dem Diagramm sieht man kein Muxer/Demuxer Modul





Als Quelle des Projekts werden IP Kameras aufgesetzt. Hierfür nutzen wir das Modul "udpsrc" aus den rtsp Modulen.

Das Real Time Streaming Protocol ist Standard für Audiovisuelle IP-Streams.

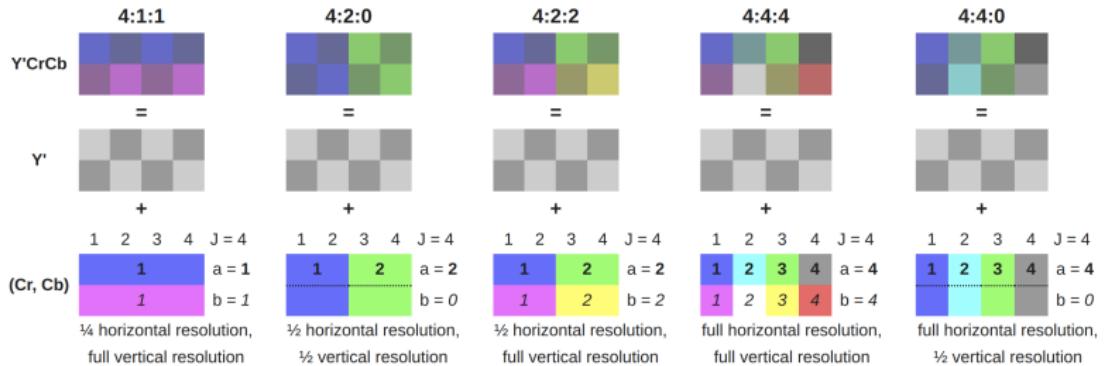
Um Pufferüberläufe bei aufsetzen mehrerer Kameras zu beheben wird der Empfangspuffer vergrößert.



- Die unterstützten Formate der angeschlossenen Kamera zeigt
`v4l2-ctl -list-formats-ext`
- um mehrere Streams über einen Socket zu senden,
`sysctl net.core.rmem_max`
`sysctl net.core.rmem_default`
- UDP-Buffer von 208 KB im Xavier AGX auf 8MB setzen.
`sudo sysctl -w net.core.rmem_max=8388608`
`sudo sysctl -w net.core.wmem_max=8388608`



- Farbverläufe werden ungenauer wahrgenommen als Luminanzkontraste.
- Daher wird nach Umwandlung in den YUV-Farbraum die Auflösung der Farbkanäle verringert.



- Die meisten Kameras (zumindest USB-Kameras) nutzen wegen der geringeren Bilddatengröße den YUV Farbraum



Raspberry Pi

```
gst-launch-1.0 -v v4l2src           # Video4Linux2 Kameratreiber  
device=/dev/video0 !               # Ort der angeschlossenen Kamera  
'video/x-raw, width=128, height=96,' # RAW-Video  
'format=I420, framerate=10/1'!      # Interlaced YUV 4:2:0  
rtpvrawpay !                      # rtp-video-raw-payload-converter  
udpsink host=10.0.0.1 port=5000    # udp Node mit Adresse und Port  
sync=false async=false             # Verwerfen von Bildern bei OF
```



Raspberry Pi

```
gst-launch-1.0 -v v4l2src device=/dev/video0 !
'video/x-raw, width=128, height=96,
/format=I420, framerate=10/1'
rtpvrawpay !
udpsink host=10.0.0.1 port=5000 sync=false async=false
```

Xavier AGX

```
gst-launch-1.0 udpsrc port=5000 !
'application/x-rtp, media=(string)video,
encoding-name=(string)RAW, sampling=YCbCr-4:2:0,
width=(string)128, height=(string)96' !
rtpvrawdepay ! xvimagesink
```



Vollständige Informationen zu jedem Modul des Stream werden durch Nutzung des 'verbose' Flag -ev in der Shell angezeigt.

