HAILO

Empowering Intelligence

TAPPAS User Guide

Release 3.30.0

30 September 2024

Table of Contents

1 Hailo TAPPAS - Optimized Execution of Video-Processing Pipelines 2
1.1 Overview . . . 2
1.2 Getting Started with Hailo-8 . . . 3
1.3 Getting Started with Hailo-15 . . . 4
1.4 Example Applications Built with TAPPAS . . . 4
1.5 Support . . . 8
1.6 Changelog . . . 8

2 Installation 14
2.1 Verify HailoRT Installation . . . 14
2.2 Installation via the Hailo SW Suite . . . 14
2.3 Using Dockers . . . 15
2.4 Yocto . . . 18
2.5 Manual Installation . . . 21
2.6 Run TAPPAS on Raspberry PI 4 . . . 25

3 GStreamer Based Examples 29
3.1 General . . . 29
3.2 x86 HW accelerated . . . 52
3.3 i.MX8 . . . 58
3.4 Raspberry Pi . . . 64
3.5 Rockchip . . . 68

4 Hailo15 C++ Example Application 76
4.1 Overview . . . 76
4.2 Running the Application . . . 76
4.3 Application at a Glance . . . 77
4.4 Seeing the Other Streams . . . 78
4.5 Where to go from here . . . 78

5 TAPPAS as a framework 79
5.1 TAPPAS Framework . . . 79
5.2 GStreamer Framework . . . 80
5.3 Write Your Own Application . . . 81

6 Elements 114
6.1 Hailo Cropper . . . 114
6.2 Hailo Overlay . . . 116
6.3 Hailo DeviceStats . . . 117
6.4 Hailo Filter . . . 118
6.5 Hailo Tracker . . . 120
6.6 Hailo Tile Aggregator . . . 121
6.7 Hailo Tile Cropper . . . 123
6.8 Hailo Python . . . 125
6.9 Hailo Net . . . 126
6.10 Hailo Muxer . . . 126
6.11 Hailo Aggregator . . . 127
6.12 Hailo Gallery . . . 129
6.13 Hailo RoundRobin . . . 130
6.14 Hailo Stream Router . . . 132
6.15 Hailo Export File . . . 134
6.16 Hailo Export ZMQ . . . 135

6.17 Hailo Import ZMQ . . . 136
6.18 Hailo On-Screen Display . . . 137
6.19 Hailo Upload . . . 140
6.20 Hailo Gray to NV12 . . . 142
6.21 Hailo NV12 to Gray . . . 143

7 Pipelines **145**
7.1 Cascaded Networks Structure . . . 145
7.2 Single Network Multi-Device Pipeline Structure . . . 147
7.3 Multi Stream Pipeline Structure . . . 148
7.4 Parallel Networks Structure . . . 149
7.5 Single Network Pipeline Structure . . . 150

8 Tools **154**
8.1 Dot Visulaizer . . . 154
8.2 Cross-compile Hailo's GStreamer plugins . . . 154

9 Scripts **158**
9.1 Using record_perf script to get records from tappas . . . 158

Disclaimer and Proprietary Information Notice

Copyright

This version of the document supersedes all previous versions.

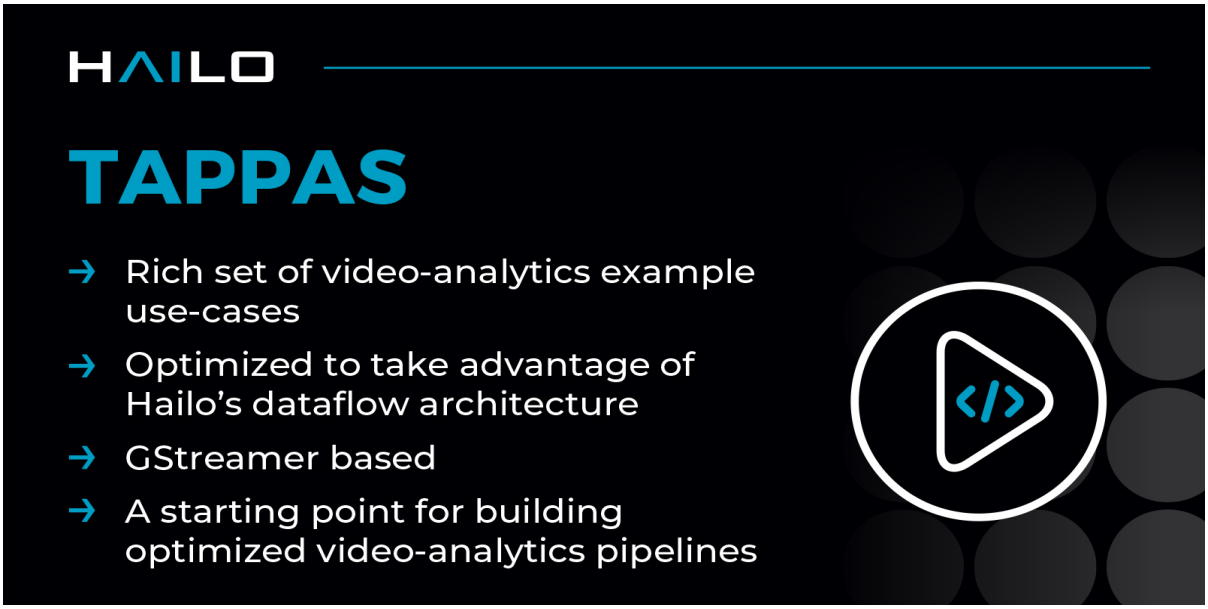
General Notice

Hailo, to the fullest extent permitted by law, provides this document "as-is" and disclaims all warranties, either express or implied, statutory or otherwise, including but not limited to the implied warranties of merchantability, non-infringement of third parties' rights, and fitness for particular purpose.

Although Hailo used reasonable efforts to ensure the accuracy of the content of this document, it is possible that this document may contain technical inaccuracies or other errors. Hailo assumes no liability for any error in this document, and for damages, whether direct, indirect, incidental, consequential or otherwise, that may result from such error, including, but not limited to loss of data or profits.

The content in this document is subject to change without prior notice and Hailo reserves the right to make changes to content of this document without providing a notification to its users.

1. Hailo TAPPAS - Optimized Execution of Video-Processing Pipelines



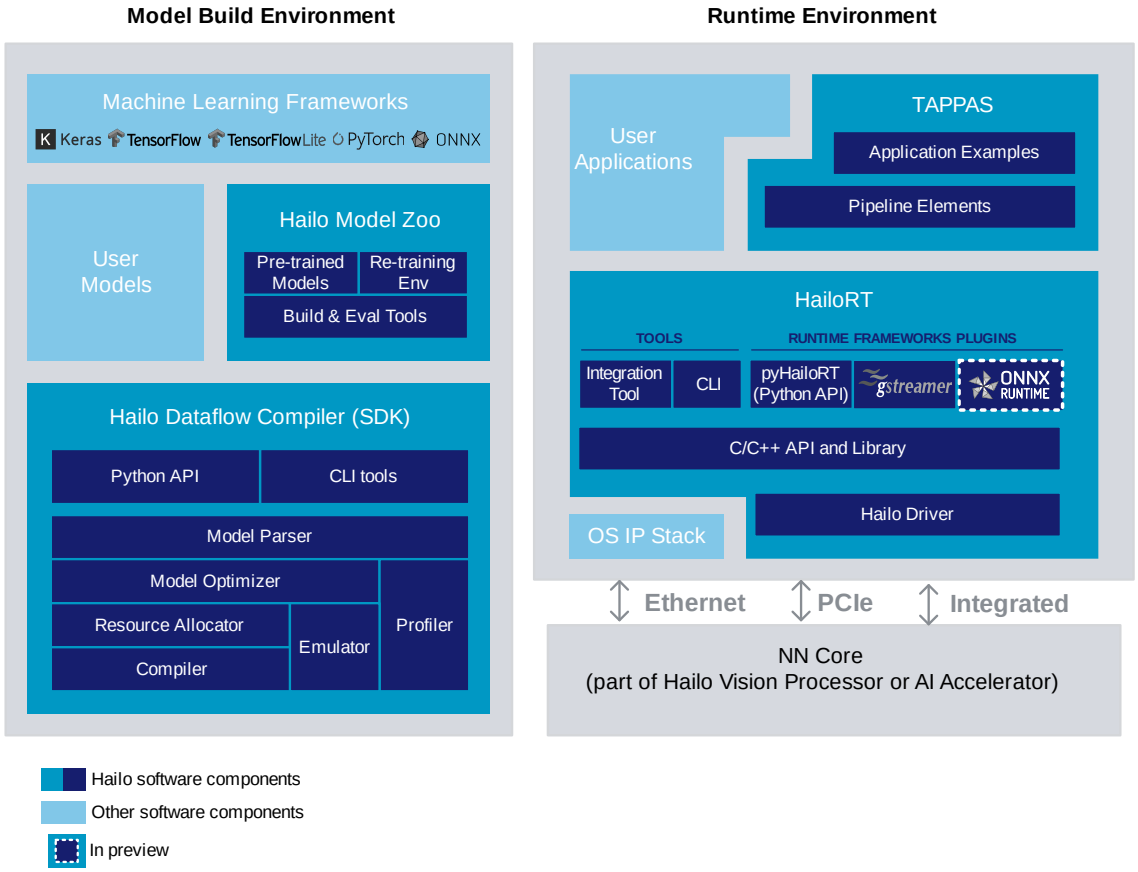
gstreamer 1.16 | 1.18 | 1.20 HailoRT 4.19.0 License LGPLv2.1

1.1. Overview

TAPPAS is Hailo's set of full application examples, implementing pipeline elements and pre-trained AI tasks.

Demonstrating Hailo's system integration scenario of specific use cases on predefined systems (software and Hardware platforms). It can be used for evaluations, reference code and demos:

- Accelerating time to market by reducing development time and deployment effort
- Simplifying integration with Hailo's runtime SW stack
- Providing a starting point for customers to fine-tune their applications



1.2. Getting Started with Hailo-8

1.2.1. Prerequisites

- Hailo-8 device
- HailoRT PCIe driver installed
- At least 6GB's of free disk space

Note: This version is compatible with HailoRT v4.19.

1.2.2. Installation

Option	Instructions	Supported OS
Hailo SW Suite*	*SW Suite Install guide*	Ubuntu x86 20.04, Ubuntu x86 22.04
Pre-built Docker image	*Docker install guide*	Ubuntu x86 20.04, Ubuntu x86 22.04, Ubuntu aarch64 20.04 (64-bit)
Manual install	*Manual install guide*	Ubuntu x86 20.04, Ubuntu x86 22.04, Ubuntu aarch64 20.04
Yocto installation	*Read more about Yocto installation*	Yocto supported BSP's
Raspberry Pi 4 installation	*Read more about Raspberry Pi 4 installation*	Raspberry Pi OS
Raspberry Pi 5 installation	Read more about Raspberry Pi 5 installation	Raspberry Pi OS

`* It is recommended to start your development journey by first installing the Hailo SW Suite`

1.2.3. Documentation

- *Framework architecture and elements documentation*
- *Guide to writing your own C++ postprocess element*
- *Guide to writing your own Python postprocess element*
- *Debugging and profiling performance*
- *Cross compile* - A guide for cross-compiling

1.3. Getting Started with Hailo-15

TAPPAS is now released separately for Hailo-8, for Hailo-15 please refer to https://github.com/hailo-ai/tappas/tree/master-vpu.

For a quick start with Hailo-15, please refer to the Vision Processor Software Package documentation section in Hailo's Developer Zone.

1.4. Example Applications Built with TAPPAS

Note: For the Raspberry Pi 5 applications, go to Hailo Raspberry Pi 5 Examples.

Note: These example applications are part of the Hailo AI Software Suite.

Hailo offers an additional set of Application Code Examples.

TAPPAS comes with a rich set of pre-configured pipelines optimized for different common hosts.

Important:

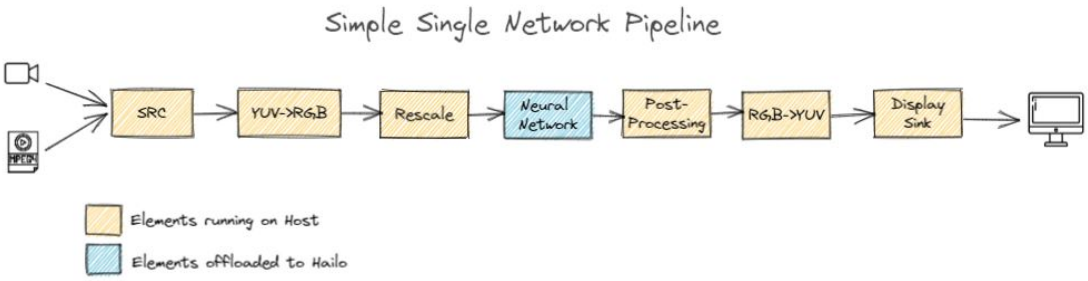
- All example applications utilize both the host (for non-neural tasks) and the Neural-Network Core (for neural-networks inference), therefore performance results are affected by the host.
- General application examples do not include any architecture-specific accelerator usage, and therefore will provide the easiest way to run an application, but with sub-optimal performance.
- Architecture-specific application examples (i.MX, Raspberry PI, etc.) use platform-specific hardware accelerators and are not compatible with different architectures.

Note: All i.MX example application are validated on i.MX8 and i.MX6 platforms and are compatible with the architectures.

Note: Running application examples requires a direct connection to a monitor.

1.4.1. Basic Single Network Pipelines

Pipelines that run a single network. The diagram below shows the pipeline data-flow.

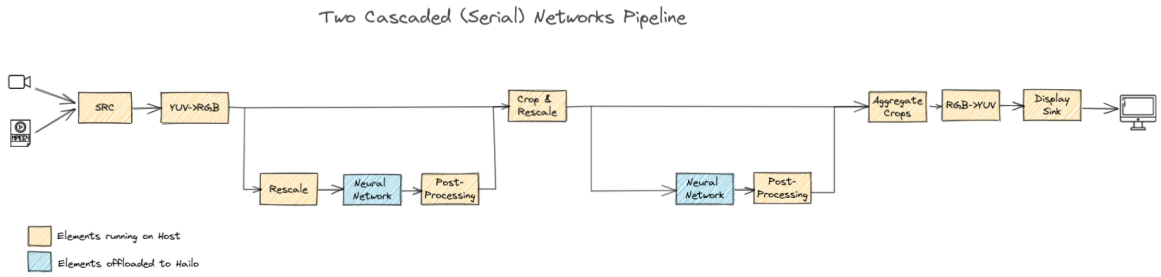


The following table details the currently available examples.

	General	*i.MX8*	*RPi4*	*x86 Hardware Accelerated*	*Rockchip*
Object Detection	✔	✔	✔		✔
Depth Estimation	✔	✔	✔		
Instance segmentation	✔				
Classification with Python Postprocessing	✔				
Object Detection Multiple Devices (Century)	✔			✔	
Face Recognition	✔				

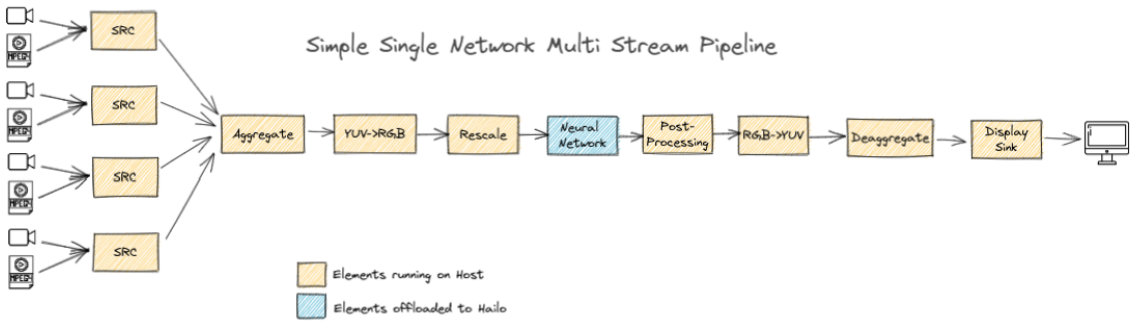
1.4.2. Two Network Pipelines

Examples of basic pipelines running two networks. The cascaded (serial) flow shows two networks running in series. This example pipeline is of the popular configuration where the first network is a detector which finds some Region-of-Interest (ROI) in the input image and the second network processes the cropped ROI (a face-detection-and-landmarking use case of this pipeline is shown at the top of this guide). The pipeline is shown in the following diagram:



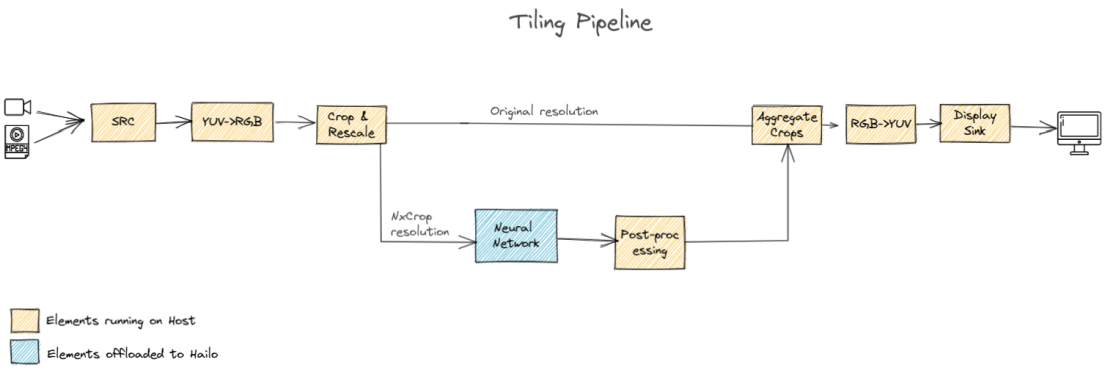
	General	i.MX8	RPi4	x86 Hardware Accelerated	Rockchip
Cascaded - Face Detection & Landmarks	✔		✔		
Cascaded - Person Det & Single Person Pose Estimation	✔	✔			
Cascaded - Face Detection & Recognition	✔				

1.4.3. Multi-Stream Pipelines



	General	i.MX8	RPi4	x86 Hardware Accelerated	Rockchip
Multi-stream Object Detection	✔			✔	✔
Multi-stream Multi-Device Object Detection	✔				

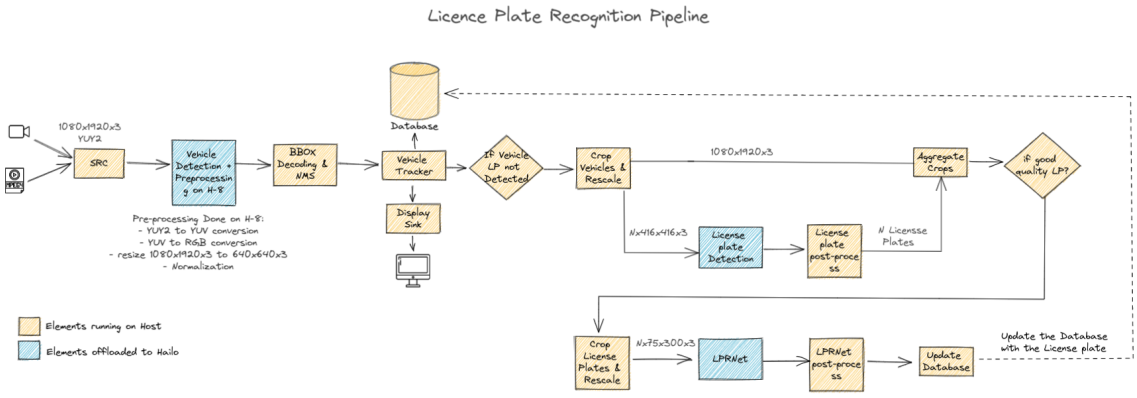
1.4.4. Pipelines for High-Resolution Processing Via Tiling



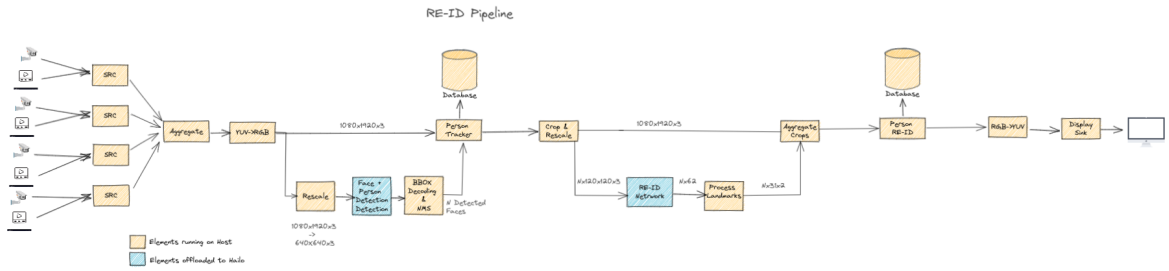
	General	i.MX8	RPi4	x86 Hardware Accelerated	Rockchip
HD Object Detection	✔				✔

1.4.5. Example Use Case Pipelines

Our LPR reference application demonstrates the use of 3 networks, with a database. The pipeline demonstrates inference based decision making (Vehicle detection) for secondary inference tasks (License plate data extraction). This allows multiple networks to cooperate in the pipeline for reactive behavior.



Our Multi-Person Multi-Camera Tracking reference application demonstrates person tracking across multiple streams using RE-ID tracking. The pipeline demonstrates another method for inference based decision making that also connects between different video streams.



	General	*i.MX8*	*RPi4*	*x86 Hardware Accelerated*	*Rockchip*
LPR	✔	✔			✔
RE-ID	✔				

1.5. Support

If you need support, please post your question on our Hailo community Forum for assistance.

Contact information is available at hailo.ai.

1.6. Changelog

v3.30.0 (October 2024)

- TAPPAS is now released separately for Hailo-8, for Hailo-15 please refer to https://github.com/hailo-ai/tappas/tree/master-vpu
- Various bug fixes and stability Improvements

v3.29.1 (August 2024)

- Hailo-8:
  - Updated infrastructure to better support Raspberry Pi 5
  - Added an option to control Yolo (Detection) Hailort post-process parameters via a JSON configuration
  - Semantic segmentation post-process now extracts the argmax tensor using Regular Expressions

v3.29.0 (July 2024)

- Hailo-15:
  - Updated all Hailo-15 example applications to use the latest API
  - Added a new C++ based example application that demonstrates the use of the Hailo-15 API *Link*
  - This release is aligned with the Hailo-15 Vision Processor Software Package 2024-07
- Hailo-8:

  - Added a new example application which demonstrates x86 hardware-accelerated *multi-stream detection*
  - Various bug fixes and stability improvements for Raspberry Pi 5
- Fixed various stability issues across apps and platforms

v3.28.1 (May 2024)

- Added a new property to `hailomuxer` which allows the sub-frame to be leaky
- `hailooverlay` now properly supports Hailo-15 with a new DMABuf sync mechanism
- `hailovideoscale` (Hailo-15 Element) now supports the `letterbox` property

v3.28.0 (April 2024)

- Tappas was updated in this version, with a revised list of supported platform and apps
- Added `yolov8` (as default) to Detection application examples
- Fixed various stability issues across apps and platforms
- This release is aligned to Hailo-15 Vision Processor Software Package 2024-04

v3.27.2 (March 2024)

- Aligned to Hailo-15 Vision Processor Software Package 2024-01.2
- Added a new example application - Frontend Update
- Updated the hailo OSD API
- Detection application now works with an updated `hailonet` element
- Various bugs fixes

v3.27.1 (February 2024)

- Aligned to Hailo-15 Vision Processor Software Package 2024-01.1
- Added a new Hailo-15 external host application that saves udp stream to file

v3.27.0 (January 2024)

- Updated the Hailo-15 applications to use the updated Media Library implementation:
  - Basic Security Camera (streaming)
  - Detection
  - Single Stream OSD (On-Screen Display)
- Added a folder for external host scripts and added the UDP Stream Display script

---

Note: TAPPAS supports both Hailo-15 and Hailo-8. Temporarily, in this version, only the following Hailo-8 based example applications are supported:

- Detection
  - yolov5
  - mobilenet_ssd
- Multi-Stream Detection
  - Multi-Stream Detection
  - MultiStream Detection with Stream Multiplexer
- License Plate Recognition

These applications are supported under the general folder (x86-based platforms).