Für Synchronisation muss z.B. der Offset des Timestamp genutzt werden.

```
pi@pi: ~
Leitung wird auf ABSPIELEN gesetzt ...
New clock: GstSystemClock
/GstPipeline:pipeline0/gstV4l2Src:v4l2src0.GstPad:src: caps = video/x-raw, width=(int)320, height=(int)240, framerate=(fraction)10/1, format=(string)UYVY, colorimetry=(string)bt601, interlace-mode=(string)progressive
/GstPipeline:pipeline0/gstCapsFilter:capsfilter0.GstPad:src: caps = video/x-raw, width=(int)320, height=(int)240, framerate=(fraction)10/1, format=(string)UYVY, colorimetry=(string)bt601, interlace-mode=(string)progressive
/GstPipeline:pipeline0/gstRtpRawPay0.RtpPad:src: caps = application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)RAW, sampling=(string)YCbCr-4:2:2, depth=(string)8, width=(string)320, height=(string)240, colorimetry=(string)BT601-5, payload=(int)96, ssrc=(uint)1736874360, timestamp-offset=(uint)2456897657, seqnum-offset=(uint)14497, a-framerate=(string)10
/GstPipeline:pipeline0/gstRtpRawPay0.RtpPad:sink: caps = application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)RAW, sampling=(string)YCbCr-4:2:2, depth=(string)8, width=(string)320, height=(string)240, colorimetry=(string)BT601-5, payload=(int)96, ssrc=(uint)1736874360, timestamp-offset=(uint)2456897657, seqnum-offset=(uint)14497, a-framerate=(string)10
/GstPipeline:pipeline0/gstRtpRawPay0.RtpPad:sink: caps = video/x-raw, width=(int)320, height=(int)240, framerate=(fraction)10/1, format=(string)UYVY, colorimetry=(string)bt601, interlace-mode=(string)progressive
/GstPipeline:pipeline0/gstCapsFilter:capsfilter0.GstPad:sink: caps = video/x-raw, width=(int)320, height=(int)240, framerate=(fraction)10/1, format=(string)UYVY, colorimetry=(string)bt601, interlace-mode=(string)progressive
/GstPipeline:pipeline0/gstRtpRawPay0.RtpPad:sink: timestamp = 2456974610
/GstPipeline:pipeline0/gstRtpRawPay0.RtpPad:sink: seqnum = 14497
```



Die Objekt-Hierarchie für den Programmcode erfährt man mit `gst-inspect`

`gst-inspect-1.0`

```
GObject
\; => GstElement
\; \; => GstBaseSink
\; \; \; => GstVideoSink
\; \; \; \; => xvimagesink
```



```
class PythonPipeline(object):
    def __init__(self):
        self.pipeline = Gst.Pipeline()
        self.filesrc = Gst.ElementFactory.make('filesrc')
        self.oggdemux = Gst.ElementFactory.make('oggdemux')

        self.pipeline.add(self.filesrc)
        self.pipeline.add(self.oggdemux)
        self.pipeline.add(self.theoradec)
        self.pipeline.add(self.videoconvert)
        self.pipeline.add(self.xvimagesink)

        self.filesrc.link(self.oggdemux)
        self.oggdemux.link(self.theoradec)
        self.theoradec.link(self.videoconvert)
        self.videoconvert.link(self.xvimagesink)
```



```
class SimplePipeline(object):
    def __init__(self):
        pipe_desc = ('filesrc name=src !'
                    'oggdemux ! theoraenc !'
                    'videoconvert ! xvimagesink')
        self.pipeline = Gst.parse_launch(pipe_desc)
        self.filesrc = self.pipeline.get_by_name('src')
```



Verschiedene Möglichkeiten z.B.:

- ① als Bildserie
mit gst-Modul 'filesink' (Ressourcensparend)
- ② via OpenCV
mit gst-modul 'appsink' (Achtung, Duplikat auf der Festplatte)
- ③ mit Nvidias Deepstream[©]
(umfangreiche proprietäre Lösung)
- ④ NNStreamer
(Open Source, unterstützt Tizen OS)
- ⑤ RidgeRun r2inference & gst-inference
(Open Source, unterstützt viele DNN Formate)



- Die Open Source Computer Vision Library ist in V4 nativ C++, jedoch voll durch Python, Matlab und Java anzusteuern.
- Sie beherbergt über 2500 Algorithmen zur Bilderkennung von klassisch bis cutting edge.
- Overlays zur Hervorhebung von Bildinhalten gehören ebenso zu den Grundfunktionen



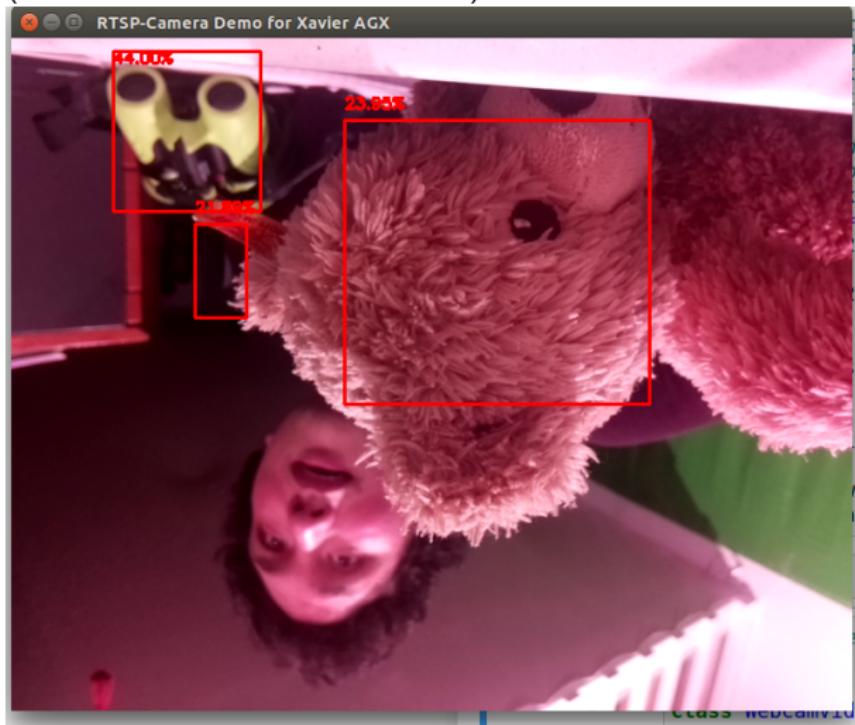
OpenCV bietet verschiedene Möglichkeiten zur Einbindung von neuralen Netzen an
mögliche Layer mit OpenCV (für Caffe und Tensorflow) sind

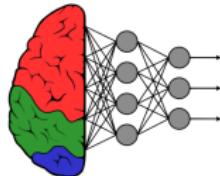
- Convolution
- Deconvolution
- Pooling
- InnerProduct
- TanH, ReLU, Sigmoid, BNLL, Power, AbsVal
- Softmax
- Reshape, Flatten, Slice, Split
- LRN (Local Response Normalization)
- MVN (Mean Variance Normalization)
- Dropout

Auch die Nutzung vortrainierter DNNs ist möglich.



OpenCV wurde hier genutzt um das DNN einzulesen,
die Videoframes für das DNN zu konvertieren und die Erkennung darzustellen.
Den Code findet man im Jupyter-Notebook.
(Mein Gesicht unscharf links unten)





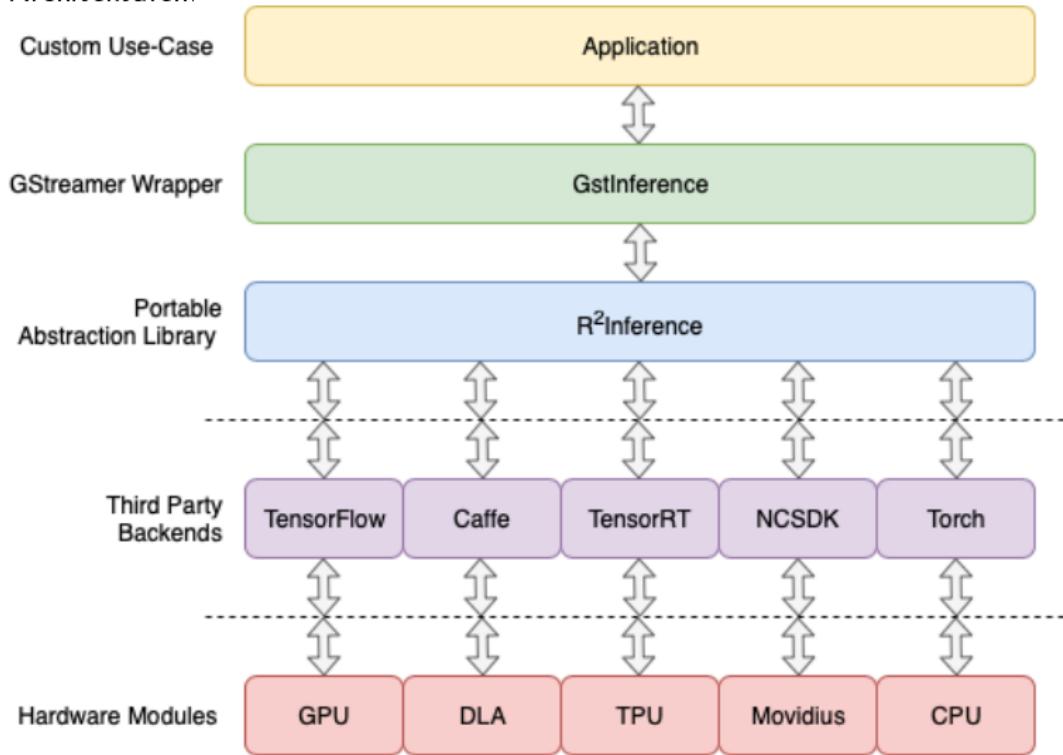
GstInference ermöglicht ein trainiertes DNN als Modul in eine gstreamer Pipeline zu integrieren.