---

v3.26.2 (December 2023)

- Aligned to Hailo-15 Vision Processor Software Package 2023-10.2
- Add grayscale support for Media Library Front-end
- Various bug fixes for Hailo-15 pipelines

v3.26.1 (November 2023)

- Aligned to Hailo-15 Vision Processor Software Package 2023-10.1
- Updated OSD configuration to support new dynamic features and adjust to the updated Media Library implementation
- Added a script for displaying UDP streams
- Basic security camera (Media Library implementation) now support 5 outputs

v3.26.0 (October 2023)

- Added Hailo-15 supported application examples:
  - Detection
  - License Plate Recognition
  - A new Hailo-15 specific example application - Basic Security Camera (streaming)
- Removed Yolact models support from Instance Segmentation
- Various bug fixes:
  - Fixed the Detection application on i.MX6 platforms
  - Fixed an issue with Face Recognition which prevented faces to be recognized ins some scenarios
  - Fixed an issue which caused a warning when running some networks

v3.25.0 (July 2023)

- Improved Yolov5seg post-process performance
- Updated Yolo networks to use the HailoRT native post-process (selected models)
- Added “non-blocking mode” and “wait-time” properties to hailoroundrobin element

v3.24.0 (March 2023)

- Added support for *Rockchip RK3588* (validated on Firefly ITX-3588J platform)
- Video Management System now supports multi-device (Ubuntu 22.04 only)
- Video Management System (single device) now works on Ubuntu 20.04
- Added a new model to *Instance Segmentation Pipeline*:
  - *yolov5seg* - which has improved performance compared to *yolact*
- New applications for *i.MX8*:
  - Object Detection and Pose Estimation (cascaded)
  - Multi-Stream Detection
- Added a TAPPAS Graphic User Interface to easily run selected general example applications (preview) on the TAPPAS Docker - to activate it, run *tappas-gui*
- Added back *yolox_l_leaky* to the *Century general application*
- Reduced docker size

v3.23.1 (February 2023)

- Updated to HailoRT 4.12.1
- Fixed a documentation mistake in *Writing your own Python postprocess*

v3.23.0 (December 2022)

- New Apps:
  - Added *x86_hw_accelerated* example pipelines that use Video Acceleration API (VA-API) over Intel processors that support Quick Sync:
    * Video Management System - a pipeline that demonstrates a VMS application which runs several streams and different tasks - Face Recognition, Face Attributes and Person Attributes. Currently this example pipeline is supported on Ubuntu 22.04 only
    * *Multi-stream detection*
    * *Century*
  - Pose Estimation pipeline with two cascading networks - *Person detection and single person pose estimation*
  - *Face recognition*
  - Updated i.MX6 Object Detection App - New network, updated the pipeline to include i.MX6 hardware acceleration
- Added new models to *Instance Segmentation Pipeline*:
  - yolact_regnetx_1.6gf
  - yolact_regnetx_800mf (80 classes)
- *Century app* now uses a new network (yolov5m)
- *Multi-Camera Multi-Person Tracking (RE-ID)* - Improved pipeline performance and accuracy
- Added support for Ubuntu 22.04 (release-grade)

v3.22.0 (November 2022)

- New element *hailoimportzmq* - provides an entry point for importing metadata exported by *hailoexportzmq* (HailoObjects) into the pipeline
- Added Depth Estimation, Object Detection and Classification pipelines for i.MX6 Pipelines
- Changed the debugging tracers to use an internal tracing mechanism

v3.21.0 (October 2022)

- New Apps:
  - *Multi-stream detection that uses HailoRT Stream Multiplexer* - Demonstrates the usage of HailoRT stream multiplexer (preview)
- New elements - *hailoexportfile* and *hailoexportmq* which provide an access point in the pipeline to export metadata (HailoObjects)
- Improved pipeline profiling by adding new tracers and replacing the GUI of *gst-shark*
- Ubuntu 22 is now supported (GStreamer 1.20, preview)
- Yocto Kirkstone is now supported (GStreamer 1.20)

v3.20.0 (August 2022)

- New Apps:
  - *Detection every X frames pipeline* - Demonstrates the ability of skipping frames using a tracker
- Improvements to Multi-Camera Multi-Person Tracking (RE-ID) pipeline (released)

v3.19.1 (July 2022)

- New Apps:
  - Multi-Camera Multi-Person Tracking (RE-ID) pipeline *multi_person_multi_camera_tracking.sh* (preview)

v3.19.0 (June 2022)

- New Apps:

  - Added Cascading networks, Depth Estimation, Pose Estimation and Semantic Segmentation pipelines for *i.MX Pipelines*
- Added an option to control post-process parameters via a JSON configuration for the detection application
- Added support for Raspberry Pi Raspbian OS
- Native Application now uses TAPPAS post-process
- LPR (License Plate Recognition) pipeline is simplified to bash only
- New detection post-process - Nanodet

Note: Ubuntu 18.04 will be deprecated in TAPPAS future version

Note: Python 3.6 will be deprecated in TAPPAS future version

v3.18.0 (April 2022)

- New Apps:
  - LPR (License Plate Recognition) pipeline and facial landmark pipeline for *i.MX Pipelines*
- Added the ability of compiling a specific TAPPAS target (post-processes, elements)
- Improved the performance of Raspberry Pi example applications

v3.17.0 (March 2022)

- New Apps:
  - LPR (License Plate Recognition) pipeline for *General Pipelines* (preview)
  - Detection & pose estimation app
  - Detection (MobilenetSSD) - Multi scale tiling app
- Update infrastructure to use new HailoRT installation packages
- Code is now publicly available on Github

v3.16.0 (March 2022)

- New Apps:
  - Hailo Century app - Demonstrates detection on one video file source over 6 different Hailo-8 devices
  - Python app - A classification app using a post-process written in Python
- New Elements:
  - Tracking element “HailoTracker” - Add tracking capabilities
  - Python element “HailoPyFilter” - Enables to write post-processes using Python
- Yocto Hardknott is now supported
- Raspberry Pi 4 Ubuntu dedicated apps
- HailoCropper cropping bug fixes
- HailoCropper now accepts cropping method as a shared object (.so)

v3.14.1 (March 2022)

- Fix Yocto Gatesgarth compilation issue
- Added support for hosts without X-Video adapter

v3.15.0 (February 2022)

- New Apps:

  - Detection and depth estimation - Networks switch app
  - Detection (MobilenetSSD) - Single scale tilling app

v3.14.0 (January 2022)

- New Apps:
  - Cascading apps - Face detection and then facial landmarking
- New Yocto layer - Meta-hailo-tappas
- Window enlargement is now supported
- Added the ability to run on multiple devices
- Improved latency on Multi-device RTSP app

v3.13.0 (November 2021)

- Context switch networks in multi-stream apps are now supported
- New Apps:
  - Yolact - Instance segmentation
  - FastDepth - Depth estimation
  - Two networks in parallel on the same device - FastDepth + Mobilenet SSD
  - Retinaface
- Control Element Integration - Displaying device stats inside a GStreamer pipeline (Power, Temperature)
- New Yocto recipes - Compiling our GStreamer plugins is now available as a Yocto recipe
- Added a C++ detection example (native C++ example for writing an app, without GStreamer)

v3.12.0 (October 2021)

- Detection app - MobilenetSSD added
- NVR multi-stream multi device app (detection and pose estimation)
- Facial Landmarks app
- Segmentation app
- Classification app
- Face detection app
- Hailomuxer gstreamer element
- Postprocess implementations for various networks
- GStreamer infrastructure improvements
- Added ARM architecture support and documentation

v3.11.0 (September 2021)

- GStreamer based initial release
- NVR multi-stream detection app
- Detection app
- Hailofilter gstreamer element
- Pose Estimation app

2. Installation

2.1. Verify HailoRT Installation

Note: This section does not apply to Hailo SW Suite users since HailoRT is already installed.

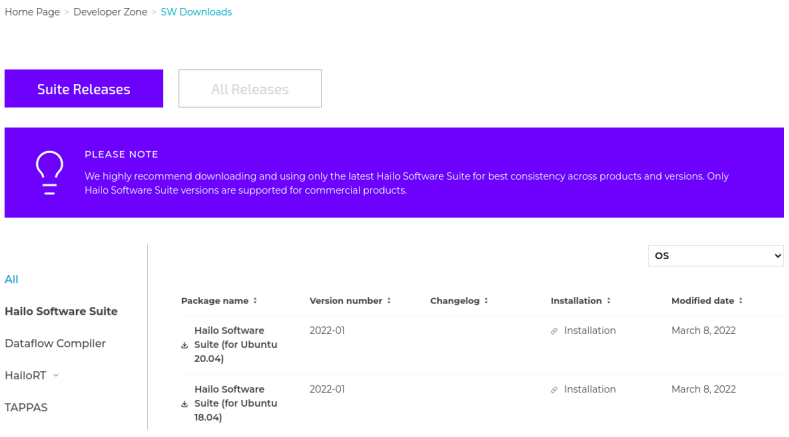
Confirm that hailo has been identified correctly by running this command: `hailortcli fw-control identify`, The expected output should look similar to the one below:

```
$ hailortcli fw-control identify
Identifying board
Control Protocol Version: 2
Firmware Version: X.X.X (develop,app)
Logger Version: 0
Board Name: Hailo-8
Device Architecture: HAILO8_B0
Serial Number: 0000000000000009
Part Number: HM218B1C2FA
Product Name: HAILO-8 AI ACCELERATOR M.2 M KEY MODULE
```

2.2. Installation via the Hailo SW Suite

2.2.1. Download the SW Suite

To get started, go to the Hailo Developer Zone and enter the SW Suite download page. From there you can download your prefered version of the `Hailo SW Suite`. If this is your first time in the `Developer Zone`, you will be asked to register to the site before downloads are enabled, login and then you will be able to download the package.



2.2.2. Install the SW Suite

Once the zip is downloaded, extract the folder with:

```
tar -xf hailo_sw_suite_2023_01.tar.gz
```

There are three options for installing the `Suite`: `Docker` installation, `Manual` installation, and a `Self Extractable` installation.

The instructions can be found for all installation flows, along with their required pre-requisites and sanity checks, in the Hailo Developer Zone.

This is all that is required to start the installation. TAPPAS is included in the SW Suite, so by installing the Suite you should now have the corresponding version of TAPPAS installed inside.

Note that the TAPPAS system requirements are documented in the Required Packages section *Here*.

2.3. Using Dockers

2.3.1. Install Docker

The section below would help you with the installation of Docker.

```
# Install curl
sudo apt-get install -y curl

# Get and install docker
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh

# Add your user (who has root privileges) to the Docker group
sudo usermod -aG docker $USER

# Reboot/log out in order to apply the changes to the group
sudo reboot
```

Note: Consider reading *Running out of disk space* if your system is space limited

2.3.2. Running TAPPAS container from pre-built Docker image

Preparations

HailoRT PCIe driver is required - install instructions are provided in HailoRT documentation. Make sure that the driver is installed correctly by: *Verify Hailo installation*.

Note: The TAPPAS container already contains the required HailoRT version pre-installed.

Download from Hailo developer zone tappas_VERSION_ARCH_docker.zip and unzip the file, it should contain the following files:

- **hailo_docker_tappas_VERSION.tar**: the pre-built docker image
- **run_tappas_docker.sh**: Script that loads and runs the docker image
- **dockerfile.tappas_run**: Dockerfile used within the first load

Running for the first time

In order to use TAPPAS release Docker image, you should run the following script:

```
./run_tappas_docker.sh --tappas-image TAPPAS_IMAGE_PATH
```

Note: TAPPAS_IMAGE_PATH is the path to the **hailo_docker_tappas_VERSION.tar**

The script would load the docker image, and start a new container. The script might take a couple of minutes, and after that, you are ready to go.

Resuming (Second time and on)

From now an on you should run the script with the `--resume` flag

```
./run_tappas_docker.sh --resume
```

Note: The reason that you want to use the `--resume` flag is that the container already exists, so only attaching to the container is required.

Flags and advanced use-cases

```
./run_tappas_docker.sh [options]
Options:
 --help        Show this help
 --tappas-image    Path to tappas image
 --resume       Resume an old container
 --container-name    Start a container with a specific name, defaults to hailo_tappas_
↪container
```

Use-cases

For building a new container with the default name:

```
./run_tappas_docker.sh --tappas-image TAPPAS_IMAGE_PATH
```

For resuming an old container:

```
./run_tappas_docker.sh --resume
```

Both of this methods can receive a container name:

```
./run_tappas_docker.sh --tappas-image TAPPAS_IMAGE_PATH --container-name CONTAINER_
↪NAME
./run_tappas_docker.sh --resume  --container-name CONTAINER_NAME
```

for example:

```
./run_hailort_docker.sh hailo_docker_tappas_3.14.0.tar --container-name hailo_
↪tappas_container
```


2.3.3. Upgrade Version

To upgrade, run: *Running TAPPAS container from pre-built Docker image*

Note: TAPPAS requires a specific HailoRT version, therefore, you will might need to upgrade HailoRT version as well. To check which HailoRT version is supported, please visit *This Link*

2.3.4. Troubleshooting

Hailo containers are taking to much space

Creating new docker containers with `--override` does not assure that the directory of cached images and containers is cleaned. to prevent your system to ran out of memory and clean /var/lib/docker run `docker system prune` from time to time.

Running out of disk space

Change Docker root directory - By default, Docker stores most of its data inside the `/var/lib/docker` directory on Linux systems. There may come a time when you want to move this storage space to a new location. For example, the most obvious reason might be that you're running out of disk space.

Firstly, stop the Docker from running

```
$ sudo systemctl stop docker.service
$ sudo systemctl stop docker.socket
```

Next, we need to edit the `/lib/systemd/system/docker.service` file

```
$ sudo vim /lib/systemd/system/docker.service
```

The line we need to edit looks like this:

```
ExecStart=/usr/bin/dockerd -H fd://
```

Edit the line by putting a `-g` and the new desired location of your Docker directory. When you're done making this change, you can save and exit the file.

```
ExecStart=/usr/bin/dockerd -g /new/path/docker -H fd://
```

```
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket containerd.service

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -g /new/path/docker -H fd:// --containerd=/run/containerd/containerd.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always

# Note that StartLimit* options were moved from "Service" to "Unit" in systemd 229.
```

If you haven't already, create the new directory where you plan to move your Docker files to.

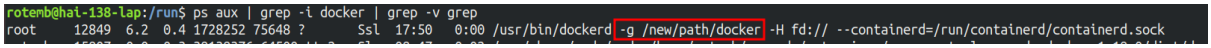
```
$ sudo mkdir -p /new/path/docker
```

Next, reload the systemd configuration for Docker, since we made changes earlier. Then, we can start Docker.


```
$ sudo systemctl daemon-reload
$ sudo systemctl start docker
```

Just to make sure that it worked, run the ps command to make sure that the Docker service is utilizing the new directory location.

```
$ ps aux | grep -i docker | grep -v grep
```



Cannot allocate memory in static TLS block

In some sceneraios (especially aarch64), you might face the following error:

```
(gst-plugin-scanner:15): GStreamer-WARNING **: 13:58:20.557: Failed to load plugin '/
↪usr/lib/aarch64-linux-gnu/gstreamer-1.0/libgstlibav.so': /lib/aarch64-linux-
↪gnu/libgomp.so.1: cannot allocate memory in static TLS block
```

The solution is to export an enviroment variable:

```
export LD_PRELOAD=/usr/lib/aarch64-linux-gnu/libgomp.so.1
```

2.4. Yocto

This section will guide through the integration of Hailo's Yocto layer's into your own Yocto environment.

Two layers are provided by Hailo, the first one is `meta-hailo` which compiles the `HailoRT` sources, and the second one is `meta-hailo-tappas` which compiles the `TAPPAS` sources.

`meta-hailo-tappas` is a layer that based un-top of `meta-hailo` that adds `TAPPAS` recipes.

The layers are stored in Meta-Hailo Github, with branch for each supported yocto release:

- Dunfell 3.1 (kernel 5.4.85)
- Kirkstone 4.0 (kernel 5.15)

Warning: On i.MX8-based devices Kirkstone branch does not support OpenGL, therefore, the Kirkstone applications portfolio is reduced.

2.4.1. Setup

HailoRT

Add the following to your image in your `conf/bblayers.conf`:

```
BBLAYERS += " ${BSPDIR}/sources/meta-hailo/meta-hailo-accelerator \
       ${BSPDIR}/sources/meta-hailo/meta-hailo-libhailort"
```

Add the recipes to your image in your `conf/local.conf`:

```
IMAGE_INSTALL_append = "hailo-firmware libhailort hailortcli hailo-pci libgsthailo"
```


TAPPAS

Add the following to your image in your `conf/bblayers.conf`:

```
BBLAYERS += "${BSPDIR}/sources/meta-hailo/meta-hailo-tappas"
```

Add the following to your image in your `conf/local.conf`:

```
IMAGE_INSTALL:append = "libgsthailotools tappas-apps hailo-post-processes tappas-
↪tracers"
```

2.4.2. Build your image

Run bitbake and build your image. After the build successfully finished, burn the Image to your embedded device.

Note: building on non-IMX devices: To increase the performance of our applications, we patched imx gstreamer-plugins-base. In non-IMX devices you may encounter an error indicating that recipes under `meta-hailo-tappas/recipes-multimedia/gstreamer/` cannot be parsed. In this case remove this directory under the meta-hailo-tappas layer, and re-build the image.

```
rm -rf meta-hailo/meta-hailo-tappas/recipes-multimedia/gstreamer/
```

Validating the integration's success

Make sure that the following conditions have been met on the target device:

- Running `hailortcli fw-control identify` prints the right configurations
- Running `gst-inspect-1.0 | grep hailo` returns hailo elements:

  ```
  hailo: hailonet: hailonet element
  hailodevicestats: hailodevicestats element
  ```

- Running `gst-inspect-1.0 | grep hailotools` returns hailotools elements:

  ```
  hailotools: hailomuxer: Muxer pipe fitting
  hailotools: hailofilter: Hailo postprocessing and drawing element
  ...
  ```

- post-processes shared object files exists at `/usr/lib/hailo-post-processes`

2.4.3. Recipes

libgsthailo

Hailo's GStreamer plugin for running inference on the hailo8 chip. Depends on `libhailort` and GStreamer.

The recipe compiles and copies the `libgsthailo.so` file to `/usr/lib/gstreamer-1.0` on the target device's root file system, make it loadable by GStreamer as a plugin.

libgsthailotools

Hailo's TAPPAS gstreamer elements. Depends on `libgsthailo`, GStreamer, opencv, xtensor and xtl. The source files located in the TAPPAS release under `core/hailo`. The recipe compiles with meson and copies the `libgsthailotools.so` file to `/usr/lib/gstreamer-1.0` on the target device's root file system.

tappas-apps

Hailo's TAPPAS embedded application recipe, including GStreamer apps for embedded. The recipe copies the app script, the hef and media files to /home/root/apps/. Depends on GStreamer, opencv, cxxopts, xtensor and xtl.

hailo-post-processes

The recipe compiles and copies the post processes to `/usr/lib/hailo-post-processes`. Deppends on opencv, xtensor, xtl, rapidjson and cxxopts.

tappas-tracers

Hailo's TAPPAS gstreamer tracers. Depends on `libgsthailo` and GStreamer. The source files located in the TAPPAS release under `core/hailo/tracers`. The recipe compiles with meson and copies the `libgsthailotracers.so` file to `/usr/lib/gstreamer-1.0` on the target device's root file system.

For instructions on how to use the tracers on a yocto built machine, see *debugging*

2.4.4. Troubleshooting

1. The device does not appear on lspci

If the device does not appear after running lspci, there may be two possible reasons:

- Symptom:

  The device is not connected correctly

- Symptom:

  The u-boot device tree does not support pcie.

  Solution:

  To fix this, replace the ftd_file you are using on u-boot.

```
setenv fdt_file imx6q-sabresd-pcie.dtb
```

2. HDMI port is connected but there is no display

Symptom:

On some imx devices you need to manually configure the u-boot to show video using HDMI port.

Solution:

To fix this issue you should set the u-boot to use HDMI port, defining the resolution, FPS and output format. The configuration is "added" (do not override this) to the mmcargs:

For example on IMX6Q-Sabresd, this the default value of mmargs:

```
mmcargs="setenv bootargs console=${console},${baudrate} ${smp} root=$
↪{mmcroot}"
```


Using this command we add the needed info to this variable:

```
setenv mmcargs "setenv bootargs console=${console},${baudrate} ${smp} root=
↪${mmcroot} video=mxcfb0:dev=hdmi,1280x720M@30,if=RGB24"
```

2.5. Manual Installation

The manual installation of TAPPAS requires preparation, Hailo's recommended method is to begin with `Hailo SW Suite` or `Pre-built Docker image`. This guide will instruct how to install the required components manually.

Note: Only Ubuntu 20.04 and 22.04 are supported

2.5.1. Hailort Installation

First HailoRT + HailoRT PCIe driver, needs to be installed. Follow the HailoRT installation guide for further instructions. After the installation, *confirm that HailoRT is working correctly*.

2.5.2. Preparations

Download from Hailo developer zone `tappas_VERSION_linux_installer.zip`.

x86

Unzip `tappas_VERSION_linux_installer.zip`

```
unzip tappas_VERSION_linux_installer.zip
```

Non-x86

1. Unzip TAPPAS and clone HailoRT sources

   ```
   unzip tappas_VERSION_linux_installer.zip
   ```

2. Change the directory to TAPPAS, and create a directory named `hailort` and clone `HailoRT` sources

   ```
   cd `tappas_VERSION`
   mkdir hailort
   git clone https://github.com/hailo-ai/hailort.git hailort/sources
   ```

Note: In some cases, HailoRT (which is installed via TAPPAS) is not the latest version, while the driver, which was installed before HailoRT and TAPPAS - has a different version. In these cases, this error will occur:

> "Could not find a configuration file for package "HailoRT" that exactly matches requested version."

Meaning, HailoRT library and driver installed versions are mismatched - while they are required to be fully identical.

In such a case - re-install the driver version which is compatible to your installed HailoRT version.

2.5.3. Required Packages

The following APT packages need to be installed, using the command below:

- ffmpeg
- x11-utils
- python3 (pip and setuptools).
- python3-virtualenv
- python-gi-dev
- libgirepository1.0-dev
- gcc-12 and g++-12
- cmake
- libzmq3-dev
- git
- rsync

To install the above packages, run the following command:

```
sudo apt-get install -y rsync ffmpeg x11-utils python3-dev python3-pip python3-
↪setuptools python3-virtualenv python-gi-dev libgirepository1.0-dev gcc-12 g++-12
↪cmake git libzmq3-dev
```

The following packages are required as well, and their installation instructions can be viewed from the links below:

- *OpenCV installation*.
- *GStreamer installation*.
- *PyGobject installation*.

In case any requirements are missing, a requirements table will be printed when calling manual installation.

2.5.4. OpenCV Installation

To install OpenCV, run the following commands:

```
sudo apt-get install -y libopencv-dev python3-opencv
```

To check your OpenCV version, run the following command:

```
# To check the OpenCV version installed
pkg-config --modversion opencv4
```

Tip: If you are running on an old OS the apt-get version might be too old (You will be notified on the next steps), you can install OpenCV manually as shown below.

Opencv compilation from source

```
# Download Opencv and unzip
wget https://github.com/opencv/opencv/archive/4.5.2.zip
unzip 4.5.2.zip

# cd and make build dir
cd opencv-4.5.2
mkdir build
cd build

# Make and install
cmake -DOPENCV_GENERATE_PKGCONFIG=ON \
  -DBUILD_LIST=core,imgproc,imgcodecs,calib3d,features2d,flann \
  -DCMAKE_BUILD_TYPE=RELEASE \
  -DWITH_PROTOBUF=OFF -DWITH_QUIRC=OFF \
  -DWITH_WEBP=OFF -DWITH_OPENJPEG=OFF \
  -DWITH_GSTREAMER=OFF -DWITH_GTK=OFF \
  -DOPENCV_DNN_OPENCL=OFF -DBUILD_opencv_python2=OFF \
  -DINSTALL_C_EXAMPLES=ON \
  -DINSTALL_PYTHON_EXAMPLES=ON \
  -DCMAKE_INSTALL_PREFIX=/usr/local ..

num_cores_to_use=$(($(nproc)/2))
make -j$num_cores_to_use
sudo make install

# Update the linker
sudo ldconfig
```

2.5.5. GStreamer Installation

Run the following command to install GStreamer:

```
sudo apt-get install -y libcairo2-dev libgirepository1.0-dev libgstreamer1.0-dev
↪libgstreamer-plugins-base1.0-dev libgstreamer-plugins-bad1.0-dev gstreamer1.0-
↪plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-
↪plugins-ugly gstreamer1.0-libav gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-
↪alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio
↪gcc-12 g++-12 python-gi-dev
```

Please refer to: GStreamer official installation guide for more details

2.5.6. PyGobject Installation

Run the following command to install PyGobject:

```
sudo apt install python3-gi python3-gi-cairo gir1.2-gtk-3.0
```

Please refer to: PyGobject official installation guide for more details

2.5.7. TAPPAS Installation

On most platforms (such as x86-based platforms), run:

```
./install.sh --skip-hailort
```

and then, *Make sure that HailoRT works*

Raspberry Pi 4 has its own set of example applications, so the installation command on this platform is slightly different:

```
./install.sh --skip-hailort --target-platform rpi
```

and then, *return to the Raspberry Pi section*

On Rockchip, run:

```
./install.sh --skip-hailort --target-platform rockchip
```

and then, return to the Rockchip section.

2.5.8. Upgrade TAPPAS

To Upgrade TAPPAS, first clean the GStreamer cache

```
rm -rf ~/.cache/gstreamer-1.0/
```

Remove old `libgsthailotools.so`

```
rm /usr/lib/$(uname -m)-linux-gnu/gstreamer-1.0/libgsthailotools.so
```

and then, *TAPPAS installation section*

2.5.9. Troubleshooting

Cannot allocate memory in static TLS block

In some scenarios (especially `aarch64`), this warning might occur:

```
(gst-plugin-scanner:15): GStreamer-WARNING **: 13:58:20.557: Failed to load plugin '/
↪usr/lib/aarch64-linux-gnu/gstreamer-1.0/libgstlibav.so': /lib/aarch64-linux-
↪gnu/libgomp.so.1: cannot allocate memory in static TLS block
```

The solution is to export an environment variable:

```
export LD_PRELOAD=/usr/lib/aarch64-linux-gnu/libgomp.so.1
```

PCIe descriptor page size error

If you encounter the following error: (actual page size might vary)

```
[HailoRT] [error] CHECK_AS_EXPECTED failed - max_desc_page_size given 16384 is
↪bigger than hw max desc page size 4096"
```

Some hosts doesn't support certain PCIe descriptor page size. in order to overcome this issue add the text below to /etc/modprobe.d/hailo_pci.conf (create the file if it doesn't exist)


```
options hailo_pci force_desc_page_size=4096
# you can do this by running the following command:
echo 'options hailo_pci force_desc_page_size=4096' >> /etc/modprobe.d/hailo_pci.
↪conf
```

Reboot the machine for this change to take effect. You can also reload the driver without rebooting by running the following commands:

```
modprobe -r hailo_pci
modprobe hailo_pci
```

2.6. Run TAPPAS on Raspberry PI 4

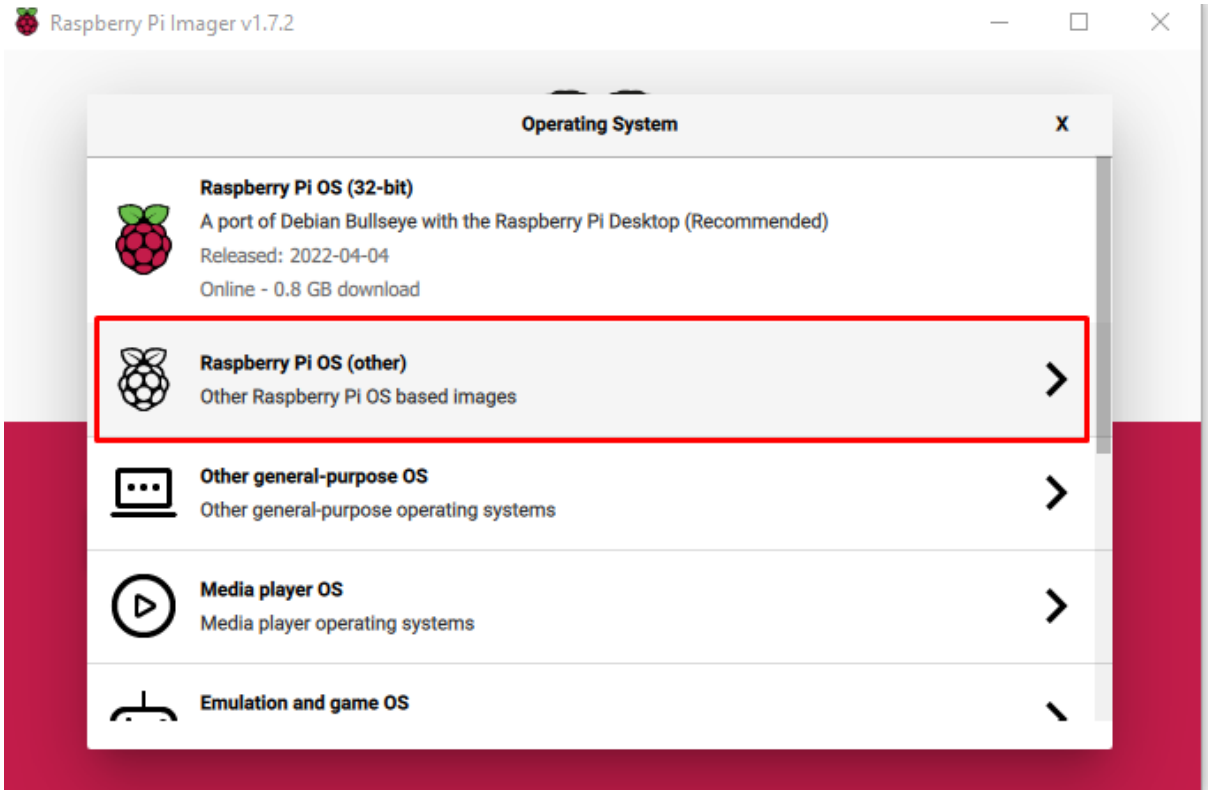
2.6.1. Overview

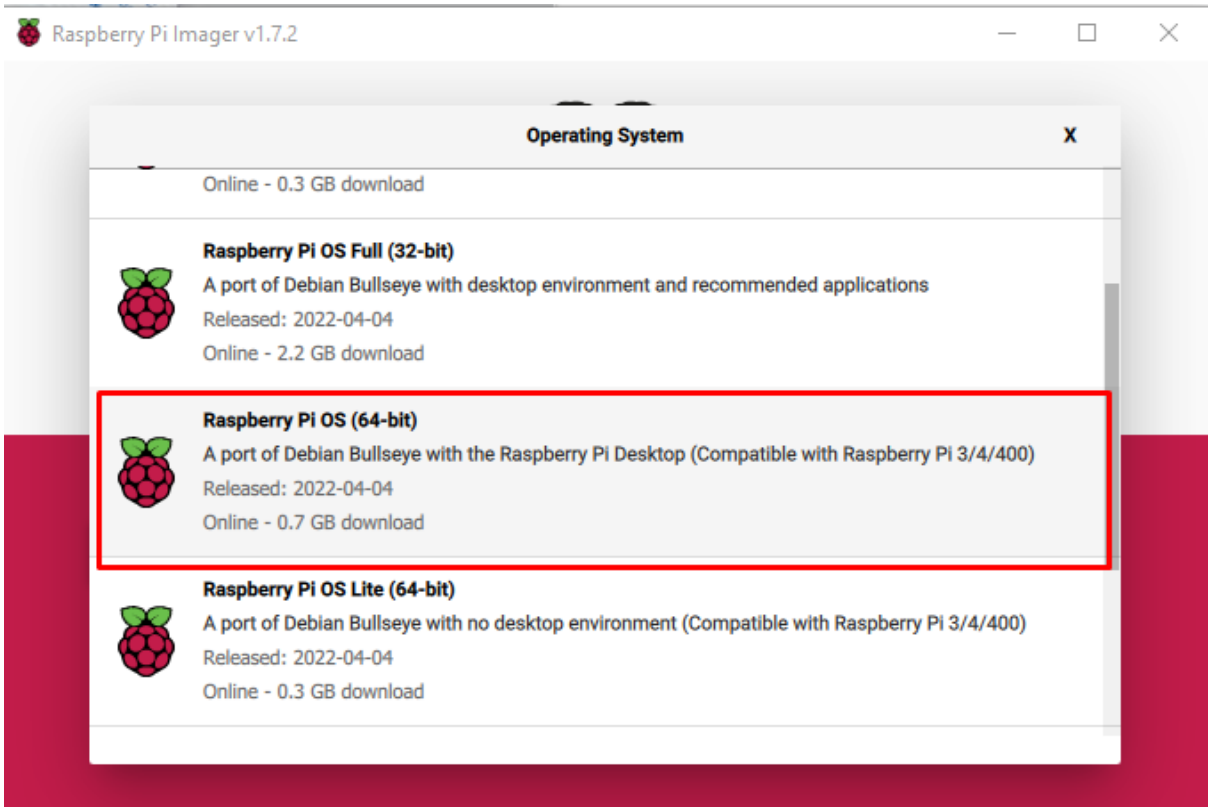
This guide focuses on the installation of TAPPAS on Raspberry Pi OS.

2.6.2. Preparing the Device

- Device: Raspberry Pi Compute Module 4
- Camera: Raspberry Pi Camera V2
- Operating System: Debian GNU/Linux 11 (bullseye)
- Kernel: Linux 5.15.32-v8+
- Architecture: arm64

Burn Debian GNU/Linux 11 (bullseye) Image Download Raspberry Pi Imager





Install Raspberrypi kernel headers

```
sudo apt-get install raspberrypi-kernel-headers
```

2.6.3. TAPPAS Installation

Read through on how to *pre build docker image for Raspberry Pi* or *installing TAPPAS manually*

Note: Raspberry Pi we limit the number of cpu cores to 1 during compilation (more cores accelerates the compilation process, but may cause 'out of swap memory' issue on weak machines like Raspberry Pi)

```
g++-9: fatal error: Killed signal terminated
```

2.6.4. Working with HDMI display

After installing TAPPAS and HailoRT, A known issue regarding working with display connected via HDMI requires the following steps:

1) Add the HailoRT PCIe driver to blacklist:

```
# open rpi blacklist configuration file with sudo privileges
sudo vim /etc/modprobe.d/raspi-blacklist.conf

# add new line to the file
blacklist hailo_pci
```

2) Reboot the machine

3) Install the module manually:


```
sudo modprobe hailo_pci
```

Note: installing the module manually is required on every boot of the raspberry-pi.

4) *Confirm that HailoRT works*

2.6.5. Working with Raspberry PI's MIPI Camera

We tested our applications with Raspberry Pi Camera Module 2

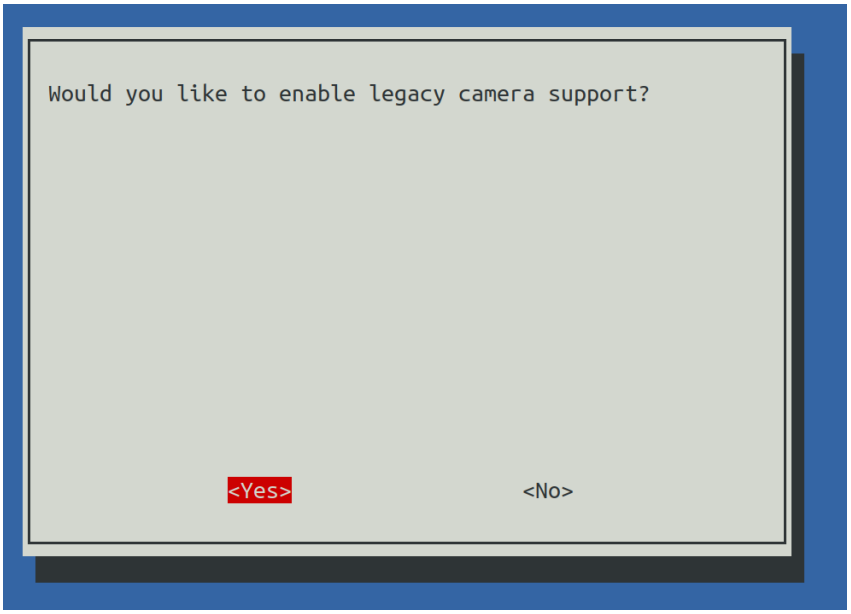
To support TAPPAS apps, enable camera features that support v4l by doing the following steps:

1) Configure Raspicam:

```
sudo wget https://datasheets.raspberrypi.com/cmio/dt-blob-cam1.bin -O /boot/dt-
↪blob.bin
```

2) Enable Legacy camera features:

```
sudo raspi-config
```



3) Reboot the machine

4) Check Raspicam output:

```
vcgencmd get_camera
```

```
pi@raspberrypi:~ $ vcgencmd get_camera
supported=1 detected=1
```

2.6.6. Run TAPPAS Applications

To read further and learn more more details about each application refer to the link *This section*

2.6.7. Troubleshooting

Cannot allocate memory in static TLS block

In some sceneraios, you might face the following error:

```
(gst-plugin-scanner:15): GStreamer-WARNING **: 13:58:20.557: Failed to load plugin '/
↪usr/lib/aarch64-linux-gnu/gstreamer-1.0/libgstlibav.so': /lib/aarch64-linux-
↪gnu/libgomp.so.1: cannot allocate memory in static TLS block
```

The solution is to export an enviroment variable:

```
export LD_PRELOAD=/usr/lib/aarch64-linux-gnu/libgomp.so.1
```


3. GStreamer Based Examples

3.1. General

3.1.1. GStreamer Based General Applications

Where to Begin?

For the starting point Hailo provides a *Sanity Pipeline* that helps verify that the installation phase went well.

Important: Example applications performance varies on different hosts (affected by the host's processing power and throughput).

---

1. *Sanity Pipeline* - Helps you verify that all the required components are installed correctly
2. *Detection* - single-stream object detection pipeline on top of GStreamer using the Hailo-8 device.
3. *Depth Estimation* - single-stream depth estimation pipeline on top of GStreamer using the Hailo-8 device.
4. *Instance segmentation* - single-stream instance segmentation on top of GStreamer using the Hailo-8 device.
5. *Multi-stream detection* - Multi stream object detection (up to 8 RTSP camera into one Hailo-8 chip).
6. *Face Detection and Facial Landmarking Pipeline* - Face detection and then facial landmarking.
7. *Tiling* - Single scale tiling detection application.
8. *Multi-stream Multi-device* - Demonstrate Hailo's capabilities using multiple-chips and multiple-streams.
9. *Python* - Classification app using `resnet_v1_50` with python post-processing.
10. *License Plate Recognition* - LPR app using `yolov5m` vehicle detection, `tiny-yolov4` license plate detection, and `lprnet` OCR extraction with Hailonet network-switch capability.
11. *Multi Person Multi Camera Tracking Pipeline* - Tracking persons across multiple streams.
12. *Century Pipeline* - Detection Pipeline with multiple devices.

3.1.2. License Plate Recognition

Overview

`license_plate_recognition.sh` demonstrates model scheduling between 3 different networks in a complex pipeline with inference based decision making. The overall task is to `detect and track vehicles` in the pipeline and then `detect/extract license plate numbers` from newly tracked instances. When enough newly tracked vehicles are detected (single class yolov5m), a network switch is made to a tiny-yolov4 based network that detects license plates. If enough license plates of good image quality (not blurred) are found, then the network is changed again to a thrid HEF - lprnet license plate text extraction (OCR). Once the license plate is detected and its text extracted, the pipeline updates the `hailotracker` JDE Tracking element upstream with the license plate for the corresponding vehicle. From there the vehicle is tracked along with its license plate number, and is omitted from being re-inferred on new frames. The logic for the network switching is handled by the hailonet elements behind the scenes.

Options

```
./license_plate_recognition.sh [--show-fps]
```

- `--show-fps` is an optional flag that enables printing FPS on screen.

Configuration

The yolo post processes parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/license_plate_recognition/resources/configs

Run

Exporting `TAPPAS_WORKSPACE` environment variable is a must before running the app.

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/license_plate_recognition/license_
↪plate_recognition.sh
```

The output should display as:

Models

- `yolov5m_vehicles`: yolov5m pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_vehicles.yaml
- `tiny_yolov4_license_plates`: tiny_yolov4 pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/tiny_yolov4_license_plates.yaml
- `lprnet`: lprnet pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/lprnet.yaml

Method of Operation

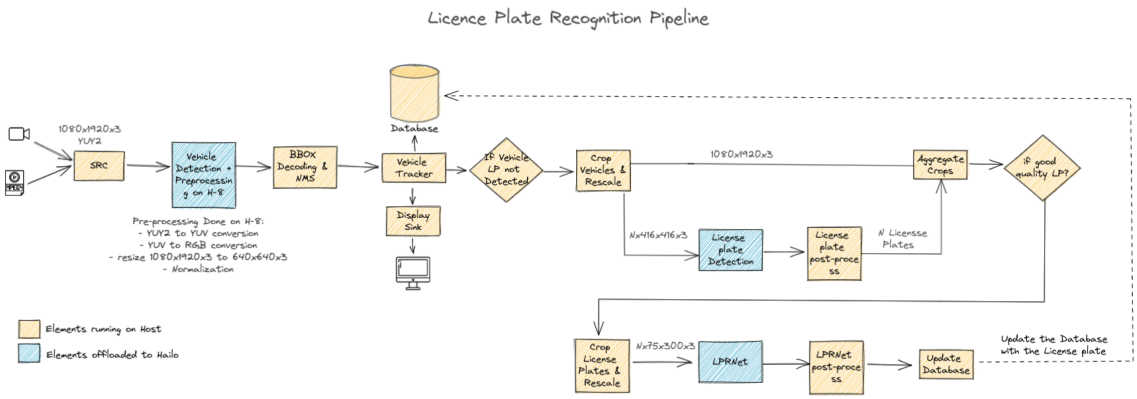
This section explains Network Switching.

The app builds a gstreamer pipeline (described below) and utilizes the `scheduling-algorithm` property of its hailonet elements. This notifies the hailonet elements of a request to switch networks on the same device. The hailonets perform network switching by blocking the sink pads when it is time to switch, turning off one hailonet and turning on the other. Before turning a hailonet element on, it has to flush the buffers out of the element, this is all handled internally. *read more about hailonet*

Pipeline Operation

This section is optional and provides a drill-down into the implementation of the `License Plate Recognition` app with a focus on explaining the `GStreamer` pipeline.

Pipeline Diagram



The following elements form the structure of the pipeline:

- `Model 0` - Vehicle Detection and Tracking
  - `filesrc` reads data from a file in the local file system.
  - `decodebin` constructs a decoding sub-pipeline using available decoders and demuxers
  - `videoconvert` converts the frame into network input format.
  - `hailonet` performs the inference on the Hailo-8 device.
    This instance of hailonet performs yolov5m network inference for vehicle detection.
    read more about hailonet
  - `hailofilter` performs the given post-process, chosen with the `so-path` property. This instance is in charge of yolo post-processing.
  - `hailotracker` performs JDE Tracking using a kalman filter, applying a unique id to tracked vehicles. This element also receives updates of license plate text and associates them to their corresponding tracked vehicle.
    read more about hailotracker
  - `tee` splits the pipeline into two branches. While one buffer continues the drawing and displaying, the other continues to license plate detection and extraction.
  - `hailooverlay` draws the post-process results on the frame.
  - `fpsdisplaysink` outputs video onto the screen, and displays the current and average frame rate.
- `Model 1` - License Plate Detection
  - `hailocropper` crops vehicle detections from the original full HD image and resizes them to the input size of the following `hailonet` (license plate detection). Extra decision making is applied to only pass vehicles that have not had license plate detected and text extracted yet.
    read more about hailocropper
    * `hailonet` this instance of hailonet performs tiny-yolov4 network inference for license plate detection. When initializing the pipeline this instance of hailonet is set to is-active=false.
    * `hailofilter` this instance of hailofilter is in charge of tiny-yolov4 post processing.
  - `hailoaggregator` waits for all crops belonging to the original frame to arrive and merges all metas into their original frame. So, for example, if the upstream `hailocropper` cropped 4 vehicles from the original frame, then this `hailoaggregator` will wait to receive 4 buffers along with the original frame.
    read more about hailoaggregator
- `Model 2` - License Plate Text Extraction (OCR)

  - `hailocropper` another cropping element, this time the decision making is an image quality estimator - if the license plate detection is determined to be too blurry for OCR, then it is dropped. If the detection is not too blurry, then a crop of the license plate is taken from the original full HD image and sent to for OCR inference.
    * `hailonet` this intance of hailonet performs lprnet network inference for license plate text extraction. When initializing the pipeline this instance of hailonet is set to is-active=false.
    * `hailofilter` this instance of hailofilter is in charge of OCR post processing.
  - `hailoaggregator` waits for all crops belonging to the original frame to arrive and merges all metas into their original frame.
  - `hailofilter` captures incoming buffers. From these the ocr text is extracted and sent upstream behind the scenes. These events contain both the OCR postprocess results and the unique tracking id of the vehicle they were extracted from. The event is caught by the `hailotracker` element which updates the corresponding entry in its tracked vehicle database.

Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

You can use Retraining Dockers (available on Hailo Model Zoo), to replace the following models with ones that are trained on your own dataset:

- `yolov5m_vehicles`
  - Retraining docker
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update `configs/yolov5_vehicle_detection.json` with your new post-processing parameters (NMS)
- `tiny_yolov4_license_plates`
  - Retraining docker
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update `configs/yolov4_license_plate.json` with your new post-processing parameters (NMS)
- `lprnet`
  - Retraining docker
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update ocr_postprocess.cpp with your new parameters, then recompile to create `libocr_post.so`

3.1.3. Depth Estimation Pipelines

Depth Estimation

`depth_estimation.sh` demonstrates depth estimation on one video file source. This is done by running a `single-stream object depth estimation pipeline` on top of GStreamer using the Hailo-8 device.

Options

```
./depth_estimation.sh [--video-src FILL-ME]
```

- `-i --input` is an optional flag, a path to the video displayed.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it
- `--show-fps` is an optional flag that enables printing FPS on screen

Run

```
cd /local/workspace/tappas/apps/h8/gstreamer/general/depth_estimation
./depth_estimation.sh
```

The output should look like:

Model

- `fast_depth` in resolution of 224X224X3: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/fast_depth.yaml

How it works

This app is based on our *single network pipeline template*

With two modifications to the template:

1. We use `tee` to show two screens, one with the depth estimation mask applied and one without
2. The network expect no borders, so an `aspectratiocrop` mechanism is needed `aspectratiocrop aspect-ratio=1/1`

3.1.4. Multi-Stream RTSP Object Detection and Pose Estimation Pipeline

Overview

This GStreamer pipeline demonstrates object detection on 8 camera streams over RTSP protocol. This pipeline also demonstrates using two hailo8 devices in parallel.

All the streams are processed in parallel through the decode and scale phases, and enter the Hailo devices frame by frame. **Each** hailo device is in charge of one inference task (one for yolov5 and the other for centerpose)

Afterwards the post-process and drawing phases add the classified object and bounding boxes to each frame. The last step is to match each frame back to its respective stream and output all of them to the display.

Read more about RTSP: *RTSP*

Prerequisites

- TensorPC
- Ubuntu 20.04
- *RTSP*
- Two Hailo-8 devices connected via PCIe

Preparations

1. Before running, configuration of the RTSP camera sources is required. open the `rtsp_detection_and_pose_estimation.sh` in edit mode with your preffered editor. Configure the eight sources to match your own cameras.