- `bypass_sink`, sowie `bypass_src` erlauben das überspringen der Node.



Die R²Inference Bibliothek sorgt dabei für die Übersetzung aus den verschiedenen Architekturen.





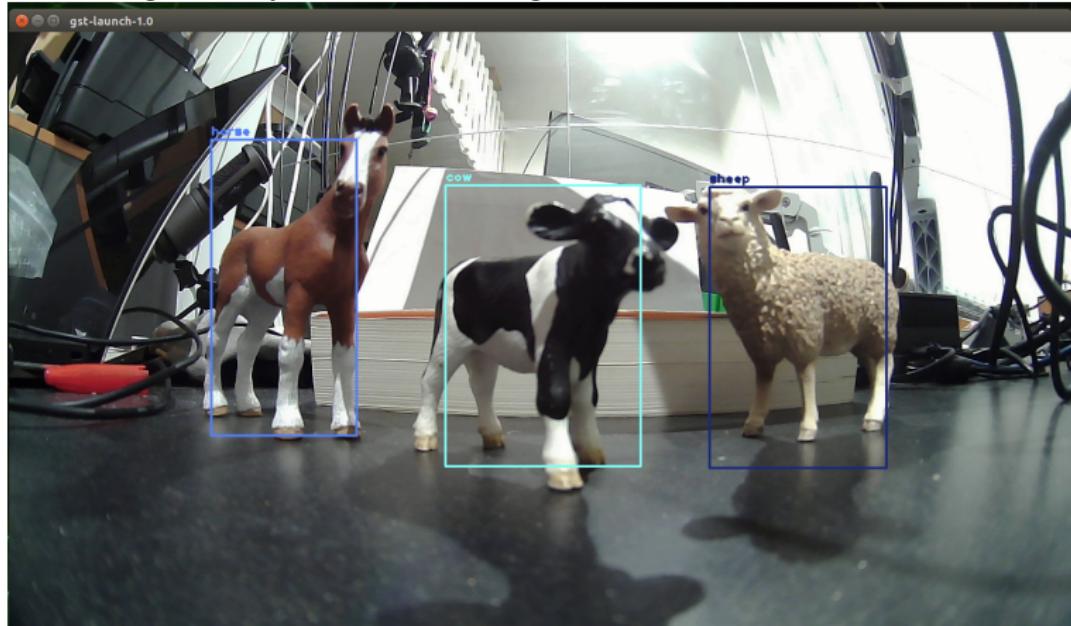
Dieses Bash-Skript erlaubt die gleiche Funktionalität mit höherer Effizienz als in OpenCV.

```
#!/bin/env bash
# gstreamer TinyYolo GstInference
CAMERA='/dev/video0'
MODEL_LOCATION='TinyYoloV2_TensorFlow/graph_tiny yolov2_tensorflow.pb'
INPUT_LAYER='input/Placeholder'
OUTPUT_LAYER='add_8'
LABELS='TinyYoloV2_TensorFlow/labels.txt'

gst-launch-1.0 v4l2src device=$CAMERA ! \
'video/x-raw, width=1280, height=720' ! \
tee name=t. ! videoconvert ! videoscale ! \
queue ! net.sink_model t. ! queue ! net.sink_bypass \
tiny yolov2 name=net model-location=$MODEL_LOCATION \
backend=tensorflow backend::input-layer=$INPUT_LAYER \
backend::output-layer=$OUTPUT_LAYER \
net.src_bypass ! videoconvert ! detectionoverlay labels="$(cat $LABELS)" \
font-scale=1 thickness=2 ! videoconvert ! xvimagesink sync=false
```



Darstellung mit TinyYoloV2 via vorheriger Gstreamer-Pipeline





RidgeRun erleichtert den Einstieg mit einer GUI zum erstellen gängiger Pipelines .