```
readonly SRC_0="rtsp://<ip address>/?h264x=4 user-id=<username> user-pw=<password>"
readonly SRC_1="rtsp://<ip address>/?h264x=4 user-id=<username> user-pw=<password>"
etc..
```

Run the Pipeline

```
./rtsp_detection_and_pose_estimation.sh
```

1. `--show-fps` prints the fps to the output.
2. `--num-of-sources` sets the number of rtsp sources to use by given input. the default and recommended value in this pipeline is 8 sources
3. `--print-gst-launch` prints the ready gst-launch command without running it
4. `--print-devices-stats` prints the power and temperature measured
5. `--debug` uses gst-top to print time and memory consuming elements, saves the results as text and graph

   NOTE : When the debug flag is used and the app is running inside of a docker, exit the app by tying `Ctrl+C` in order to save the results. (Due to docker X11 display communication issues)

Models

- `yolov5m_wo_spp_60p` - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml
- `centerpose` - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/centerpose_regnetx_1.6gf_fpn.yaml

Configuration

The yolo post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/multistream_multidevice/resources/configs/yolov5.json

Overview of the Pipeline

These apps are based on our *multi stream pipeline template*

Using Retraining to replace models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained on your own dataset:

- `yolov5m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update `resources/configs/yolov5.json` with the new post-processing parameters (NMS)
- `centerpose`
  - Retraining docker
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - **Update centerpose.cpp** with the new parameters, then recompile to create `libcenterpose_post.so`

3.1.5. Detection Pipeline

Overview

`detection.sh` **demonstrates detection on one video file source and verifies Hailo's configuration.** This is done by running a `single-stream object detection pipeline` on top of GStreamer using the Hailo-8 device.

Options

```
./detection.sh [--input FILL-ME]
```

- `--network` is an optional flag that sets which network to use. Choose from [yolov5, mobilenet_ssd, nanodet, yolov8], default is yolov8. This will set which `hef file` to use, the corresponding `hailofilter` function, and the scaling of the frame to match the width/height input dimensions of the network.
- `--input` is an optional flag, a path to the video displayed (default is detection.mp4).
- `--show-fps` is an optional flag that enables printing FPS on screen.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it.
- `--print-device-stats` prints the power and temperature measured on the Hailo device.

Configuration

In case the selected network is yolo, the app post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/detection/resources/configs

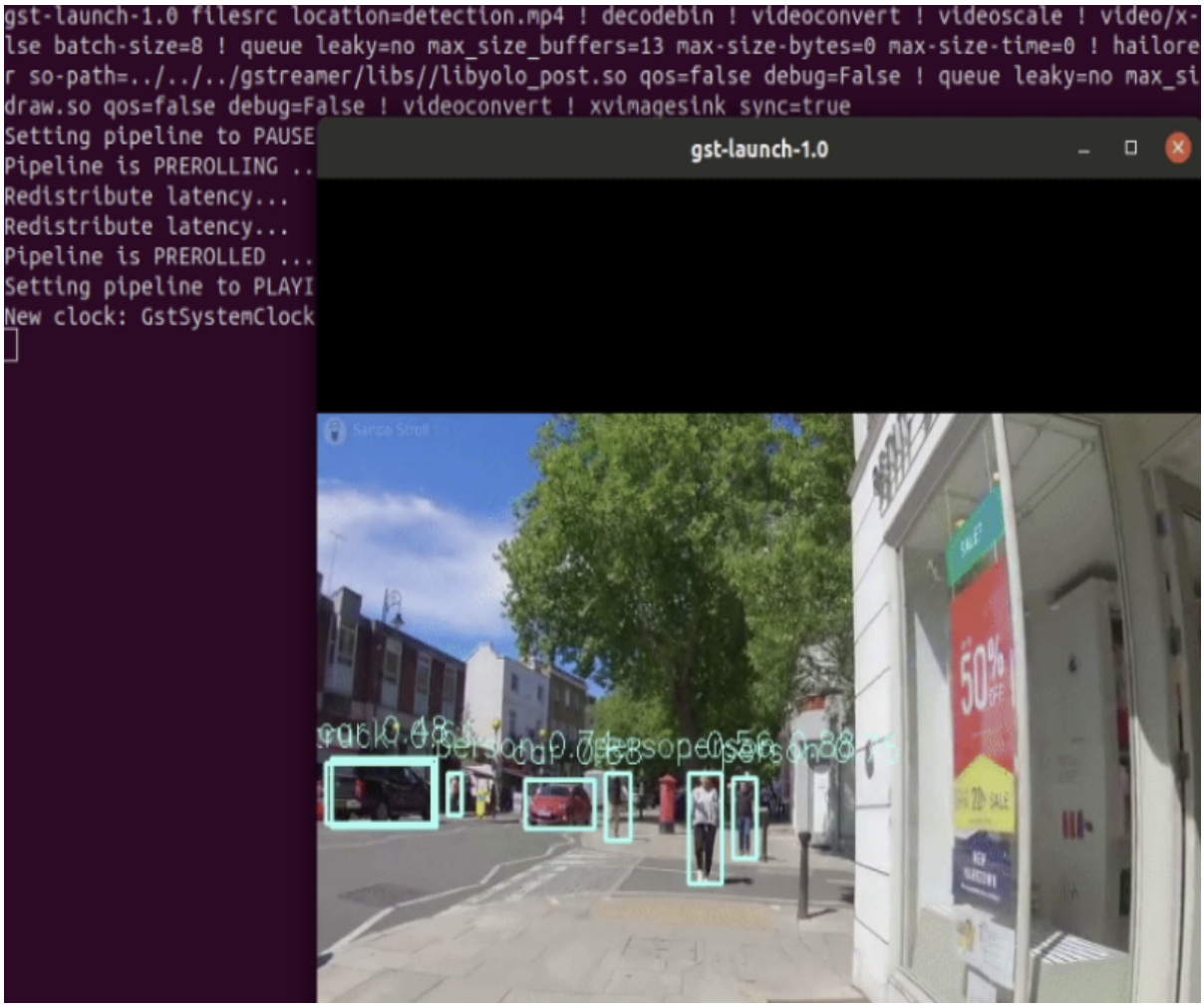
Supported Networks

- 'yolov8m' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov8m.yaml
- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml
- 'mobilenet_ssd' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/ssd_mobilenet_v1.yaml

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/detection
./detection.sh
```

The output should display as:



Method of Operation

This app is based on our *single network pipeline template*

How to Use Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained on the users own dataset:

- `yolov8m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
- `yolov5m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
- `mobilenet_ssd`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update mobilenet_ssd.cpp with your new parameters, then recompile to create `libmobilenet_ssd_post.so`

3.1.6. Sanity Pipeline

Overview

Sanity apps purpose is to help verify that all the required components have been installed successfully.

First of all, `sanity_gstreamer.sh` needs to be run to ensure that the image presented looks like the one that will be presented later.

Sanity GStreamer

This app should launch first.

Note: Open the source code in your preferred editor to see how simple this app is.

In order to run the app just `cd` to the `sanity_pipeline` directory and launch the app

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/sanity_pipeline
./sanity_gstreamer.sh
```

The display should look like the image below:

If the output is similar to the image shown above, continue to the next verification phase.

3.1.7. Cascading Networks

Face Detection and Facial Landmarks

`face_detection_and_landmarks.sh` **demonstrates face detection and facial landmarking on one video file source.**
This is done by running a face detection pipeline (infer + postprocessing), cropping and scaling all detected faces, and sending them into the 2nd network of facial landmarking. All resulting detections and landmarks are then aggregated and drawn on the original frame. The two networks are running using one Hailo-8 device with two `hailonet` elements.

Options

```
./face_detection_and_landmarks.sh [OPTIONS] [-i INPUT_PATH]
```

- `-i --input` is an optional flag, a path to the video/camera displayed.
- `--print-gst-launch` prints the ready gst-launch command without running it
- `--show-fps` optional - enables printing FPS on screen
- `--max-camera-resolution` The maximun input resolution from camera as an input

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/cascading_networks
./face_detection_and_landmarks.sh
```

The output should look like:

Models

- `lightface_slim` in resolution of 320X240X3 - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/lightface_slim.yaml
- `tddfa_mobilenet_v1` in resolution of 120X120X3 - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/tddfa_mobilenet_v1.yaml

Method of Operation

This app is based on our *cascaded networks pipeline template*

Object Detection And Pose Estimation

`object_detection_and_pose_estimation.sh` **demonstrates object detection and pose estimation on one video file**
This is done by running an object detection pipeline, cropping and scaling each detected person, and sending them into a 2nd network of pose estimation. All resulting detections and landmarks are then aggregated and drawn on the original frame. The two networks are running using one Hailo-8 device with two `hailonet` elements.

Options

```
./object_detection_and_pose_estimation.sh [OPTIONS] [-i INPUT_PATH]
```

- `-i --input` is an optional flag, a path to the video/camera displayed.
- `--print-gst-launch` prints the ready gst-launch command without running it
- `--show-fps` optional - enables printing FPS on screen
- `--max-camera-resolution` The maximun input resolution from camera as an input

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/cascading_networks
./object_detection_and_pose_estimation.sh
```

The output should look like:

Model

Joined together:

- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml
- 'mspn_regnetx_800mf' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/mspn_regnetx_800mf.yaml

Method of Operation

This app is based on our *cascaded networks pipeline template*

3.1.8. Python Classification Pipeline

Classification

The purpose of `classification.sh` is to demonstrate classification on one video file source with python post-processing by running a `single-stream object classification pipeline` on top of GStreamer using the Hailo-8 device.

Options

```
./classification.sh [--input FILL-ME]
```

- `--input` is an optional flag, a path to the video displayed (default is classification_movie.mp4).
- `--show-fps` is a flag that prints the pipeline's fps to the screen.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it.

Supported Networks

- 'resnet_v1_50' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/resnet_v1_50.yaml

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/classification
./classification.sh
```

Model

- `resnet_v1_50`: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/resnet_v1_50.yaml

Method of Operation

This app is based on *single network pipeline template* With a slight modification, instead of using `hailofilter` for post-process, `hailopython` is used.

3.1.9. Face Recognition Pipeline

Overview

`face_recognition.sh` demonstrates face recognition pipeline on a video stream.

Options

```
./face_recognition.sh
```

- `-i --input` is an optional flag, a path to an input source (video file / camera device)
- `--print-gst-launch` prints the ready gst-launch command without running it
- `--show-fps` is an optional flag that enables printing FPS on screen
- `--network` to set which network to use. choose from [scrfd_10g, scrfd_2.5g], default is scrfd_10g"

Models

- `scrfd_10g`: scrfd pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/scrfd_10g.yaml
- `scrfd_2.5g`: scrfd pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/scrfd_2.5g.yaml
- `arcface_mobilefacenet`: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/arcface_mobilefacenet.yaml

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/face_recognition/
./face_recognition.sh
```

The output should look like:

Method of Operation

The pipeline is divided to 4 steps:

1. **Face detection and landmarks:** Detect faces and predict the locations of key facial landmarks (such as eyes, nose, and mouth), in the video stream. The scrfd_10g network is more accurate but computationally heavier, while the scrfd_2.5g network is less accurate but more lightweight, providing better performance.
2. **Face alignment:** This step involves using the detected landmarks and the original video frame to compute an affine transformation that aligns the face with a predefined destination matrix. This ensures that the face is consistently positioned in the same way for the next step in the pipeline.
3. **Embedding matrix:** Run Arcface network to generate an embedding matrix for each aligned face. An embedding is a compact representation of the face that captures its unique characteristics. This embedding can then be compared to other embeddings to determine the similarity between faces.

4. **Gallery:** Use the generated embeddings to find the closest matching face in the local database (named "Local Gallery" and stored in a JSON file). This allows the application to identify the person in the video stream.

Saving Faces to the Local Gallery

The local gallery file `face_recognition_local_gallery.json` is stored under `apps/h8/gstreamer/general/face_recognition/resources/gallery` directory. It contains the embeddings of the faces.

To add faces to the gallery, you can use the `save_face.sh` script.

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/face_recognition/
./save_faces.sh
```

Options:

- `--network` to set which network to use. choose from [scrfd_10g, scrfd_2.5g], default is scrfd_10g
- `--clean` to clean the local gallery file enitrely

The script goes over the `.png` files in `resources/faces` directory, and saves each face into the gallery. The name of the face is determined by the file name.

To use your own video sources and faces, add your images to the `resources/faces` directory and remove the original ones. Make sure to use `.png` format image files and a file name including the name of the person. Also use –clean option to order the script to clean the gallery file before saving the new faces.

Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained on your own dataset:

- `scrfd_10g`
  - No retraining docker is available.
  - Post process CPP file edit update post-processing:
    - Update face_detection.cpp (`scrfd()` function) with your new parameters, then recompile to create `libface_detection_post.so`
- `scrfd_2.5g`
  - No retraining docker is available.
  - Post process CPP file edit update post-processing:
    - Update face_detection.cpp (`scrfd()` function) with your new parameters, then recompile to create `libface_detection_post.so`
- `arcface_mobilefacenet`
  - Retraining docker
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update arcface.cpp with your new parameters, then recompile to create `libface_recognition_post.so`

3.1.10. Tiling Pipeline

Single Scale Tiling

Single scale tiling FHD Gstreamer pipeline demonstrates splitting each frame into several tiles which are processed independently by `hailonet` element. This method is especially effective for detecting small objects in high-resolution frames.

Model

- `ssd_mobilenet_v1_visdrone` in resolution of 300X300: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/ssd_mobilenet_v1_visdrone.yaml

The VisDrone dataset consists of only small objects which can be assumed to be always confined within an single tile. As such it is better suited for running single-scale tiling with little overlap and without additional filtering.

Options

```
./tiling.sh [OPTIONS] [-i INPUT_PATH]
```

- `-i --input` is an optional flag, a path to the video file displayed.
- `--print-gst-launch` prints the ready gst-launch command without running it
- `--show-fps` optional - enables printing FPS on screen
- `--tiles-x-axis` optional - set number of tiles along x axis (columns)
- `--tiles-y-axis` optional - set number of tiles along y axis (rows)
- `--overlap-x-axis` optional - set overlap in percentage between tiles along x axis (columns)
- `--overlap-y-axis` optional - set overlap in percentage between tiles along y axis (rows)
- `--iou-threshold` optional - set iou threshold for NMS.
- `--sync-pipeline` optional - set pipeline to sync to video file timing.

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/tiling
./tiling.sh
```

The output should look like:

Method of Operation

This app is based on our *tiling pipeline template*

Multi Scale Tiling

Multi-scale tiling FHD Gstreamer pipeline demonstrates a case where the video and the training dataset includes objects in different sizes. Dividing the frame to small tiles might miss large objects or "cut" them to small objects. The solution is to split each frame into number of scales (layers) each includes several tiles.

Multi-scale tiling strategy also allows us to filter the correct detection over several scales. For example we use 3 sets of tiles at 3 different scales:

- Large scale, one tile to cover the entire frame (1x1)
- Medium scale dividing the frame to 2x2 tiles.
- Small scale dividing the frame to 3x3 tiles.

In this mode we use 1 + 4 + 9 = 14 tiles for each frame. We can simplify the process by highlighting the main tasks: Crop -> Inference -> Ppost-process -> Aggregate Remove exceeded boxes Remove large landscape Perform NMS

Model

- `mobilenet_ssd` in resolution of 300X300X3: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/ssd_mobilenet_v1.yaml

Options

```
./multi_scale_tiling.sh [OPTIONS] [-i INPUT_PATH]
```

- `-i --input` is an optional flag, a path to the video file displayed.
- `--print-gst-launch` prints the ready gst-launch command without running it
- `--show-fps` optional - enables printing FPS on screen
- `--tiles-x-axis` optional - set number of tiles along x axis (columns)
- `--tiles-y-axis` optional - set number of tiles along y axis (rows)
- `--overlap-x-axis` optional - set overlap in percentage between tiles along x axis (columns)
- `--overlap-y-axis` optional - set overlap in percentage between tiles along y axis (rows)
- `--iou-threshold` optional - set iou threshold for NMS.
- `--border-threshold` optional - set border threshold to Remove tile's exceeded objects.
- `--scale-level` optional - set scales (layers of tiles) in addition to the main layer. 1: [(1 X 1)] 2: [(1 X 1), (2 X 2)] 3: [(1 X 1), (2 X 2), (3 X 3)]]'

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/tiling
./multi_scale_tiling.sh
```

The output should look like:

Mode of Operation

As multi scale tiling is almost equal to single scale, the differences are listed below:

```
TILE_CROPPER_ELEMENT="hailotilecropper internal-offset=$internal_offset
↪name=cropper tiling-mode=1 scale-level=$scale_level
```

`hailotilecropper` sets `tiling-mode` to 1 (0 - single-scale, 1 - multi-scale) and `scale-level` to define what is the structure of scales/layers in addition to the main scale.

`hailonet` hef-path is `mobilenet_ssd` which is training dataset includes objects in different sizes.

```
 hailotileaggregator flatten-detections=true iou-threshold=$iou_threshold border-
↪threshold=$border_threshold name=agg

``hailotileaggregator`` sets ``border-threshold`` used in remove tile's exceeded
↪objects process.
```

Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained in the dataset:

- `mobilenet_ssd`
  - Retraining docker
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update mobilenet_ssd.cpp with the new paremeters, then recompile to create `libmobilenet_ssd_post.so`

3.1.11. Century Pipeline

Overview

`century.sh` demonstrates detection on one video file source over multiple Hailo-8 devices, either using the Century platform, or other multi device configurations (i.e., multiple M.2 modules connected directly to the same host). While this application defaults to 4 devices, any number of Hailo-8 devices are supported.

This pipeline runs the detection network Yolov5.

Options

```
./century.sh [--input FILL-ME]
```

- `--input` is an optional flag, a path to the video displayed (default is detection.mp4).
- `--show-fps` is an optional flag that enables printing FPS to the console.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it
- `--device-count` is an optional flag that sets the number of devices to use (default 4)

Configuration

The app post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/century/resources/configs/yolov5.json

Supported Networks

- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml
- 'yolox_l_leaky' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/century
./century.sh
```

The output should look like:

How it works

This app is based on our *multi device pipeline template*.

How to use Retraining to replace models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

You can use Retraining Dockers (available on Hailo Model Zoo), to replace the following models with ones that are trained on your own dataset:

- `yolov5m`
  - Retraining docker
  - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update `resources/configs/yolov5.json` with your new post-processing parameters (NMS)
- `yolox_l_leaky`
  - Retraining docker
  - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update `resources/configs/yolox.json` with your new post-processing parameters (NMS)

3.1.12. Instance Segmentation Pipeline

Overview

`instance_segmentation.sh` **demonstrates instance segmentation on one video file source and verifies Hailo's configura**
This is done by running a `single-stream instance segmentation pipeline` on top of GStreamer using the Hailo-8 device.

Options

```
./instance_segmentation.sh [--input FILL-ME]
```

- `--input` is an optional flag, a path to the video displayed (default is detection.mp4).
- `--show-fps` is an optional flag that enables printing FPS on screen.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it.

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/instance_segmentation
./instance_segmentation.sh
```

The output should display as:

Model

- `yolov5n_seg` - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5n_seg.yaml

Method of Operation

This app is based on the *single network pipeline template*

Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained on your own dataset:

- `yolov5n_seg`
  - No retraining docker is available.
  - Post process CPP file edit update post-processing:
    * Update yolov5seg.cpp with your new parameters, then recompile to create `libyolov5seg_post.so`

3.1.13. Multi-Stream Object Detection Pipeline

Overview

This GStreamer pipeline demonstrates object detection on multiple camera streams over RTSP protocol / Files.

All the streams are processed in parallel through the decode and scale phases, and enter the Hailo device frame by frame.

Afterwards post-process and drawing phases add the classified object and bounding boxes to each frame. The last step is to match each frame back to its respective stream and output all of them to the display.

Read more about RTSP: *RTSP*

Prerequisites

- TensorPC
- In case of using RTSP cameras: *RTSP*
- Hailo-8 device connected via PCIe

Preparations

In case of using RTSP cameras, configuration of the RTSP camera sources is required before running. open the `multi_stream_detection_rtsp.sh` in edit mode with your preferred editor. Configure the eight sources to match your own cameras.

```
readonly SRC_0="rtsp://<ip address>/?h264x=4 user-id=<username> user-pw=<password>"
readonly SRC_1="rtsp://<ip address>/?h264x=4 user-id=<username> user-pw=<password>"
etc..
```

Run the pipeline

```
./multi_stream_detection.sh
```

OR

```
./multi_stream_detection_rtsp.sh
```

1. `--show-fps` Prints the fps to the output.
2. `--debug` Uses gst-top to print time and memory consuming elements, saves the results as text and graph.
3. `--num-of-sources` Sets the number of sources to use by given input. The default and recommended value in this pipeline is 12 (for files) or 8 (for camera streams over RTSP protocol).
4. `--set-live-source` Use the actual source data given (example: /dev/video2). this flag is optional. if it's in use, num_of_sources is limited to 4.
5. `--tcp-address` If specified, set the sink be a TCP (expected format is 'host:port').
6. `--network` If specified, set the network to use. Choose from [yolov5, yolox, yolov8], default is yolov5.
7. `--device-count` If specified, set the number of devices to use. Default (and maximum value) is the minimum between 4 and the number of devices on machine.
8. `--print-gst-launch` Prints the ready gst-launch command without running it.

   NOTE: : When the debug flag is used and the app is running inside of a docker, exit the app by tying `Ctrl+C` in order to save the results. (Due to docker X11 display communication issues)

The output should look like:

Supported Networks

- 'yolov8m' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov8m.yaml
- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml
- 'yolovx_l_leaky' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolovx_l_leaky.yaml

Overview of the Pipeline

These apps are based on our *multi stream pipeline template*

RTSP Specific Elements Used

- `rtspsrc` Makes a connection to an rtsp server and read the data. Used as a src to get the video stream from rtsp-cameras.
- `rtph264depay` Extracts h264 video from rtp packets.

Example of HailoRT Stream Multiplexer

- This application shows the usage of the HailoRT Stream Multiplexer. This feature controls the time shared on the Hailo device between all streams. The Stream Multiplexer is enabled by the `Hailonet` scheduling-algorithm property when in use in multiple `Hailonet` elements that run the same HEF file. When the Stream Multiplexer is in use, there is no need to use `funnel` and `streamiddemux` like elements because the logic is handled internally.

Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained on the users own dataset:

- `yolov8m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
- `yolov5m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
- `yolox_l_leaky`
  - Retraining docker
  - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file

3.1.14. Multi Person Multi Camera Tracking Application

Overview

`multi_person_multi_camera_tracking.sh` demonstrates network switching in a complex pipeline with inference based decision-making. The overall task is to `track people across multiple cameras` in the pipeline with multiple streams. The pipeline is comprised of two general networks: Person/Face Detection, and RE-ID network. The networks run on the Hailo device in parallel where the HailoRT model scheduler handles which network gets running time. When newly tracked persons are detected (single class yolov5s) in, Each where a RE-ID network is run for each person and than a comparison is made across prior persons to RE-ID the person. Once a Person is RE-ID'd, the pipeline updates the `hailotracker` JDE Tracking element upstream with the new global ID for the corresponding person. From there the person is tracked along with its global id, and is re-inferred on new frames.

Options

```
./multi_person_multi_camera_tracking.sh [--show-fps]
```

- `--show-fps` is an optional flag that enables printing FPS on screen.
- `--num-of-sources` sets the number of video sources to use by given input. the default, recommended and maximal value in this pipeline is 4 sources"

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/multi_person_multi_camera_tracking
./multi_person_multi_camera_tracking.sh
```

Expected output:

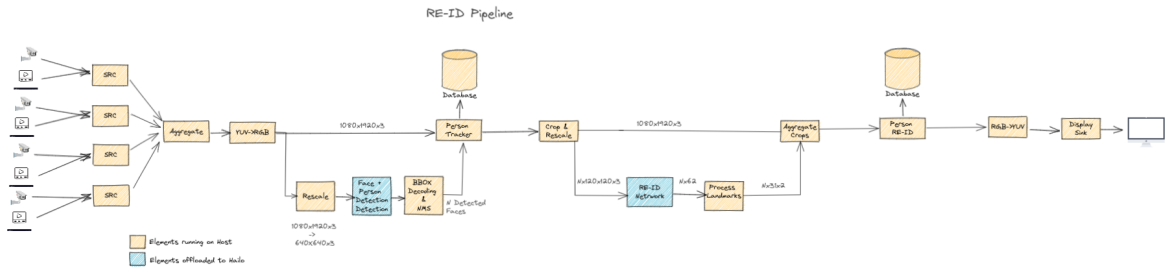
Models

- `yolov5s_personface`: yolov5s pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5s_personface.yaml
- `repvgg_a0_person_reid_2048`: repvgg_a0_person_reid_2048 pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/repvgg_a0_person_reid_2048.yaml

Method of Operation

The app is based on the *Cascaded Networks Structure <../../../../../docs/pipelines/cascaded_nets.rst>* In addition it uses the *HailoTracker <../../../../../docs/elements/hailo_tracker.rst>* and the new *HailoGallery Element <../../../../../docs/elements/hailo_gallery.rst>* gstreamer elements. These elements manage to track the persons over the same stream, and across the multiple streams respectively. Note that we are not using the regular *HailoOverlay <../../../../../docs/elements/hailo_overlay.rst>*, Instead we are using *HailoFilter <../../../../../docs/elements/hailo_filter.rst>* to draw our results since we want to also blur the faces of the persons in the picture and draw different color for each different person.

Pipeline Diagram



Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained in the User's dataset:

- `yolov5s_personface`
  - Retraining docker
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update `resources/configs/yolov5_personface.json` with your new post-processing parameters (NMS)
- `repvgg_a0_person_reid_2048`
  - Retraining docker
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update re_id.cpp with your new parameters, then recompile to create `libre_id.so`

3.2. x86 HW accelerated

3.2.1. GStreamer based x86 applications

Overview

GStreamer has many plugins that support hardware acceleration for a large number of devices via several acceleration packages. The hardware acceleration can be used for decoding/encoding operations, color convert, scaling, compositing etc.

On x86 based devices, one of the main plugins for hardware acceleration is vaapi plugin, that uses the VA-API hw acceleration library.

For more information about VA-API and device compatibility: VAAPI (Video Acceleration API)

The applications under the x86_hw_accelerated directory are using the vaapi gstreamer plugin and dependent on it. In order to verify your device is compatible and supports the x86 hardware acceleration in gstreamer (AKA VA-API), please use the following guide.

Is my machine VA-API compatible?

In order to find out If your machine is compatible we will have to verify:

1. You have an Intel CPU of 8th generation or higher.
2. Your Intel CPU has integrated graphics inside.

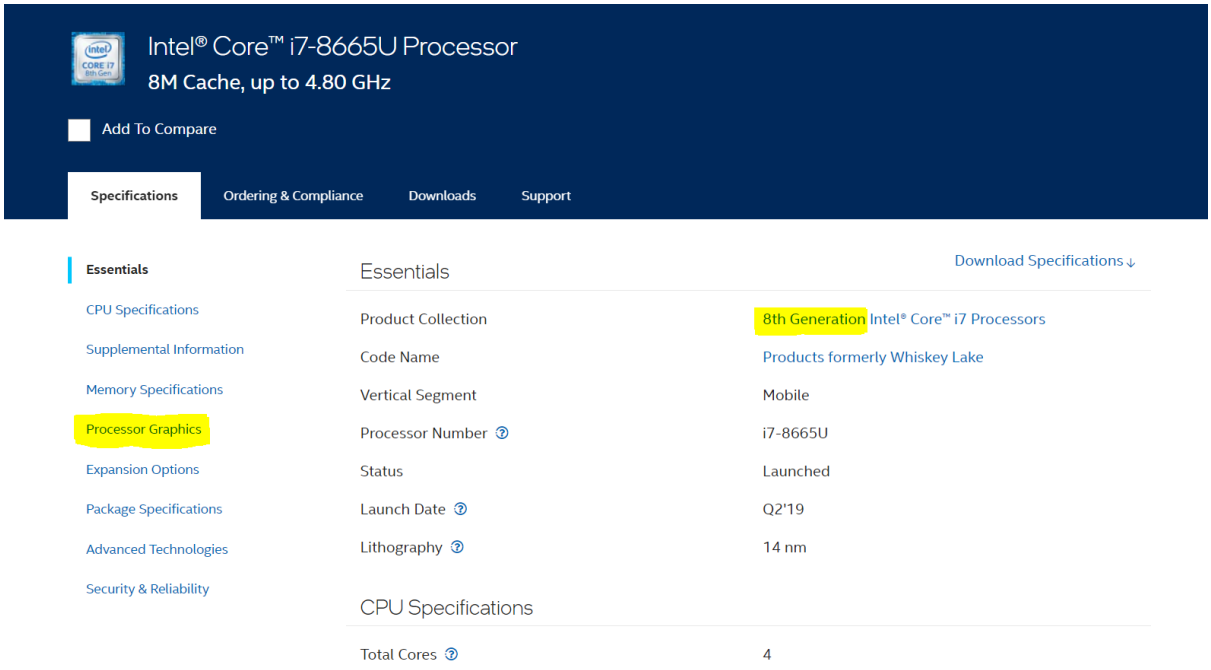
To determine if your CPU has integrated graphics you can follow the following guide

To find out what is the generation of your Intel cpu and also whether it has integrated graphics:

1. Type *lscpu | grep "Model Name:"* on your computer.
2. Take the model name and search for it in the Intel Ark

3. There you can find the generation of the CPU.
4. If the page of the specific processor has a *Processor Graphics* section, it has integrated graphics.

Here is an example of a cpu specifications on Intel Ark:



Highlighted are the *Processor Graphics* section and the CPU's generation.

VA-API Installation Guide

1. Run the VA-API accelerator's insallation via our script: ./install_accelerator.sh
2. Update the environment sourcing the following script: "source $TAPPAS_WORKSPACE/scripts/vaapi/set_env.sh"
3. Hailo provides a *Sanity Pipeline* that helps you verify that the installation phase went well. This is a good starting point.

Supported Pipelines

1. *Sanity Pipeline* - Helps you verify that all the required components are installed correctly
2. *Multi-stream detection* - Multi stream object detection using x86 hw acceleration.
3. *Century Pipeline* - Detection Pipeline with multiple devices.

3.2.2. Sanity pipeline

Overview

Sanity apps purpose is to help you verify that all the required components have been installed successfully. This sanity app helps you to verify that the x86 accelerators are installed correctly.

First of all, you would need to run `sanity.sh` and make sure that the image presented looks like the one that would be presented later.

Sanity GStreamer

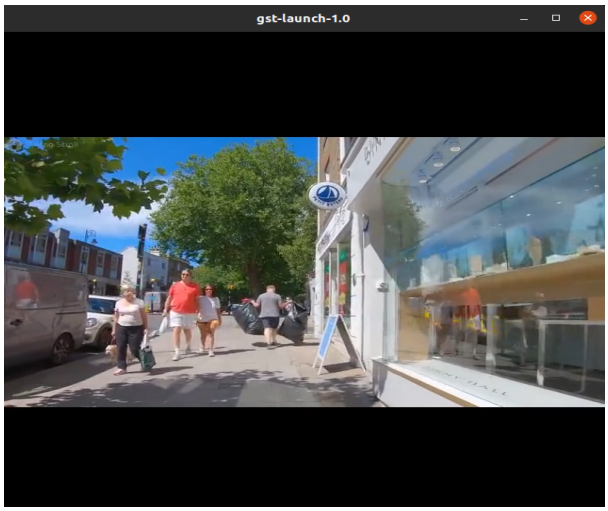
This app should launch first.

Note: Open the source code in your preferred editor to see how simple this app is.

In order to run the app just `cd` to the `sanity` directory and launch the app

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/x86_hw_accelerated/sanity
./sanity.sh
```

The output should look like:



If the output is similar to the image shown above, you are good to go to the next verification phase!

3.2.3. x86 Accelerated Multi-Stream Pipeline

Overview

This GStreamer pipeline demonstrates Person+Face detection on multiple video streams.

All the streams are processed in parallel through the decode and scale phases, and enter the Hailo device frame by frame.

Afterwards postprocess and drawing phases add the classified object and bounding boxes to each frame. The last step is to match each frame back to its respective stream and output all of them to the display.

In this pipeline the decoding phase is accelerated by *VA-API*.

Prerequisites

- RSC101
- Ubuntu 22.04 (or docker with ubuntu 22.04)
- Hailo-8 device connected via PCIe

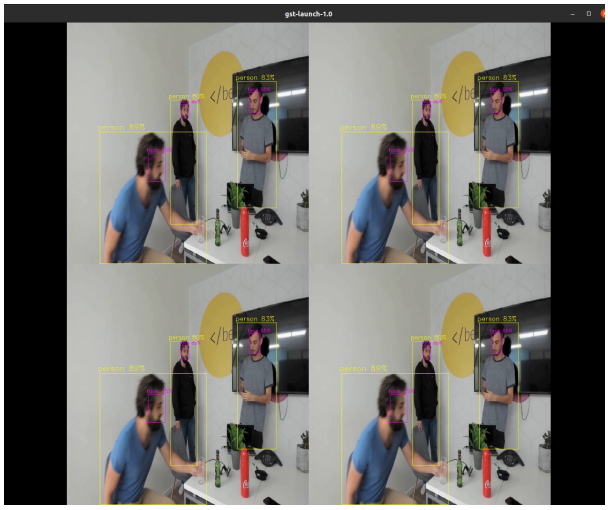
Preparations

Run the pipeline

```
./multistream_detection.sh
```

1. `--show-fps` Prints the fps to the output.
2. `--num-of-sources` Sets the number of sources to use by given input. The default and recommended value in this pipeline is 12 (for files) or 8 (for camera streams over RTSP protocol)"
3. `--network` If specified, set the network to use. Choose from [yolov5, yolox, yolov8], default is yolov5.
4. `--device-count` If specified, set the number of devices to use. Default (and maximum value) is the minimum between 4 and the number of devices on machine.

The output should look like:



Supported Networks

- 'yolov8m' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov8m.yaml
- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml
- 'yolovx_l_leaky' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolovx_l_leaky.yaml

Overview of the pipeline

These apps are based on our *multi stream pipeline template*

Example of HailoRT Stream Multiplexer

- This application shows the usage of the HailoRT Stream Multiplexer. This feature controls the time shared on the Hailo device between all streams. The Stream Multiplexer is enabled by the `Hailonet` scheduling-algorithm property when in use in multiple `Hailonet` elements that run the same HEF file. When the Stream Multiplexer is in use, there is no need to use `funnel` and `streamiddemux` like elements because the logic is handled internally.

Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained on the users own dataset:

- `yolov8m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
- `yolov5m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
- `yolox_l_leaky`
  - Retraining docker
  - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file

3.2.4. x86 Accelerated Century Pipeline

Overview

`century.sh` demonstrates detection on one video file source over multiple Hailo-8 devices, either using the Century platform, or other multi device configurations (i.e., multiple M.2 modules connected directly to the same host). While this application defaults to 4 devices, any number of Hailo-8 devices are supported.
This pipeline runs the detection network Yolov5.

In this pipeline the decoding phase is accelerated by *VA-API*.

Options

```
./century.sh [--input FILL-ME]
```

- `--input` is an optional flag, a path to the video displayed (default is detection.mp4).
- `--show-fps` is an optional flag that enables printing FPS to the console.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it
- `--device-count` is an optional flag that sets the number of devices to use (default 4)

Configuration

The app post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/century/resources/configs/yolov5.json

Supported Networks

- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/century
./century.sh
```

The output should look like:

How the application works

This app is based on our *multi device pipeline template*.

How to use Retraining to replace models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

You can use Retraining Dockers (available on Hailo Model Zoo), to replace the following models with ones that are trained on your own dataset:

- `yolov5m`
  - Retraining docker
  - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update `resources/configs/yolov5.json` with your new post-processing parameters (NMS)

3.3. i.MX8

3.3.1. iMX8 GStreamer Based Applications

Warning: The Kirkstone applications portfolio is reduced on i.MX8-based devices, since the Kirkstone branch does not support OpenGL.

1. *Detection* - single-stream object detection pipeline on top of GStreamer using the Hailo-8 device.
2. *Depth Estimation* - single-stream depth estimation pipeline on top of GStreamer using the Hailo-8 device.
3. *Face Detection and Facial Landmarking Pipeline* - Face detection and then facial landmarking.
4. *License Plate Recognition* - LPR app using `yolov5m` vehicle detection, `tiny-yolov4` license plate detection, and `lprnet` OCR extraction with Hailonet network-switch capability.

3.3.2. Face Detection and Facial Landmarks Pipeline

Face Detection and Facial Landmarks

`face_detection_and_landmarks.sh` **demonstrates face detection and facial landmarking on one video file source.**
This is done by running a face detection pipeline (infer + postprocessing), cropping and scaling all detected faces, and sending them into the 2nd network of facial landmarking. All resulting detections and landmarks are then aggregated and drawn on the original frame. The two networks are running using one Hailo-8 device with two `hailonet` elements.

Options

```
./face_detection_and_landmarks.sh [OPTIONS] [-i INPUT_PATH]
```

- `-i --input` is an optional flag, a path to the video/camera displayed.
- `--print-gst-launch` prints the ready gst-launch command without running it
- `--show-fps` optional - enables printing FPS on screen

Run

```
cd /home/root/apps/cascading_networks
./face_detection_and_landmarks.sh
```

The output should look like:

Models

- `lightface_slim` in resolution of 320X240X3 - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/lightface_slim.yaml
- `tddfa_mobilenet_v1` in resolution of 120X120X3 - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/tddfa_mobilenet_v1.yaml

Method of Operation

This app is based on our *cascaded networks pipeline template*

Object Detection And Pose Estimation

`object_detection_and_pose_estimation.sh` **demonstrates object detection and pose estimation on one video file**
This is done by running an object detection pipeline, cropping and scaling each detected person, and sending them into a 2nd network of pose estimation. All resulting detections and landmarks are then aggregated and drawn on the original frame. The two networks are running using one Hailo-8 device with two `hailonet` elements.

Options

```
./object_detection_and_pose_estimation.sh [OPTIONS] [-i INPUT_PATH]
```

- `-i --input` is an optional flag, a path to the video/camera displayed.
- `--print-gst-launch` prints the ready gst-launch command without running it
- `--show-fps` optional - enables printing FPS on screen

Run

```
cd $TAPPAS_WORKSPACE/apps/gstreamer/general/cascading_networks
./object_detection_and_pose_estimation.sh
```

The output should look like:

Model

Joined together:

- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml
- 'mspn_regnetx_800mf' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/mspn_regnetx_800mf.yaml

Method of Operation

This app is based on our *cascaded networks pipeline template*

3.3.3. Depth Estimation Pipelines

Depth Estimation

`depth_estimation.sh` demonstrates depth estimation on one video file source. This is done by running a `single-stream object depth estimation pipeline` on top of GStreamer using the Hailo-8 device.

Options

```
./depth_estimation.sh [--video-src FILL-ME]
```

- `-i --input` is an optional flag, a path to the video displayed.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it
- `--show-fps` is an optional flag that enables printing FPS on screen

Run

```
cd /home/root/apps/depth_estimation
./depth_estimation.sh
```

The output should look like:

Model

- `fast_depth` in resolution of 224X224X3: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/fast_depth.yaml

How it works

This app is based on our *single network pipeline template*

With one modification to the template:

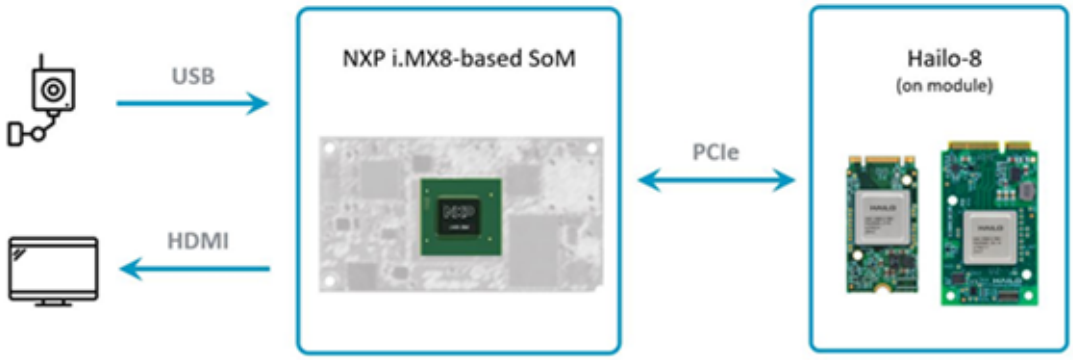
1. The network expect no borders, so an `aspectratiocrop` mechanism is needed `aspectratiocrop aspect-ratio=1/1`

3.3.4. Detection Pipeline

Overview

Our requirement from this pipeline is a real-time high-accuracy object detection to run on a single video stream using an embedded host. The required input video resolution was HD (high definition, 720p).

The chosen platform for this project is based on NXP's i.MX8M ARM processor. The Hailo-8TM AI processor is connected to it as an AI accelerator.



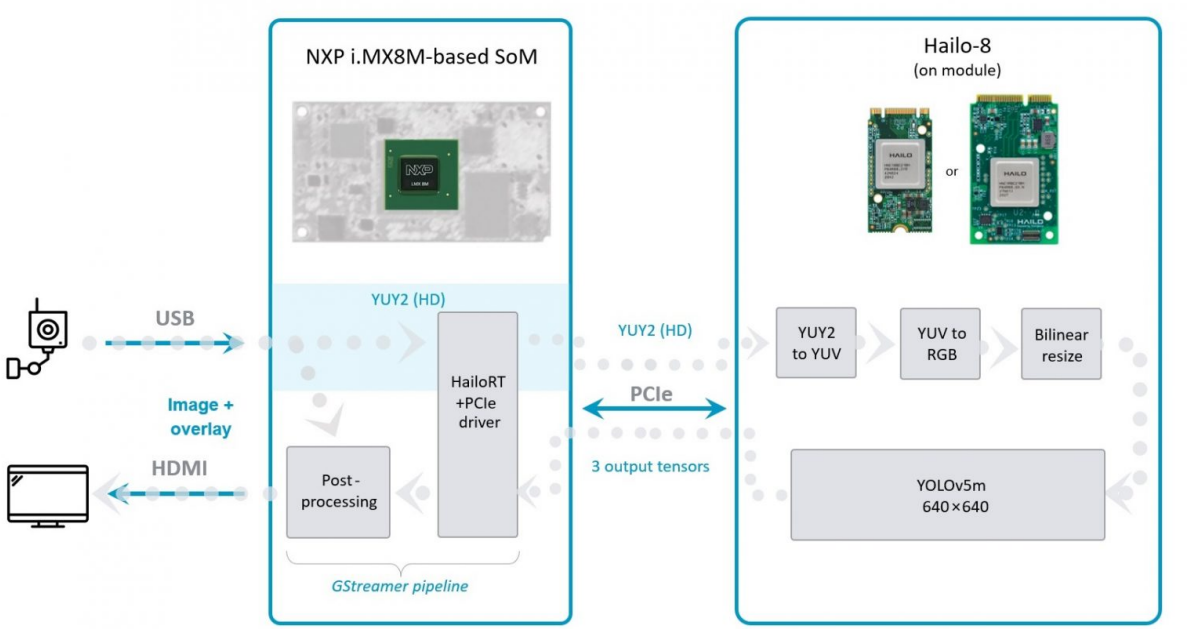
Drill Down

Although the i.MX8M is a capable host, processing and decoding real-time HD video is bound to utilize a lot of the CPU's resources, which may eventually reduce performance. To solve this problem, most of the vision pre-processing pipeline has been offloaded to the Hailo-8 device in our application.

The camera sends the raw video stream, encoded in YUV color format using the YUY2 layout. The data passes through Hailo's runtime software library, called HailoRT, and through Hailo's PCIe driver. The data's format is kept unmodified, and it is sent to the Hailo-8 device as is.

Hailo-8's NN core handles the data preprocessing, which includes decoding the YUY2 scheme, converting from the YUV color space to RGB and, finally, resizing the frames into the resolution expected by the deep learning detection model.

The Hailo Dataflow Compiler supports adding these pre-processing stages to any model when compiling it. In this case, they are added before the YOLOv5m detection model.



Options

```
./detection.sh [--input FILL-ME]
```

- `--input` is an optional flag, path to the video camera used (default is /dev/video2).
- `--show-fps` is an optional flag that enables printing FPS on screen.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it"

Configuration

The app post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/imx8/detection/resources/configs/yolov5.json

Run

```
./detection.sh
```

The output should look like:

Model

- 'yolov5m_wo_spp_yuv' with color convert and resize: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_yuv.yaml

How it works

This app is based on our *single network pipeline template*

Links

Blog post about this setup

How to use Retraining to replace models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

You can use Retraining Dockers (available on Hailo Model Zoo), to replace the following models with ones that are trained on your own dataset:

- `yolov5m_wo_spp_yuv`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use `yolov5m_wo_spp_yuv.yaml` for the compilation
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update `resources/configs/yolov5.json` with your new post-processing parameters (NMS)

3.3.5. License Plate Recognition

Overview

`license_plate_recognition.sh` demonstrates model scheduling in a complex pipeline with inference based decision making. The overall task is to `detect and track vehicles` in the pipeline and then `detect/extract license plate numbers` from newly tracked instances.

Configuration

The yolo post processes parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/license_plate_recognition/resources/configs

Configuration

The yolo post processes parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/general/license_plate_recognition/resources/configs

Run

```
./apps/license_plate_recognition/license_plate_recognition.sh
```

The output should look like:

Models

- `yolov5m_vehicles_yuy2`: yolov5m pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_vehicles_yuy2.yaml
- `tiny_yolov4_license_plates_yuy2`: tiny_yolov4 pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/tiny_yolov4_license_plates_yuy2.yaml
- `lprnet_yuy2`: lprnet pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/lprnet_yuy2.yaml

How the application works

This app uses HailoRT Model Scheduler, read more about HailoRT Model Scheduler GStreamer integration at *HailoNet*

Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

You can use Retraining Dockers (available on Hailo Model Zoo), to replace the following models with ones that are trained on your own dataset:

- `yolov5m_vehicles`
  - Retraining docker
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file

        * Update `configs/yolov5_vehicle_detection.json` with your new post-processing parameters (NMS)

- `tiny_yolov4_license_plates`
    - Retraining docker
    - TAPPAS changes to replace model:
        * Update HEF_PATH on the .sh file
        * Update `configs/yolov4_license_plate.json` with your new post-processing parameters (NMS)
- `lprnet`
    - Retraining docker
    - TAPPAS changes to replace model:
        * Update HEF_PATH on the .sh file
        * Update ocr_postprocess.cpp with your new parameters, then recompile to create `libocr_post.so`

3.4. Raspberry Pi

3.4.1. Raspberry Pi GStreamer based applications

Where should I start?

That's a great question! Hailo provides a *Sanity Pipeline* that helps you verify that the installation phase went well. This is a good starting point.

1. *Sanity Pipeline* - Helps you verify that all the required components are installed correctly
2. *Detection* - single-stream object detection pipeline on top of GStreamer using the Hailo-8 device.
3. *Depth Estimation* - single-stream depth estimation pipeline on top of GStreamer using the Hailo-8 device.

3.4.2. Depth Estimation Pipelines

Depth Estimation

`depth_estimation.sh` demonstrates depth estimation on one video file source. This is done by running a `single-stream object depth estimation pipeline` on top of GStreamer using the Hailo-8 device.

Options

```
./depth_estimation.sh [--video-src FILL-ME]
```

- `-i --input` is an optional flag, a path to the video displayed.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it
- `--show-fps` is an optional flag that enables printing FPS on screen

Run

```
cd /local/workspace/tappas/apps/h8/gstreamer/raspberrypi/depth_estimation
./depth_estimation.sh
```

The output should look like:

Model

- `fast_depth` in resolution of 224X224X3: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/fast_depth.yaml

How it works

This app is based on our *single network pipeline template*

With few modifications to the template:

1. We use `tee` to show two screens, one with the depth estimation mask applied and one without
2. The network expect no borders, so an `aspectratiocrop` mechanism is needed `aspectratiocrop aspect-ratio=1/1`
3. Use decode elements instead of `decodebin`
4. Increase the number of threads on the `videoconvert`

3.4.3. Detection Pipeline

Overview

`detection.sh` **demonstrates detection on one video file source and verifies Hailo's configuration.** This is done by running a `single-stream object detection pipeline` on top of GStreamer using the Hailo-8 device.

Options

```
./detection.sh [--input FILL-ME]
```

- `--input` is an optional flag, a path to the video file displayed (default is detection.mp4).
- `--network` is a flag that sets which network to use. choose from [yolov5, mobilenet_ssd], default is yolov5. this will set the hef file to use, the `hailofilter` function to use and the scales of the frame to match the width and heigh input dimensions of the network.
- `--show-fps` is an optional flag that enables printing FPS on screen.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it
- `--print-device-stats` Print the power and temperature measured

Configuration

In case the selected network is yolo, the app post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/raspberrypi/detection/resources/configs/yolov5.json

Supported Networks

- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml
- 'mobilenet_ssd' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/ssd_mobilenet_v1.yaml

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/raspberrypi/detection
./detection.sh
```

The output should look like:

How it works

This app is based on our *single network pipeline template*

With small modifications:

1. Use decode elements instead of `decodebin`
2. Increase the number of threads on the `videoconvert`

How to use Retraining to replace models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

You can use Retraining Dockers (available on Hailo Model Zoo), to replace the following models with ones that are trained on your own dataset:

- `yolov5m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update `resources/configs/yolov5.json` with your new post-processing parameters (NMS)
  - TAPPAS changes to replace model:
    - Update HEF_PATH on the .sh file
    - Update `resources/configs/yolov5.json` with your new post-processing parameters (NMS)
- `mobilenet_ssd`
  - Retraining docker

- TAPPAS changes to replace model:
  * Update HEF_PATH on the .sh file
  * Update mobilenet_ssd.cpp with your new parameters, then recompile to create `libmobilenet_ssd_post.so`

3.4.4. Sanity pipeline

Overview

Sanity apps purpose is to help you verify that all the required components have been installed successfully.