Platform: Jetson

Backend: TensorFlow

Model: Inceptionv4

Input layer: input

Output layer: InceptionV4/Logits/Predictions

Model location: graph_inceptionv4_tensorflow.pb

Labels: imagenet_labels.txt

Source: Camera stream (v4l2) ▾ Source location

Sink: nvoverlaysink

Optional utilities

The following elements are optional yet very useful. Check the documentation for more details on their properties.

- Inference Filter Filter class id.
- Inference Overlay 1 12 Dashed
- Inference Crop 3:3

Buttons: Reset, Generate

```
gst-launch-1.0 inceptionv4 name=net model.location=graph.inceptionv4_tensorflow.pb backend=tensorflow labels="$(cat imagenet_labels.txt)" backend::input-layer=input backend::output-layer=InceptionV4/Logits/Predictions v4l2src device=/dev/video0 ! nvvidconv ! queue ! tee name=t t. ! nvvidconv ! queue ! net.sink.model.t. ! nvvidconv ! video/x-raw,format=RGBA ! queue ! net.sink.bypass net.src.model inferencecrop aspect-ratio=3:3 ! fakesink net.src.bypass ! queue ! inferenceoverlay thickness=1 font-scale=12 style=2 ! videoconvert ! video/x-raw,format=I420 ! nvvidconv ! nvoverlaysink
```

https://developer.ridgerun.com/wiki/index.php?title=GstInference/Example_pipelines_with_hierarchical_metadata



gstd

Gstreamer Daemon (Ermöglicht die Listung und Steuerung für mehrere Pipelines)

Basis-Befehle

```
# Create the pipeline
pipeline_create testpipe videotestsrc name=vts ! autovideosink

# Play the pipeline
pipeline_play testpipe

# Change a property
element_set testpipe vts pattern ball

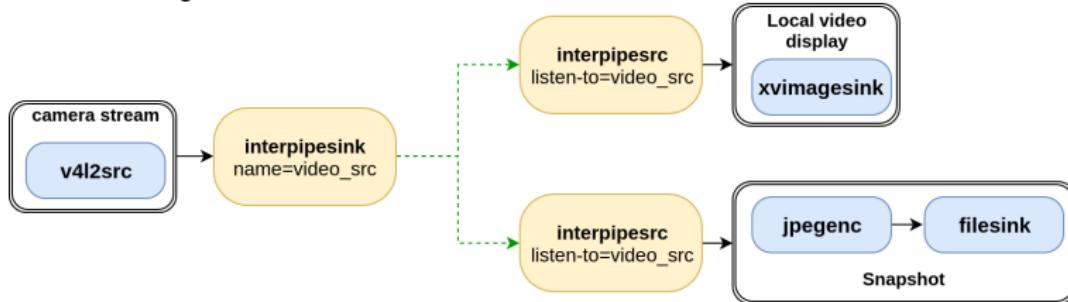
# Stop the pipeline
pipeline_stop testpipe

# Destroy the pipeline
pipeline_delete testpipe
```



GstInterPipe bietet weitere fortgeschrittene Funktionen

- Buffer Forwarding (warten auf def. Zustand)
- Dynamic Switching (Auswahl aus verschiedenen Pipelines)
- Caps Negotiation (garantiert Kompatibilität der PL)
- Event Forwarding (Ereignisbehandlung)
- Timestamp Synchronization
- New Node Notification
- Clock Sharing





Komplexe Pipelines in Shell-Skript, werden schnell ungebräuchlich und erschweren das Debugging. (Pipeline in Rosa)



https://docs.opencv.org/master/d9/df8/tutorial_root.html
<https://brettviren.github.io/pygst-tutorial-org/pygst-tutorial.html>
https://developer.ridge-run.com/wiki/index.php?title=GstInference/Example_pipelines_with_hierarchical_metadata
<https://github.com/nnstreamer/nnstreamer>
https://developer.ridge-run.com/wiki/index.php?title=GstInterpipe--Features_Detailed_Description
<https://devblogs.nvidia.com/building-iva-apps-using-deepstream-5-0/>