First of all, you would need to run `sanity_gstreamer.sh` and make sure that the image presented looks like the one that would be presented later.

Sanity GStreamer

This app should launch first.

Note: Open the source code in your preferred editor to see how simple this app is.

In order to run the app just `cd` to the `sanity_pipeline` directory and launch the app

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/raspberrypi/sanity_pipeline
./sanity_gstreamer.sh
```

The output should look like:

If the output is similar to the image shown above, you are good to go to the next verification phase!

3.5. Rockchip

3.5.1. RockChip GStreamer based applications

1. *Detection* - single-stream object detection pipeline on top of GStreamer using the Hailo-8 device.
2. *License Plate Recognition* - LPR app using `yolov5m` vehicle detection, `tiny-yolov4` license plate detection, and `lprnet` OCR extraction with Hailonet network-switch capability.
3. *Multistream Detection* - multistream detection with multi stream out using rockchip elements
4. *Tiling* - Single scale tiling detection application.

3.5.2. Detection Pipeline

Overview

`detection.sh` demonstrates detection on one video file source and verifies Hailo's configuration, by running a `single-stream object detection pipeline` on top of GStreamer using the Hailo-8 device.

Options

```
./detection.sh [--input FILL-ME]
```

- `--network` is an optional flag that sets which network to use. Choose from [yolov5, nanodet], default is yolov5. This will set which `hef file` to use, the corresponding `hailofilter` function, and the scaling of the frame to match the width/height input dimensions of the network.
- `--input` is an optional flag, a path to the video displayed (default is detection.mp4).
- `--show-fps` is an optional flag that enables printing FPS on screen.
- `--print-gst-launch` is a flag that prints the ready gst-launch command without running it.
- `--print-device-stats` Print the power and temperature measured on the Hailo device.

Configuration

In case the selected network is yolo, the app post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/rockchip/detection/resources/configs

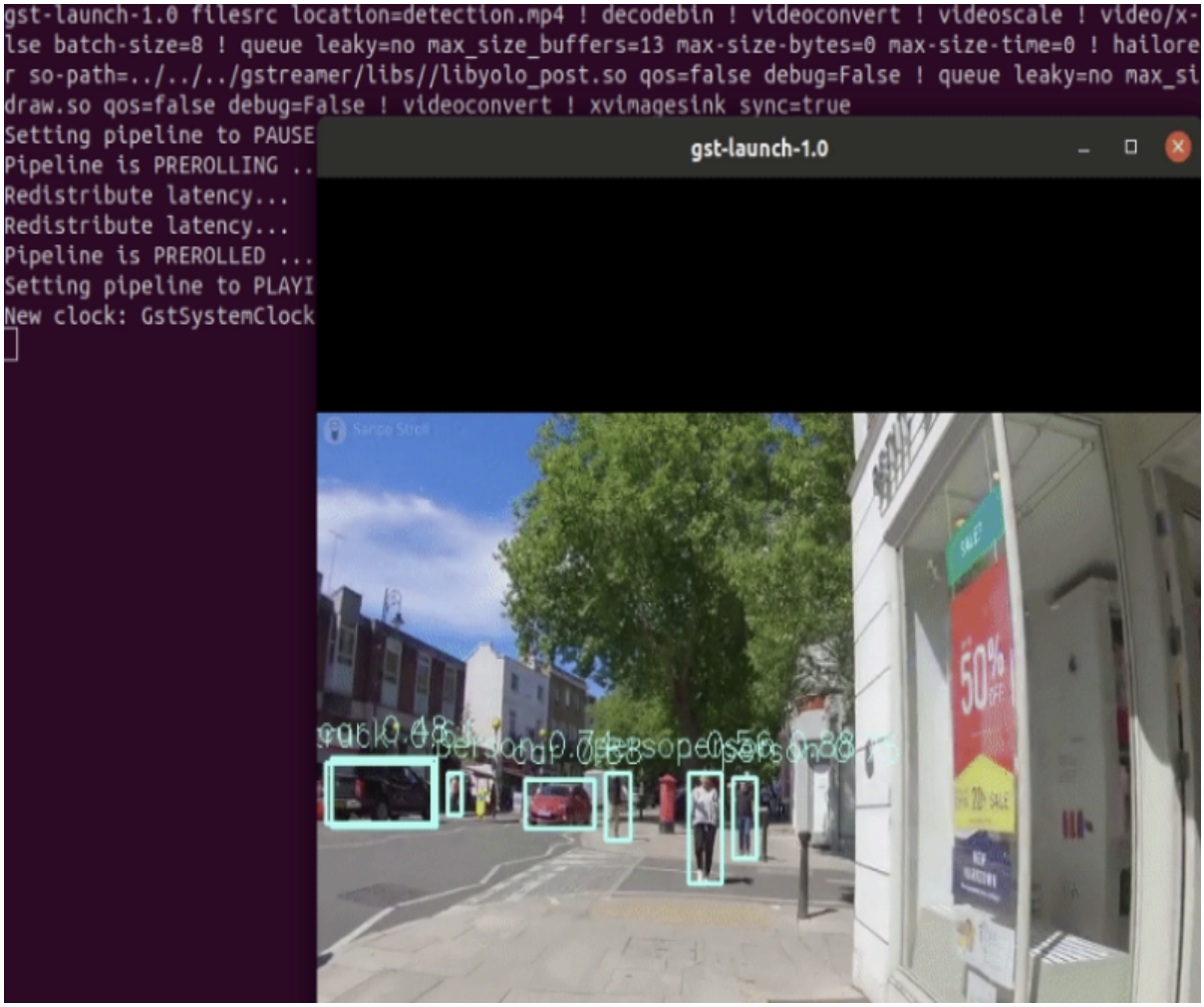
Supported Networks

- 'yolov5m_wo_spp_60p' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_wo_spp_60p.yaml

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/rockchip/detection
./detection.sh
```

The output should look like:



Method of Operation

This app is based on the *single network pipeline template*

Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained in the dataset:

- `yolov5m`
  - Retraining docker
    - For optimum compatibility and performance with TAPPAS, use for compilation the corresponding YAML file from above.

- TAPPAS changes to replace model:
  * Update HEF_PATH on the .sh file
  * Update `resources/configs/yolov5.json` with your new post-processing parameters (NMS)

3.5.3. License Plate Recognition

Overview

`license_plate_recognition.sh` demonstrates model scheduling between 3 different networks in a complex pipeline with inference based decision making. The overall task is to `detect and track vehicles` in the pipeline and then `detect/extract license plate numbers` from newly tracked instances. When enough newly tracked vehicles are detected (single class yolov5m), a network switch is made to a tiny-yolov4 based network that detects license plates. If enough license plates of good image quality (not blurred) are found, then the network is changed again to a thrid HEF - lprnet license plate text extraction (OCR). Once the license plate is detected and its text extracted, the pipeline updates the `hailotracker` JDE Tracking element upstream with the license plate for the corresponding vehicle. From there the vehicle is tracked along with its license plate number, and is omitted from being re-inferred on new frames. The logic for the network switching is handled by the hailonet elements behind the scenes.

Options

```
./license_plate_recognition.sh [--show-fps]
```

- `--show-fps` is an optional flag that enables printing FPS on screen.

Configuration

The yolo post processes parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/rockchip/license_plate_recognition/resources/configs

Run

Exporting `TAPPAS_WORKSPACE` environment variable is a must before running the app.

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/rockchip/license_plate_recognition/
↪license_plate_recognition.sh
```

The output should look like:

Models

- `yolov5m_vehicles`: yolov5m pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5m_vehicles.yaml
- `tiny_yolov4_license_plates`: tiny_yolov4 pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/tiny_yolov4_license_plates.yaml
- `lprnet`: lprnet pre-trained on Hailo's dataset - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/lprnet.yaml

Application Method of Operation

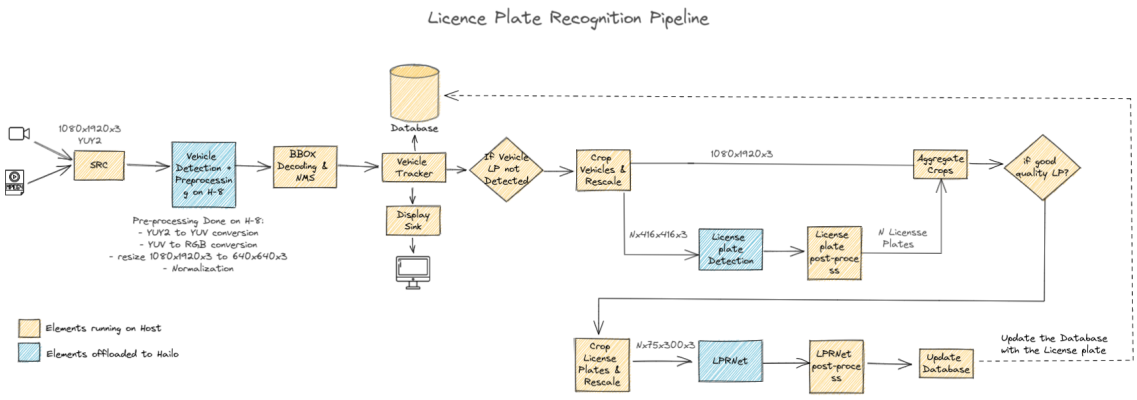
This section explains the network switching.

The app builds a gstreamer pipeline (that is explained below) and utilizes the `scheduling-algorithm` property of its hailonet elements. This lets the hailonet elements know that we wish to switch networks on the same device. The hailonets perform network switching by blocking their sink pads when it is time to switch: turning off one hailonet and turning on the other. Before turning a hailonet element on, it has to flush the buffers out of the element, this is all handled internally. *read more about hailonet*

Pipeline Method of Operation

This section is optional and provides a drill-down into the implementation of the `License Plate Recognition` app with a focus on explaining the `GStreamer` pipeline.

Pipeline Diagram



The following elements are the structure of the pipeline:

- `Model 0` - Vehicle Detection and Tracking
  - `filesrc` reads data from a file in the local file system.
  - `decodebin` constructs a decoding sub-pipeline using available decoders and demuxers
  - `videoconvert` converts the frame into network input format.
  - `hailonet` Performs the inference on the Hailo-8 device.
    This intance of hailonet performs yolov5m network inference for vehicle detection.
    read more about hailonet
  - `hailofilter` performs the given postprocess, chosen with the `so-path` property. This instance is in charge of yolo post processing.
  - `hailotracker` performs JDE Tracking using a kalman filter, applying a unique id to tracked vehicles. This element also receives updates of license plate text and associates them to their corresponding tracked vehicle.
    read more about hailotracker
  - `tee` splits the pipeline into two branches. While one buffer continues the drawing and displaying, the other continues to license plate detection and extraction.
  - `hailooverlay` draws the postprocess results on the frame.
  - `fpsdisplaysink` outputs video onto the screen, and displays the current and average frame rate.
- `Model 1` - License Plate Detection

  - `hailocropper` crops vehicle detections from the original full HD image and resizes them to the input size of the following `hailonet` (license plate detection). Extra decision making is applied to only pass vehicles that have not had license plate detected and text extracted yet.

    read more about hailocropper

    * `hailonet` this intance of hailonet performs tiny-yolov4 network inference for license plate detection. When initializing the pipeline this instance of hailonet is set to is-active=false.
    * `hailofilter` this instance of hailofilter is in charge of tiny-yolov4 post processing.
  - `hailoaggregator` waits for all crops belonging to the original frame to arrive and merges all metas into their original frame. So, for example, if the upstream `hailocropper` cropped 4 vehicles from the original frame, then this `hailoaggregator` will wait to receive 4 buffers along with the original frame.

    read more about hailoaggregator

- `Model 2` - License Plate Text Extraction (OCR)
  - `hailocropper` another cropping element, this time the decision making is an image quality estimator - if the license plate detection is determined to be too blurry for OCR, then it is dropped. If the detection is not too blurry, then a crop of the license plate is taken from the original full HD image and sent to for OCR inference.
    * `hailonet` this intance of hailonet performs lprnet network inference for license plate text extraction. When initializing the pipeline this instance of hailonet is set to is-active=false.
    * `hailofilter` this instance of hailofilter is in charge of OCR post processing.
  - `hailoaggregator` waits for all crops belonging to the original frame to arrive and merges all metas into their original frame.
  - `hailofilter` captures incoming buffers. From these the ocr text is extracted and sent upstream behind the scenes. These events contain both the OCR postprocess results and the unique tracking id of the vehicle they were extracted from. The event is caught by the `hailotracker` element which updates the corresponding entry in its tracked vehicle database.

Using Retraining to Replace Models

Note: It is recommended to first read the *Retraining TAPPAS Models* page.

Retraining Dockers (available on Hailo Model Zoo), can be used to replace the following models with ones that are trained in the dataset:

- `yolov5m_vehicles`
  - Retraining docker
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update `configs/yolov5_vehicle_detection.json` with the new post-processing parameters (NMS)
- `tiny_yolov4_license_plates`
  - Retraining docker
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update `configs/yolov4_license_plate.json` with the new post-processing parameters (NMS)

- `lprnet`
  - Retraining docker
  - TAPPAS changes to replace model:
    * Update HEF_PATH on the .sh file
    * Update ocr_postprocess.cpp with the new parameters, then recompile to create `libocr_post.so`

3.5.4. Multi-Stream Object Detection Pipeline

Overview

This GStreamer pipeline demonstrates object detection on multiple camera streams over RTSP protocol / files. All the streams are processed in parallel via the hardware accelerated decoder, and after scaling, the frames are sent to the Hailo chip frame by frame.

After postprocess and optional drawing phases, the classified object and bounding boxes are added to each frame. Each frame is then separated by the input stream, the Hailo meta data is saved to a .json file, the image stream is encoded and sent to a remote host. On the remote host, the RTP server receives the data from each stream.

This pipeline uses the Rockchip mpp elements for hardware-accelerated decoding and encoding capabilities.

Read more about RTSP: *RTSP* Read more about MPP: *MPP*

Prerequisites

- Firefly ITX-3588J
- In cases where RTSP cameras are used: *RTSP*
- Hailo-8 device connected via PCIe

Preparations

Before running the pipeline, configuration of the RTSP camera sources are required. Open the `multi_stream_detection_rtsp.sh` in edit mode with your preferred editor. Configure the eight sources to match your own cameras.

```
readonly SRC_0="rtsp://<ip address>/?h264x=4 user-id=<username> user-pw=<password>"
readonly SRC_1="rtsp://<ip address>/?h264x=4 user-id=<username> user-pw=<password>"
etc..
```

Run the Pipeline

On the remote machine is recommended to run first the receiving script:

```
./rtp-src_streams.sh
```

1. `--host-ip` Change the host to where to get the rtp stream.
2. `--port` Set the port to listen for the rtp stream."
3. `--caps` A string that defines the input stream capabilities, used by the pipeline for decode and display.

Then on the Rockchip run the next script:

```
./multi_stream_detection.sh
```


OR

```
./multi_stream_detection_rtsp.sh
```

1. `--num-of-sources` Sets the number of sources to use by given input. The default and recommended value in this pipeline is 10 "
2. `--use-overlay` If set the pipeline will print the bounding boxes after post processing, it may affect the application performance on streams which have a large amount of detections.
3. `--udp-sink-ip` Sets the ip of the remote host to send the video streams after processing.

The output should look like:

Configuration

The app post process parameters can be configured by a json file located in $TAPPAS_WORKSPACE/apps/h8/gstreamer/rockchip/multistream_detection/resources/configs/yolov5.json

Supported Networks

- 'yolov5s_nv12' - https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/yolov5s_nv12.yaml

Overview of the Pipeline

These apps are based on our *multi stream pipeline template*

Specific RTSP Elements Used

- `rtspsrc` Makes a connection to an rtsp server and read the data. Used as a src to get the video stream from rtsp-cameras.
- `rtph264depay` Extracts h264 video from rtp packets.
- `rtph264pay` Insert the h264 video to rtp packets.
- `udpsink` Sends packets using the UDP protocol.

MPP specific elements used

- `mppvideodec` Decodes the input h264 encoded stream using the MPP to allow for HW acceleration.
- `mpph264enc` Encodes the input video stream in to h264 video stream using the MPP to allow for HW acceleration.

3.5.5. Tiling Pipeline

Single Scale Tiling

Single scale tiling FHD Gstreamer pipeline demonstrates splitting each frame into several tiles which are processed independently by `hailonet` element. This method is especially effective for detecting small objects in high-resolution frames.

Model

- `ssd_mobilenet_v1_visdrone` in resolution of 300X300: https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/networks/ssd_mobilenet_v1_visdrone.yaml

The VisDrone dataset consists of only small objects which we can assume are always confined within an single tile. As such it is better suited for running single-scale tiling with little overlap and without additional filtering.

Options

```
./tiling.sh [OPTIONS] [-i INPUT_PATH]
```

- `-i --input` is an optional flag, a path to the video file displayed.
- `--print-gst-launch` prints the ready gst-launch command without running it
- `--show-fps` optional - enables printing FPS on screen
- `--tiles-x-axis` optional - set number of tiles along x axis (columns)
- `--tiles-y-axis` optional - set number of tiles along y axis (rows)
- `--overlap-x-axis` optional - set overlap in percentage between tiles along x axis (columns)
- `--overlap-y-axis` optional - set overlap in percentage between tiles along y axis (rows)
- `--iou-threshold` optional - set iou threshold for NMS.
- `--sync-pipeline` optional - set pipeline to sync to video file timing.

Run

```
cd $TAPPAS_WORKSPACE/apps/h8/gstreamer/rockchip/tiling
./tiling.sh
```

The output should look like:

How it works

This app is based on our *tiling pipeline template*

4. Hailo15 C++ Example Application

4.1. Overview

This application serves as reference code for developers who wish to build an end-to-end application on Hailo15 that integrates Hailo's AI inference capabilities and other hardware accelerated features. The application demonstrates how to assemple a native C++ pipeline with the following features:

- Integration with Media Library vision Frontend
- Integration with hardware acclerated encoders (H264, H265)
- Image cropping and resizing (tiling) using hardware accelerated DSP
- Inference using Hailort Async API
- Image overlay (OSD) using hardware accelerated DSP
- DMA buffer utilization
- Streaming to a remote server using RTSP

The reference shows how a user can integrate such features in a multi-threaded pipeline that maintains zero-copy behavior for high performance.

4.2. Running the Application

The applicaton will come pre-compiled and ready to run on the Hailo15 platform as part of the release image.

To run the ai_example_app application, follow these steps:

1. **On the host machine, run a gstreamer streaming pipeline to capture video feed from the ethernet cable.**
   Enter the following command in the terminal of the host machine:

   ```
   $ gst-launch-1.0 udpsrc port=5000 address=10.0.0.2 ! application/x-rtp,
    ↪encoding-name=H264 !
   queue max-size-buffers=30 max-size-bytes=0 max-size-time=0 leaky=no !
    ↪rtpjitterbuffer mode=0 !
   queue max-size-buffers=30 max-size-bytes=0 max-size-time=0 leaky=no !
    ↪rtph264depay !
   queue max-size-buffers=30 max-size-bytes=0 max-size-time=0 leaky=no !
    ↪h264parse ! avdec_h264 !
   queue max-size-buffers=30 max-size-bytes=0 max-size-time=0
    ↪leaky=downstream ! videoconvert n-threads=8 !
   queue max-size-buffers=30 max-size-bytes=0 max-size-time=0 leaky=no !
    ↪fpsdisplaysink text-overlay=false sync=false
   ```

   This will start the streaming pipeline and you will be able to see the video feed on the screen after starting the application in the next step.

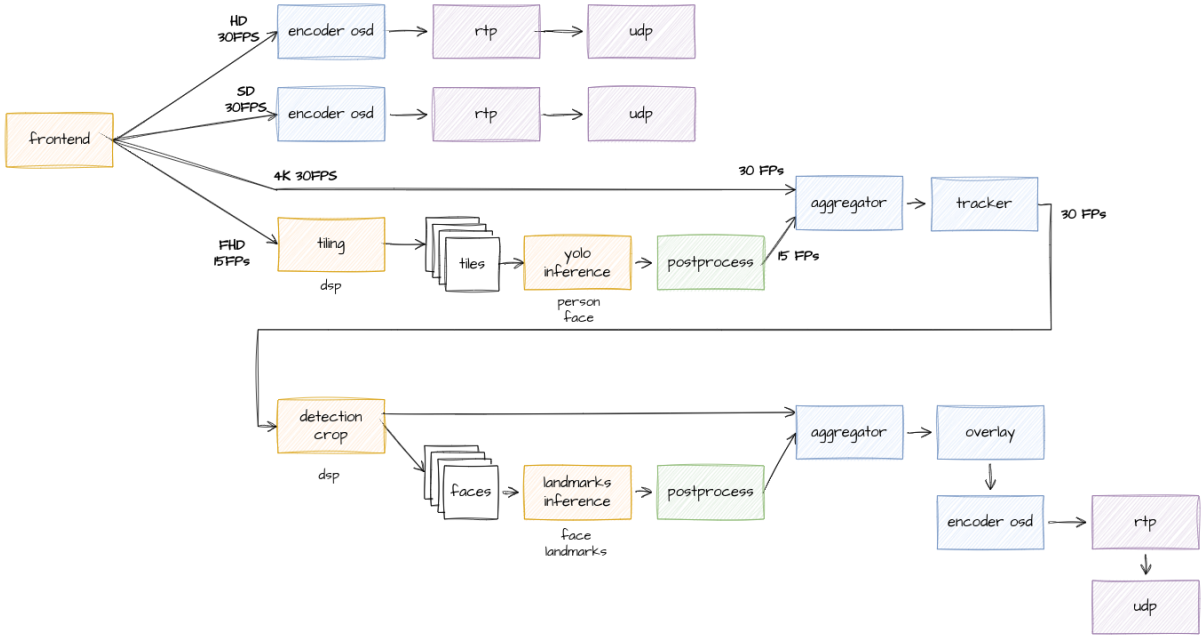
2. On the Hailo15 platform, run the executable located at the following path:

   ```
   $ ./apps/ai_example_app/ai_example_app
   ```

You should now be able to see the video feed with the inference overlay on the screen.

4.3. Application at a Glance

Now that you are able to run the application, let's discuss what you are seeing. Below you can see the pipeline that the application is running:



This may look like a lot at first, so we will break it down into smaller peices later. For now the key takeways are:

- The pipeline outputs 3 streams: two of just video (HD and SD), and a third (4K) with the inference overlay.
- **The AI pipeline is comprised of two stages:**
  - **The first stage performs yolo object detection (person and face classes) on a tiled stream**
    - Netwrork: yolov5s_personface_nv12
    - Input: 640x640 NV12
    - Classes: Person, Face
    - Output: FLOAT32, HAILO NMS(number of classes: 2, maximum bounding boxes per class: 80, maximum frame size: 3208)
  - **The second stage performs facial landmarking on faces detected in the first stage**
    - Netwrork: tddfa_mobilenet_v1_nv12
    - Input: 120x120 NV12
    - Output: UINT8, NC(62)
- In both stages of the AI pipeline the DSP is used to crop and resize the image before inference is performed

4.4. Seeing the Other Streams

In *Running the Application*, we saw how to see the inference overlay stream. If you want to see the other streams (HD & SD), you simply need to open more streaming pipelines on the host machine.

Each stream is output on a different port, so you will need to open a new pipeline for each stream you want to see. Note that in the streaming pipeline shown before targets a specific port (port=5000). To target another port, you will need to change the port number in the pipeline and run it separately. The application outputs streams to the following port numbers:

- HD Stream: 5002
- SD Stream: 5004
- 4K Stream: 5000

For example, to display the HD stream run the following ajusted pipeline on the host machine:

```
$ gst-launch-1.0 udpsrc port=5002 address=10.0.0.2 ! application/x-rtp,
 ↪encoding-name=H264 !
queue max-size-buffers=30 max-size-bytes=0 max-size-time=0 leaky=no !
 ↪rtpjitterbuffer mode=0 !
queue max-size-buffers=30 max-size-bytes=0 max-size-time=0 leaky=no !
 ↪rtph264depay !
queue max-size-buffers=30 max-size-bytes=0 max-size-time=0 leaky=no !
 ↪h264parse ! avdec_h264 !
queue max-size-buffers=30 max-size-bytes=0 max-size-time=0
 ↪leaky=downstream ! videoconvert n-threads=8 !
queue max-size-buffers=30 max-size-bytes=0 max-size-time=0 leaky=no !
 ↪fpsdisplaysink text-overlay=false sync=false
```

4.5. Where to go from here

Further documentation is available in the following sections:

- Understanding the Pipeline: Further details on the reference pipeline presented with focus on the AI stream.
- Compiling and Deploying: The application is pre-compiled and ready to run on the Hailo15 platform. If you want to make changes to the application, you will need to compile it yourself.
- Application Structure: An in depth look at the technical design of how the application is implemented. Here design decisions are explained.

5. TAPPAS as a framework

5.1. TAPPAS Framework

TAPPAS is a GStreamer based library of plug-ins. It enables using a Hailo devices within gstreamer pipelines to create intelligent video processing applications.

5.1.1. What is GStreamer?

GStreamer is a framework for creating streaming media applications.

GStreamer's development framework makes it possible to write any type of streaming multimedia application. The GStreamer framework is designed to simplify for the user how to write applications that handle audio or video or both. It isn't restricted to audio and video and can process any kind of data flow. The framework is based on plugins that will provide various codecs and other functionalities. The plugins can be linked and arranged in a pipeline. This pipeline defines the flow of the data. The GStreamer core function is to provide a framework for plugins, data flow, and media type handling/negotiation. It also provides an API to write applications using the various plugins.

For additional details check *GStreamer overview*

5.1.2. Hailo GStreamer Concepts

The functionality that Hailo brings into the GStreamer-framework allows us to infer video frames easily and intuitively without compromising on performance and flexibility.

Hailo Concepts

- **Network encapsulation** - Since in a configured network group, there are only input and output layers a GstHailoNet will be associated to a "Network" by its configured input and output pads
- **Network independent elements** - The GStreamer elements will be network independent, so the same infrastructure elements can be used for different applicative pipelines that use different NN functionality, configuration, activation, and pipelines. Using the new API we can better decouple network configuration and activation stages and thus better support network switch
- **GStreamer Hailo decoupling** - Applicative code will use Hailo API and as such will be GStreamer independent. This will help us build and develop the NN and postprocessing functionality in a controlled environment (with all modern IDE and debugging capabilities).
- **Context control** - Hailo elements will be contextless and thus leave the context (thread) control to the pipeline builder
- **GStreamer reuse** - The pipeline will use many off the shelf GStreamer elements

Hailo GStreamer Elements

- *HailoNet* - Element for sending and reciveing data from Hailo-8 chip
- *HailoFilter* - Element that enables the user to apply a postprocess or drawing operation to a frame and its tensors
- *HailoPython* - Element that enables the user to apply a postprocess or drawing operation to a frame and its tensors via python.
- *HailoMuxer* - Muxer element used for Multi-Hailo-8 setups
- *HailoDeviceStats* - Hailodevicestats is an element that samples power and temperature

- *HailoAggregator* - HailoAggregator is an element designed for applications with cascading networks. It has 2 sink pads and 1 source
- *HailoCropper* - HailoCropper is an element designed for applications with cascading networks. It has 1 sink and 2 sources
- *HailoTileAggregator* - HailoTileAggregator is an element designed for applications with tiles. It has 2 sink pads and 1 source
- *HailoTileCropper* - HailoCropper is an element designed for applications with tiles. It has 1 sink and 2 sources
- *HailoTracker* - HailoTracker is an element that applies Joint Detection and Embedding (JDE) model with Kalman filtering to track object instances.
- *HailoRoundRobin* - HailoRoundRobin is an element that provides muxing functionality in roundrobin method.
- *HailoStreamRouter* - HailoStreamRouter is an element that provides de-muxing functionality.

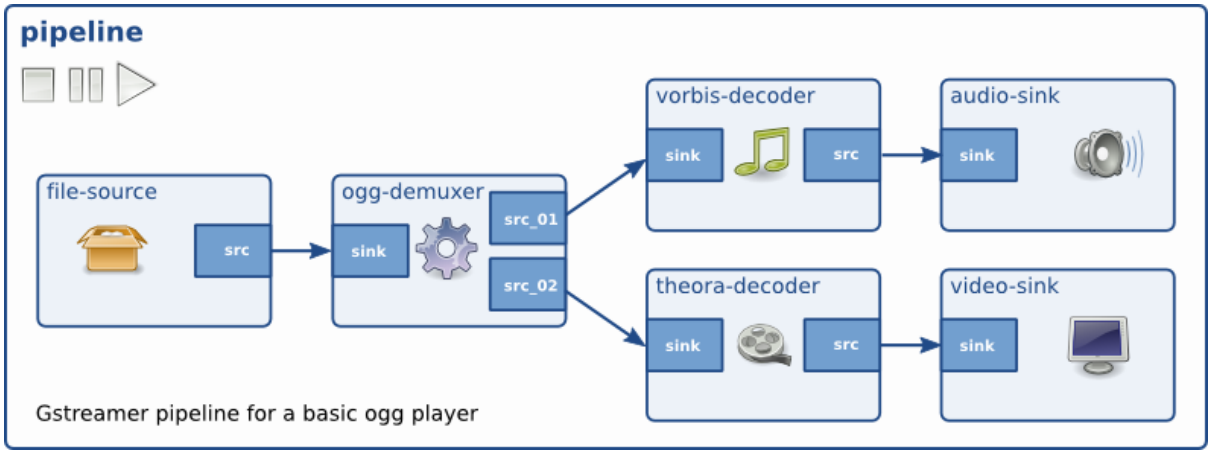
5.2. GStreamer Framework

5.2.1. GStreamer Principles

- Object-oriented - All GStreamer Objects can be extended using the GObject inheritance methods. All plugins are loaded dynamically and can be extended and upgraded independently.
- GStreamer adheres to GObject, the GLib 2.0 object model. A programmer familiar with GLib 2.0 or GTK+ will be comfortable with GStreamer.
- Extensible core plugins to encapsulate all common media streaming functionalities.
- Allow binary-only plugins - Plugins are shared libraries that are loaded at runtime.
- High performance
  - using GLib's GSlice allocator
  - ref-counting and copy on write minimize the usage of memcpy.
  - allowing hardware acceleration by using specialized plugins.

5.2.2. GStreamer Elements

- **Elements** - have one specific function for processing/ generating / consuming data. By chaining together several such elements, a pipeline can be created to perform a specific task.
- **Pads** - are an element's input and output, which can connect to other elements. A pad can be viewed as a "plug" or "port" on an element where links may be made with other elements, and through which data can flow to or from those elements. Data types are negotiated between pads using a process called Caps Negotiation. Data types are described by GstCaps.
- **Bin** - A bin is a container for a collection of elements. Since bins are subclasses of elements themselves, a bin can be mostly controlled as if it was an element, thereby subtracting away a lot of complexity from the user's application. A pipeline is a top-level bin. It provides a bus for the application and manages the synchronization for its children.



5.2.3. Terminology

NVR (Network Video Recorder)

NVR is a specialized hardware and software solution used in IP (Internet Protocol) video surveillance systems. In most cases, the NVR is intended for obtaining video streams from the IP cameras (via the IP network) for the purpose of storage and subsequent playback.

Real Time Streaming Protocol (RTSP)

Is a network control protocol designed for use in entertainment and communications systems to control streaming media servers. This protocol is used for establishing and controlling media sessions between endpoints.

Rockchip Media Process Platform (MPP)

The MPP is a library for Rockchip SoC's which provides high performance on multimedia (video and image) processing.

5.3. Write Your Own Application

5.3.1. Benchmark Tool

The benchmark tool is a new script that can help the user identify their platform's limitations when trying to build an application using TAPPAS baseline. This tool can be used to measure the FPS of different pipelines according to a variety of parameters. The pipeline that will run will be without any neural network and is able to demonstrate the performance the platform is able to provide under the defined conditions.

Benchmark Usage

Options:

--help Show this help

--video-prefix-path PATH Video prefix path

--min-num-of-sources NUM Setting number of sources to given input (Default is 1)

--max-num-of-sources NUM Setting number of sources to given input (Default is 4)

--num-of-buffers NUM Number of buffers for each stream (Default is 500)

--use-vaapi	Whether to use vaapi decoding or not (Default is no vaapi)
--use-display	Whether to use display or not (Default is no display)
--format FORMAT	Required format
--display-resolution RESOLUTION	Scale width and height of each stream in WxH mode (e.g. 640x480)

Usage Notes

1. The format should be written according to the GStreamer conventions, e.g. RGBA instead of rgba.
2. If the min and max number of sources are the same only 1 pipeline will run, otherwise several pipelines will be running consecutively.
3. If you want to use VA-API make sure you download VA-API via from the link *VA-API Install Manual*

Videos For Benchmark Tool

Before using the benchmark tool videos must be prepared for it to run. The videos must be in format mp4 . A working script has been prepared script which takes one video and creates as many soft links to this video as the user requires, numbered from 0 to n-1 for the benchmark tool. When providing the prefix after –video-prefix-path in the benchmark tool, write a path with the name of the videos until the numbering. Example: If the videos are located at /tmp/tappas/ and their names are tap0.mp4 to tap15.mp4, the prefix will be "/tmp/tappas/tap".

Benchmark Example

5.3.2. Compiling Your Code

When building the TAPPAS *Docker* (or *installing natively*), all of the TAPPAS source files are compiled and ready to use. If however you want to add your own additions (*for example, a new postprocess*) or make other changes, it will be necessary to recompile the sources. This guide covers the build system used in TAPPAS and how to compile the project.

The Meson Build System

Meson is an open source build system that places the emphasis on speed and ease of use. GStreamer uses meson for all sub-projects to generate build instructions to be executed by ninja, another build system focuses entirely on a speed that requires a higher level build system (ie: meson) to generate its input files. Similar to GStreamer, TAPPAS also uses Meson, and compiling new projects requires the adjustment of the `meson.build` files.

How to Compile

To help streamline this process the TAPPAS Framework have provided a script that handles most of the work. The script can be found at scripts/gstreamer/install_hailo_gstreamer.sh.
The following arguments are available:

- `--build-dir` Path to the build directory. Defaults to `core/hailo`.
- `--build-mode` Build mode, debug/release, default is release.
- `--skip-hailort` Skip compiling HailoRT.
- `--python-version` Specify which Python version to use.
- `--compile-libgsthailo` Compile libgsthailo instead of copying it from the release.

From the TAPPAS home directory folder you can run:


```
./scripts/gstreamer/install_hailo_gstreamer.sh
```

```
hailo@hailo_tappas:/local/workspace/tappas$ ./scripts/gstreamer/install_hailo_gstreamer.sh
/local/workspace/tappas/core/hailo/gstreamer /local/workspace/tappas
Directory already configured.

Just run your build command (e.g. ninja) and Meson will regenerate as necessary.
If ninja fails, run "ninja reconfigure" or "meson --reconfigure"
to force Meson to regenerate.

If build failures persist, run "meson setup --wipe" to rebuild from scratch
using the same options as passed when configuring the build.
To change option values, run "meson configure" instead.
ninja: Entering directory `build.release'
[0/1] Regenerating build files.
The Meson build system
Version: 0.58.0
Source dir: /local/workspace/tappas/core/hailo/gstreamer
Build dir: /local/workspace/tappas/core/hailo/gstreamer/build.release
Build type: native build
Project name: gst-hailo-tools
Project version: 3.16.0
C compiler for the host machine: gcc-9 (gcc 9.4.0 "gcc-9 (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0")
C linker for the host machine: gcc-9 ld.bfd 2.30
C++ compiler for the host machine: g++-9 (gcc 9.4.0 "g++-9 (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0")
C++ linker for the host machine: g++-9 ld.bfd 2.30
Host machine cpu family: x86_64
Host machine cpu: x86_64
Configuring config.h using configuration
Message: GCC >= 9.0.0 detected, applying extra arguments.
Dependency gstreamer-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-base-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-app-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-video-1.0 found: YES 1.14.5 (cached)
Dependency opencv4 found: YES 4.5.2 (cached)
Dependency glib-2.0 found: YES 2.56.4 (cached)
Dependency gmodule-2.0 found: YES 2.56.4 (cached)
Dependency gobject-2.0 found: YES 2.56.4 (cached)
Dependency pygobject-3.0 found: YES 3.26.1 (cached)
Program python3.6 found: YES (/usr/bin/python3.6)
Found pkg-config: /usr/bin/pkg-config (0.29.1)
Dependency python found: YES (pkgconfig)
Dependency python found: YES (pkgconfig)
Dependency blas found: YES 3.7.1 (cached)
Dependency lapack found: YES 3.7.1 (cached)
Library dl found: YES
Library stdc++fs found: YES
Dependency threads found: YES unknown (cached)
Build targets in project: 29
```

5.3.3. Hailo Objects API

The TAPPAS provides a set of abstractions for the different objects that might be handled in a user's pipeline, such as `tensors`, `detections`, `classifications`, or `landmarks`. If the user wants to write their own custom pipeline behavior (`postprocessing`, `drawing`, etc..), then it is worthwhile to become familiar with these classes.

This guide documents the different classes available in the TAPPAS and their interactions.

Structs

HailoBBox

Represents a bounding box.
SOURCE: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoBBox(float xmin, float ymin, float width, float height)
```

Functions

Function	Return Type	Description
`xmin()`	float	Normalized xmin position.
`ymin()`	float	Normalized ymin position.
`width()`	float	Normalized width of bounding box.
`height()`	float	Normalized height of bounding box.
`xmax()`	float	Normalized xmax position.
`ymax()`	float	Normalized ymax position.

HailoPoint

Represents a detected point (landmark).
SOURCE: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoPoint(float x, float y, float confidence = 1.0f)
```

Functions

Function	Return Type	Description
`x()`	float	Normalized x position.
`y()`	float	Normalized y position.
`confidence()`	float	The confidence in the point's accuracy, float between 0.0 to 1.0 - default is 1.0.

Enumerations

hailo_object_t

Typing enumeration for *HailoObject* instances.

```
typedef enum
{
  HAILO_ROI,
  HAILO_CLASSIFICATION,
  HAILO_DETECTION,
  HAILO_LANDMARKS,
  HAILO_TILE,
  HAILO_UNIQUE_ID,
  HAILO_MATRIX,
  HAILO_DEPTH_MASK,
  HAILO_CLASS_MASK,
  HAILO_CONF_CLASS_MASK
} hailo_object_t;
```

HailoTensor

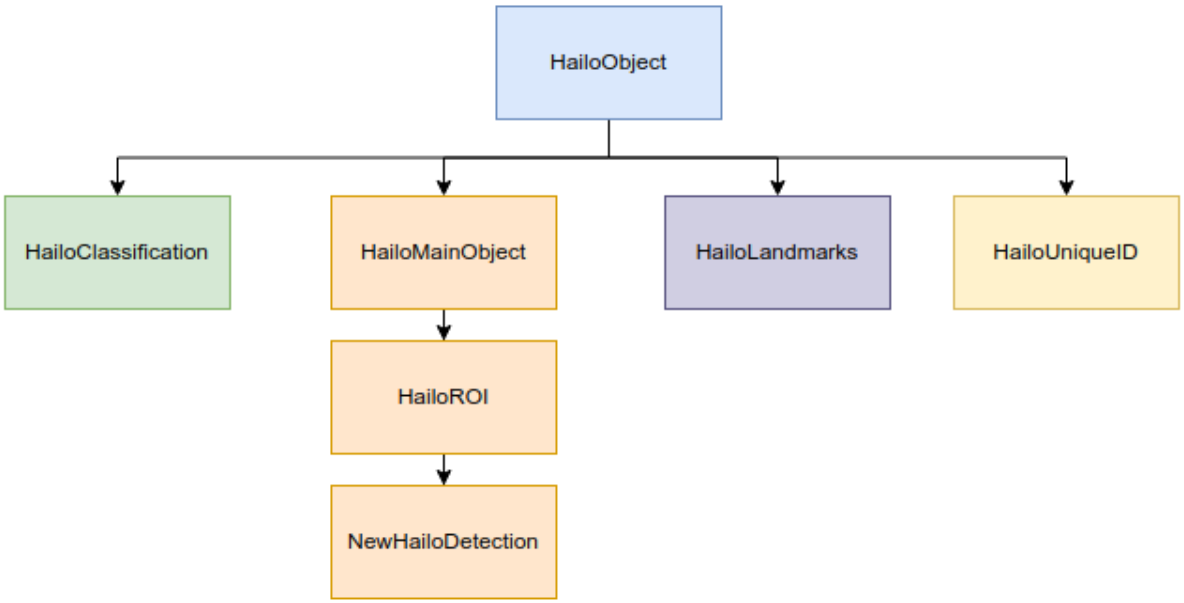
`Tensors` are the ouput vector of the network inference. Usually these are N-dimensional matrices that hold little "human readable" value at first, but after a little postprocessing become meaningful objects such as detections or landmarks. All postprocesses start by looking at the output tensors. In fact, usually there will be no need to construct one it will be *provided to your postprocess* via the *hailofilter* element. To make handling these vectors easier, they are provided in a `HailoTensor` class. `Shared pointer handle`: **HailoTensorPtr** SOURCE: core/hailo/general/hailo_tensors.hpp

Functions

Function	Return Type	Description
`name()`	std::string	Get the tensor name.
`vstream_info()`	hailo_vstream_info_t	Get the HailoRT vstream info.
`data()`	uint8_t *	Get the tensor data pointer.
`width()`	uint32_t	Get the tensor width.
`height()`	uint32_t	Get the tensor height.
`features()`	uint32_t	Get the tensor features.
`size()`	uint32_t	Get the tensor total length.
`shape()`	std::vectorstd::size_t	Get the tensor dimensions.
`fix_scale(uint8_t num)`	float	Takes a quantized number and returns its dequantized value (float).
`get(uint row, uint col, uint channel)`	uint8_t	Get the tensor value at this location.
`get_full_percision(uint row, uint col, uint channel`	float	Get the tensor dequantized value at this location.

HailoObject

`HailoObject` represents objects that are usable output after postprocessing. They can be detections, classifications, landmarks, or any other similar postprocess results.
This class is an abstraction for other objects to inherit from. To more conveniently compare different types of inheriting classes, `HailoObjects` store their object type from an enumerated list *hailo_object_t*.
The `class inheritance hierarchy` is as follows:



`Shared pointer handle`: **HailoObjectPtr**
SOURCE: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoObject()
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	The type of the object from the list of enumerated types shown above.

HailoMainObject

Inherits from *HailoObject*

`HailoMainObject` represents a *HailoObject* that can hold other *HailoObject*s. For example a face detection can hold landmarks or age classification, gender classification etc...

`Shared pointer handle`: **HailoMainObjectPtr**

`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoMainObject()
```

Functions

Function	Return Type	Description
`add_object(HailoObjectPtr obj)`	void	Add a *HailoObject* to this *HailoMainObject*.
`add_tensor(HailoTensorPtr tensor)`	void	Add a *HailoTensor* to this *HailoMainObject*.
`remove_object(HailoObjectPtr obj)`	void	Remove a *HailoObject* from this *HailoMainObject*.
`remove_object(uint index)`	void	Remove a *HailoObject* from this *HailoMainObject* by index.
`get_tensor(std::string name)`	*HailoTensorPtr*	Get a tensor from this *HailoMainObject*.
`has_tensors()`	bool	Checks whether there are tensors attached to this *HailoMainObject*.
`get_tensors()`	std::vector <*HailoTensorPtr*>	Get a vector of the tensors attached to this *HailoMainObject*.
`clear_tensors()`	void	Clear all tensors attached to this *HailoMainObject*.
`get_objects()`	std::vector <*HailoObjectPtr*>	Get the objects attached to this *HailoMainObject*.
`get_objects_typed (hailo_object_t type)`	std::vector <*HailoObjectPtr*>	Get the objects of a given type, attached to this *HailoMainObject*.

HailoROI

Inherits from *HailoMainObject*

`HailoROI` represents an ROI (Region Of Interest): a part of an image that can hold other objects. Mostly inherited by other objects but isn't abstract. Can represent the whole image by giving the right HailoBBox.

`Shared pointer handle`: **HailoROIPtr**

`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructor

```
HailoROI(HailoBBox bbox)
```

Functions

Function	Return Type	Description
`shared_from_this()`	std::shared_ptr <*HailoROI*>	Get a shared pointer to this instance.
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_ROI
`add_object(HailoObjectPtr obj)`	void	Get the bbox of this ROI.
`get_bbox()`	*HailoBBox*	Get a shared pointer to this instance.
`set_bbox(HailoBBox bbox)`	void	Set the bbox of this ROI.
`get_scaling_bbox()`	*HailoBBox*	Get the scaling bbox of this ROI, useful in case of nested ROIs.
`set_scaling_bbox(HailoBBox bbox)`	void	Set the scaling bbox of this ROI, useful in case of nested ROIs.

HailoDetection

Inherits from *HailoROI*

`HailoDetection` represents a detection in an ROI. It is assumed that all numbers are normalized (between 0 and 1) so that objects remain in relative size for easy image resizing.

`Shared pointer handle`: **HailoDetectionPtr**

`SOURCE`: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoDetection(HailoBBox bbox, const std::string &label, float confidence)
HailoDetection(HailoBBox bbox, int class_id, const std::string &label, float
↪confidence)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_DETECTION
`get_confidence()`	float	This detection's confidence.
`get_label()`	std::string	This detection's label.
`get_class_id()`	int	This detection's class id.
`operator<(const HailoDetection &other)`	bool	Overload < operator, compares confidences.
`operator>(const HailoDetection &other)`	bool	Overload > operator, compares confidences.

HailoClassification

Inherits from *HailoObject*

`HailoClassification` represents a classification of an ROI. Classifications can have different `types`, for example a classification of type 'color' can have a `label` of red or blue.

`Shared pointer handle`: **HailoClassificationPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoClassification(const std::string &classification_type, const std::string &
↪label, float confidence)
HailoClassification(const std::string &classification_type, int class_id,
↪std::string label, float confidence)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_CLASSIFICATION
`get_confidence()`	float	This classification's confidence.
`get_label()`	std::string	This classification's label (e.g. "Horse", "Monkey", "Tiger" for type "Animals").
`get_classification_type()`	std::string	This classification's type (e.g. "age", "gender", "color", etc...).
`get_class_id()`	int	This classification's class id.

HailoLandmarks

Inherits from *HailoObject*

`HailoLandmarks` represents **a set** of landmarks on a given ROI. Like *HailoClassification*, *HailoLandmarks* can also have different `types`, for example a landmark can be of type "pose" or "facial landmarking". Each landmark in the set is represented as a *HailoPoint*.

`Shared pointer handle`: **HailoLandmarksPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoLandmarks(std::string landmarks_name, float threshold = 0.0f, const std::vector
↪<std::pair<int, int>> pairs = {})
std::string landmarks_name, std::vector<HailoPoint> points, float threshold = 0.0f,
↪const std::vector<std::pair<int, int>> pairs = {})
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_LANDMARKS
`add_point(HailoPoint point)`	void	Add a point to this landmarks object.
`get_points()`	std::vector<*HailoPoint*>	Gets the set of points held by this Landmarks object.
`get_landmarks_type()`	std::string	This landmark's type (e.g. "pose estimation", "face landmark", etc...).
`get_pairs()`	std::vector<std::pair<int, int»	vector of pairs of joints that should be connected in overlay.

HailoUniqueID

Inherits from *HailoObject*

`HailoUniqueID` represents a unique id of an ROI. Sometimes the user may want to give ROIs unique ids (for example, when tracking detections), and having a *HailoObject* abstraction makes adding and removing ids very simple (via `add_object()` and `remove_object()`). If no unique if is provided at construction, then a default -1 is used.

`Shared pointer handle`: **HailoUniqueIDPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoUniqueID()
HailoUniqueID(int unique_id)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_UNIQUE_ID
`get_id()`	int	Get the unique id.

HailoMask

Inherits from *HailoObject*

`HailoMask` represents a mask of an ROI. Whenever the output of a postprocess is masks (tensors with result for every pixel) we will ROIs mask objects.

`Shared pointer handle`: **HailoMaskPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoMask(int mask_width, int mask_height, float transparency)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_MASK
`get_width()`	int	get the mask width
`get_height()`	int	get the mask height
`get_transparency()`	float	get the desired drawing transparency

HailoDepthMask

Inherits from *HailoMask*.
`HailoDepthMask` represents a mask of an ROI, with float values for each pixel. The values represent depth between minimum and maximum values.
`Shared pointer handle`: **HailoDepthMaskPtr**
SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoDepthMask(std::vector<float> &&data_vec, int mask_width, int mask_height,
↪float transparency)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_DEPTH_MASK
`get_data()`	const std::vector <*float*>	get the mask data vector

HailoClassMask

Inherits from *HailoMask*
`HailoClassMask` represents a mask of an ROI, with uint8_t class id classification for each pixel.
`Shared pointer handle`: **HailoClassMaskPtr**
SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoClassMask(std::vector<uint8_t> &&data_vec, int mask_width, int mask_height,
↪float transparency)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_CLASS_MASK
`get_data()`	const std::vector<*uint8_t*>	get the mask data vector

HailoConfClassMask

Inherits from *HailoMask*

`HailoConfClassMask` represents a mask of an ROI, contains mask-class-id and confidence float value for each pixel.

`Shared pointer handle`: **HailoConfClassMaskPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

```
HailoConfClassMask(std::vector<float> &&data_vec, int mask_width, int mask_height,
↪float transparency, int class_id)
```

Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_CONF_CLASS_MASK
`get_data()`	const std::vector<*float*>	get the mask data vector
`get_class_id()`	int	get the mask class id

HailoMatrix

Inherits from *HailoObject*

`HailoMatrix` represents a matrix, contains float values. This matrix can be added to any HailoObject for different use cases.

`Shared pointer handle`: **HailoMatrixPtr**

SOURCE: core/hailo/general/hailo_objects.hpp

Constructors

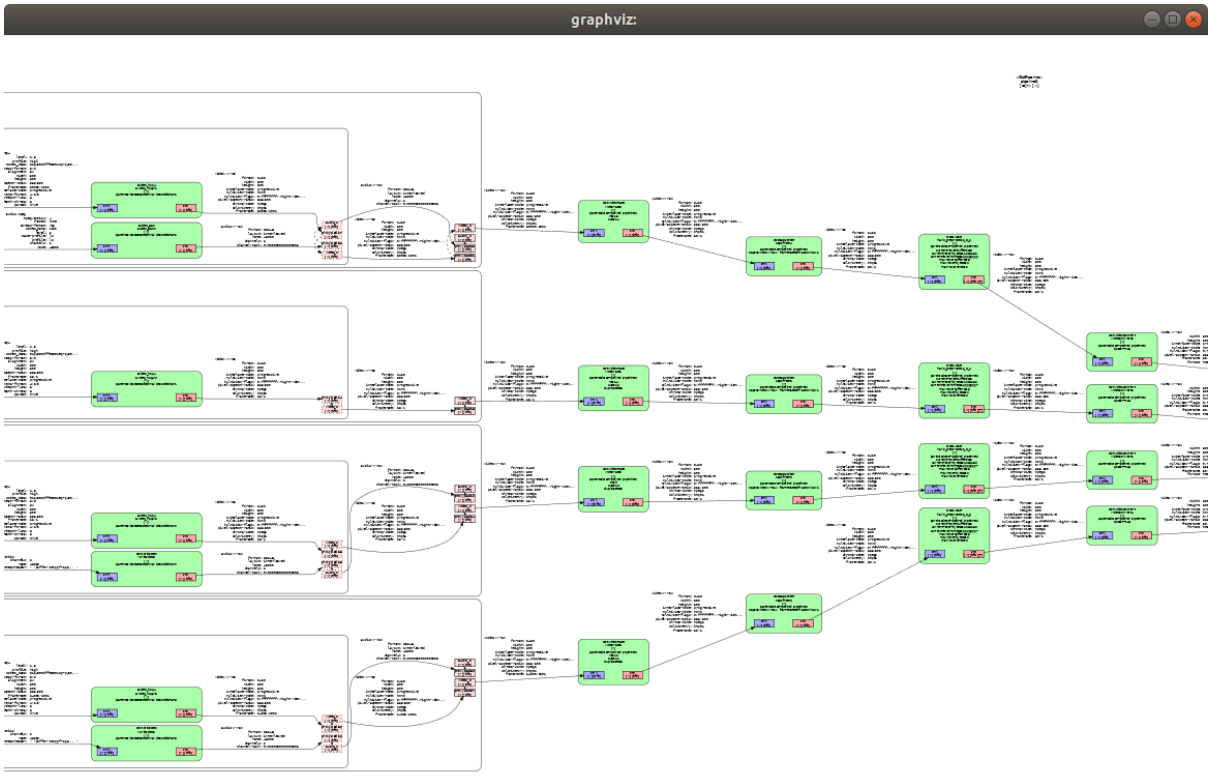
```
HailoMatrix(float *data_ptr, uint32_t mat_height, uint32_t mat_width, uint32_t mat_
↪features = HailoMatrix::DEFAULT_NUMBER_OF_FEATURES)
```


Functions

Function	Return Type	Description
`get_type()`	*hailo_object_t*	This *HailoObject*'s type: HAILO_MATRIX
`width()`	const uint32_t	get matrix width
`height()`	const uint32_t	get matrix height
`features()`	const uint32_t	get matrix number of features
`size()`	const uint32_t	get number of elements in matrix
`shape()`	std::vectorstd::size_t	get the shape of the matrix
`get_data_ptr()`	float *	get the matrix data pointer

5.3.4. Debugging

GstShark



GstShark is an open-source project from RidgeRun that provides benchmarks and profiling tools for GStreamer 1.7.1 (and above). It includes tracers for generating debug information plus some tools to analyze the debug information. GstShark provides easy to use and useful tracers, paired with analysis tools to enable straightforward optimizations.

GstShark leverages GStreamer's built in tracing system and adds plotting tools to simplify the process of understanding the bottlenecks in the user's pipeline.

As part of the TAPPAS framework, we have expanded the available profiling mechanisms to include TAPPAS related tracers and a new plotting tool.

The profiling tool provides 3 general features that can be used to debug the pipeline:

- Tracers log printouts - At the most basic level, printouts are available from the traces of the different measurements made. An experienced user will be able to make observations at runtime. Printouts can be found in the debug file: GST_DEBUG_FILE=$TAPPAS_WORKSPACE/tappas_traces.log.

- Graphic visualization - Shown above, gst-shark can generate a pipeline graph that shows how elements are connected and what caps were negotiated between them. This is a very convenient feature for observing the pipeline in a more comfortable way. The graph is generated at runtime so it is very effective for seeing and debugging how elements were actually connected and in what formats the data ended up in.
- Plotting tool - A python script that generates a graph plot for each tracer metric enabled. This is a powerful tool to visualize each metric that can be used for deeper debugging.

Install

The docker image already contains GstShark! If the user decides to not use the TAPPAS Docker image, our suggestion is to follow RidgeRun tutorial: GstShark

Bash Shortcuts

Note: These shortcuts are only available if the Docker TAPPAS installation is being used, otherwise, please refer to the following section: *Using GstShark in yocto-compiled images*

As part of the TAPPAS creation of the Docker image, some convenient shortcuts are copied to GstShark:

```
vim ~/.bashrc
```

```
# set gstreamer debug
gst_set_debug() {
 export HAILO_PROFILE_LOCATION=/tmp/profile
 export GST_DEBUG="GST_TRACER:7"
 export GST_DEBUG_FILE=$TAPPAS_WORKSPACE/tappas_traces.log
 export GST_TRACERS="cpuusage;proctime;interlatency;scheduletime;bitrate;
↪framerate;queuelevel;threadmonitor;numerator;buffer;detections;graphic"
 export GST_DEBUG_NO_COLOR=1
 echo 'Options for TRACERS:"'
 echo 'export GST_TRACERS="cpuusage;proctime;interlatency;scheduletime;bitrate;
↪framerate;queuelevel;threadmonitor;numerator;buffer;detections;graphic"'
}

# set trace to collect graphic data of gstreamer pipeline
gst_set_graphic() {
 export HAILO_PROFILE_LOCATION=/tmp/profile
 export GST_DEBUG_DUMP_DOT_DIR=/tmp/
 export GST_DEBUG="GST_TRACER:7"
 export GST_TRACERS="graphic"
 echo 'export GST_TRACERS="graphic"'
}

# unset gstreamer debug
gst_unset_debug() {
 unset GST_TRACERS
 unset GST_DEBUG
 unset GST_DEBUG_DUMP_DOT_DIR
 unset HAILO_PROFILE_LOCATION
 unset GST_DEBUG_FILE
 unset GST_DEBUG_NO_COLOR
}

# plot the gst-shark dump files
gst_plot_debug() {
 export HAILO_PROFILE_LOCATION=/tmp/profile
 split_traces_dir=$TAPPAS_WORKSPACE/tappas_traces_$(date +%d.%m.%Y_%H:%M:%S)
```

(continues on next page)

(continued from previous page)

```
 $TAPPAS_WORKSPACE/sources/gst-shark/scripts/graphics/split_traces.sh $split_
↪traces_dir
 python3 $TAPPAS_WORKSPACE/sources/gst-shark/scripts/graphics/plot_all_to_html.
↪py -p $split_traces_dir
 echo 'In order to plot the graphic pipeline graph, run:'
 echo "dot $HAILO_PROFILE_LOCATION/graphic/pipeline_<timestamp>.dot -T x11"
}
```

Note that 4 functions were added: two sets, an unset, and a plot function. The set functions enable gst-shark by setting environment variables, the chief of which is GST_TRACERS. This enables the different trace hooks in the pipeline. The available tracers are listed in the echo command at the end of each set. You can enable any combination of the available tracers, just chain them together with a ';' (notice that the difference between gst_set_debug and gst_set_graphic is that gst_set_debug enables all tracers whereas gst_set_graphic only enables the graphic tracer that draws the pipeline graph). HAILO_PROFILE_LOCATION and GST_DEBUG_DUMP_DOT_DIR set locations where the dump files are stored, the first sets where the tracer dumps are (used for gst-plot), and the latter where the dot file is saved (the graphic pipeline graph). Unset disables all tracers, and gst_plot_debug runs plot script.

Using GstShark in Yocto-Compiled Images

Enable TAPPAS tracers:

- Export the following environment variables:

```
export HAILO_PROFILE_LOCATION=/tmp/profile
export GST_DEBUG="GST_TRACER:7"
export GST_DEBUG_NO_COLOR=1
```

- Select the tracers by setting the GST_TRACERS environment variable to the list of tracers, separated by ; as in the example:

```
export GST_TRACERS="scheduletime;bitrate;threadmonitor;numerator;buffer;
↪detections"
```

- You should export only some of the tracers, exporting too many tracers may fail the embedded device. All tracers explained: *Understanding GstShark tappas plotted graphs*
- If you want to save the output to a file, run the following command before running the app:

```
export GST_DEBUG_FILE=<file_path>
```

- If you want to plot the tracers output, use a strong machine (not an embedded device) with a full tappas installation and copy there the output file, then run the following commands:

```
export GST_DEBUG_FILE=<file_path>
gst_plot_debug
```

Using GstShark

Let's say there is a GStreamer app to be profiled, Start by enabling gst-shark:

Then just run your app, it will be possible to see all kinds of tracer prints on the debug output file: GST_DEBUG_FILE=$TAPPAS_WORKSPACE/tappas_traces.log.

After you have run a gstreamer pipeline with tracers enabled, plot them using the plot script, just run:

```
gst_plot_debug
```

It will print to the console the path of the html file that contains the plots, which can be opened in the browser. In addition it will print the command to open the pipeline graph, which can be run in a terminal to open the graph.

Understanding GstShark TAPPAS Plotted Graphs

Each graph inspects a different metric of the pipeline, it is recommended to read more about what each one represents here:

- CPU Usage (cpuusage) - Measures the CPU usage every second. In multiprocessor systems this measurements are presented per core.
- Processing Time (proctime) - Measures the time an element takes to produce an output given the corresponding input.
- InterLatency (interlatency) - Measures the latency time at different points in the pipeline.
- Schedule Time (scheduling) - Measures the amount of time between two consecutive buffers in a sink pad.
- Buffer (buffer) - Prints information of every buffer that passes through every sink pad in the pipeline.
- Bitrate (bitrate) - Measures the current stream bitrate in bits per second.
- Framerate (framerate) - Measures the amount of frames that go through a src pad every second.
- Queue Level (queuelevel) - Measures the amount of data queued in every queue element in the pipeline.
- Thread Monitor (threadmonitor) - Measures the CPU usage of every thread in the pipeline.
- Numerator (numerator) - Numerates the buffers by setting the field “offset” of the buffer metadata. This trace is different from the others because it does not collect any data, it just numerates the buffers.
- Detections (detections) - Prints information about the objects detected in every buffer that passes through every pad in the pipeline. This trace only works with the TAPPAS framework since it collects the TAPPAS detection objects.
- Graphic (graphics) - Records a graphical representation of the current pipeline.

Note: When using the Thread Monitor tracer, provide meaningful names to the queues because the names of the threads in the graph will be based on the names of the queues. This will help easily identify the threads and understand their purpose when analyzing the trace. In addition, due to the way this tracer works, it is important to keep the names of the queues shorter than 16 characters. If the names are longer than this, the thread names in the graph will be truncated.

Modify Buffering Mode and Size

```
$ export GST_SHARK_FILE_BUFFERING=0
```

With the no buffering mode every I/O operation is written as soon as possible.

The following command is an example of how to define the environment variable that will change the buffering mode to full buffering and the buffering size, this command uses a positive integer value for the size:

```
$ export GST_SHARK_FILE_BUFFERING=1024
```


Individual Element Tracing (filter)

The individual element tracing, or filter parameter, allows the user to choose which elements get included in the tracing. The value to be set in the filter is a Glib Compatible Regular Expression, meaning that elements to be traced can be grouped by using a regex that matches with their name.

The filtering applies to the element name, NOT the factory. This is, if the element is specified as "identity name=myelem", it should be referred to as "myelem" and not to "identity"

Print the amount of frames that flow every 5 seconds through the different src pads in the pipe:

```
GST_TRACERS="framerate(period=5)" GST_DEBUG=GST_TRACER:7
```

Print the amount of bits that flow every 3 seconds through the different src pads in the pipe:

```
GST_TRACERS="bitrate(period=3)" GST_DEBUG=GST_TRACER:7
```

Print the amount of frames that flow every 5 seconds and bits that flow every 3 seconds through the different src pads in the pipe:

```
GST_TRACERS="framerate(period=5);bitrate(period=3)" GST_DEBUG=GST_TRACER:7
```

Print the amount of frames that flow every 5 through the identity:

```
GST_TRACERS="framerate(period=5,filter=identity);bitrate(period=3)" GST_
↪DEBUG=GST_TRACER:7
```

Using gst-instruments

gst-instruments is a set of performance profiling and data flow inspection tools for GStreamer pipelines.

- `gst-top-1.0` at the start of the pipeline will analyze and profile the run. (gst-top-1.0 gst-launch-1.0 ! audiotestsrc ! autovideosink)
- `gst-report-1.0` - generates performance report for input trace file.
- `gst-report-1.0 --dot gst-top.gsttracee | dot -Tsvg > perf.svg` - generates performance graph in DOT format.

Read more in gst-instruments github page

5.3.5. Writing Your Own Postprocess

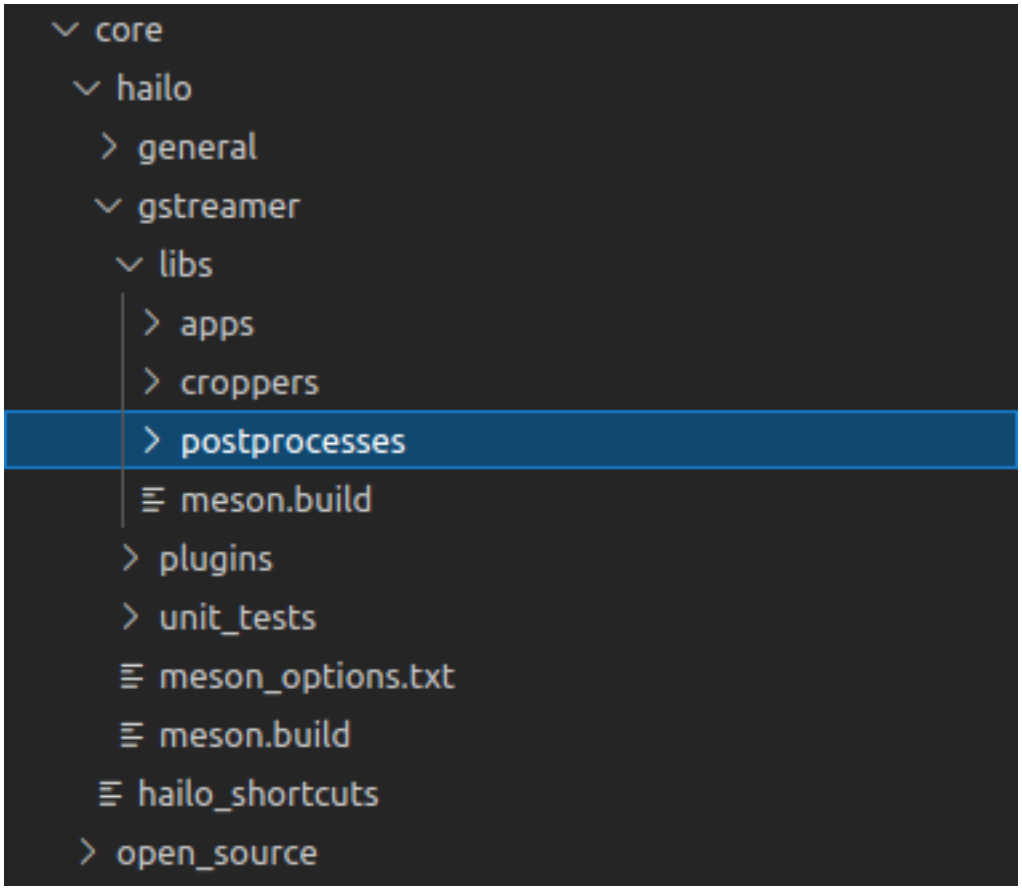
Overview

When adding a network to the TAPPAS that is not already supported, then most likely it will need to implement a new post-process and drawing filter. Fortunately with the use of the *hailofilter*, there is no need to create any new GStreamer elements, just provide the .so (compiled shared object binary) that applies to the filter! This guide will go over how to create such an .so and what mechanisms/structures are available to the user as they create their postprocess.

Getting Started

File Location

When creating or working with postprocess, it is important to know where to find all the relevant source files that already exist, and how to add new ones. From the TAPPAS home directory, one can find the `core/` folder. Inside this `core/` directory are a few subdirectories that host different types of source files. The `open_source` folder contains source files from 3rd party libraries (opencv, xtensor, etc..), while the `hailo` folder contains source files for all kinds of Hailo tools, such as the Hailo Gstreamer elements, the different metas provided, and the source files for the postprocesses of the networks that were already provided in the TAPPAS. Inside this directory is one titled `general/`, which contains sources for *the different object classes* (detections, classifications, etc..) available. Next to `general` is a directory titled `gstreamer/`, and inside that are two folders of interest: `libs/` and `plugins/`. The former contains the source code for all the postprocess and drawing functions packaged in the TAPPAS, while the latter contains source code for the different Hailo GStreamer elements, and the different metas available. This guide will mostly focus on this `core/hailo/` directory, as it has everything needed to create and compile a new .so! It is recommended that users spend time familiarizing themselves with these locations, and then when ready to continue enter the `postprocesses/` directory:



Preparing the Header File: Default Filter Function

The new postprocess can be created here in the `postprocesses/` folder. Create a new header file named `my_post.hpp`.In the first lines we want to import useful classes to our postprocess, so add the following includes:

```
#pragma once
#include "hailo_objects.hpp"
#include "hailo_common.hpp"
```

`"hailo_objects.hpp"` contains classes that represent the *different inputs (tensors) and outputs* (detections, classifications, etc...) that your postprocess might handle. You can find the header in core/hailo/general/hailo_objects.hpp. Your main point of entry for data in the postprocess is the `HailoROI`, which can have a tensor or a number of tensors attached. `"hailo_common.hpp"` provides common useful functions for handling these classes. Wrap up the header file by adding a function prototype for your filter, the whole header file should look like:

```
#pragma once
#include "hailo_objects.hpp"
#include "hailo_common.hpp"

__BEGIN_DECLS
void filter(HailoROIPtr roi);
__END_DECLS
```

The process is now complete. The `hailofilter` element does not expect much, just that the above `filter` function be provided. Adding *Multiple Filters in One .so* will be described later. Note that the `filter` function takes a `HailoROIPtr` as a parameter; this will provide you with the `HailoROI` of each passing image.

Implementing a Filter()

Start implementing the actual filter to see how to access and work with tensors. Start by creating a new file called `my_post.cpp`. Open it and include the following:

```
#include <iostream>
#include "my_post.hpp"
```

The `<iostream>` will allow printing to the console, and the `"my_post.hpp"` includes the header file we just wrote.

For now add the following implementation for the filter() so that we have a working postprocess we can test:

```
// Default filter function
void filter(HailoROIPtr roi)
{
 std::cout << "My first postprocess!" << std::endl;
}
```

This is sufficient for compiling and running a pipeline. Next we will describe how to add the postprocess to the meson project so that it compiles.

Compiling and Running

Building with Meson

Meson is an open source build system that places an emphasis on speed and ease of use. GStreamer uses meson for all sub-projects to generate build instructions to be executed by ninja, another build system focuses specifically on speed that requires a higher level build system (ie: meson) to generate its input files. Like GStreamer, TAPPAS also uses meson, and compiling new projects requires adjusting the `meson.build` files. Here we will describe how to add the user's. In the `libs/postprocesses` path will be found a meson.build, open it and add the following entry for our postprocess:

```
################################################
# MY POST SOURCES
################################################
my_post_sources = [
 'my_post.cpp',
]

my_post_lib = shared_library('my_post',
 my_post_sources,
 include_directories: [hailo_general_inc] + xtensor_inc,
 dependencies : post_deps,
 gnu_symbol_visibility : 'default',
 install: true,
 install_dir: post_proc_install_dir,
)
```

This will provide meson with all the information required to compile the postprocess. In short, we are providing paths to cpp compilers, linked libraries, included directories and dependencies. All these path variables come from the parent meson project, the meson file can be read to see what packages and directories are available at core/hailo/meson.build.

Compiling the .so

We are now ready to compile the postprocess. To help streamline this process a script has been provided that will handle most of the work. The script can be found at scripts/gstreamer/install_hailo_gstreamer.sh. It includes some flags that allows the user to do more specific operations, but they are not needed right now.

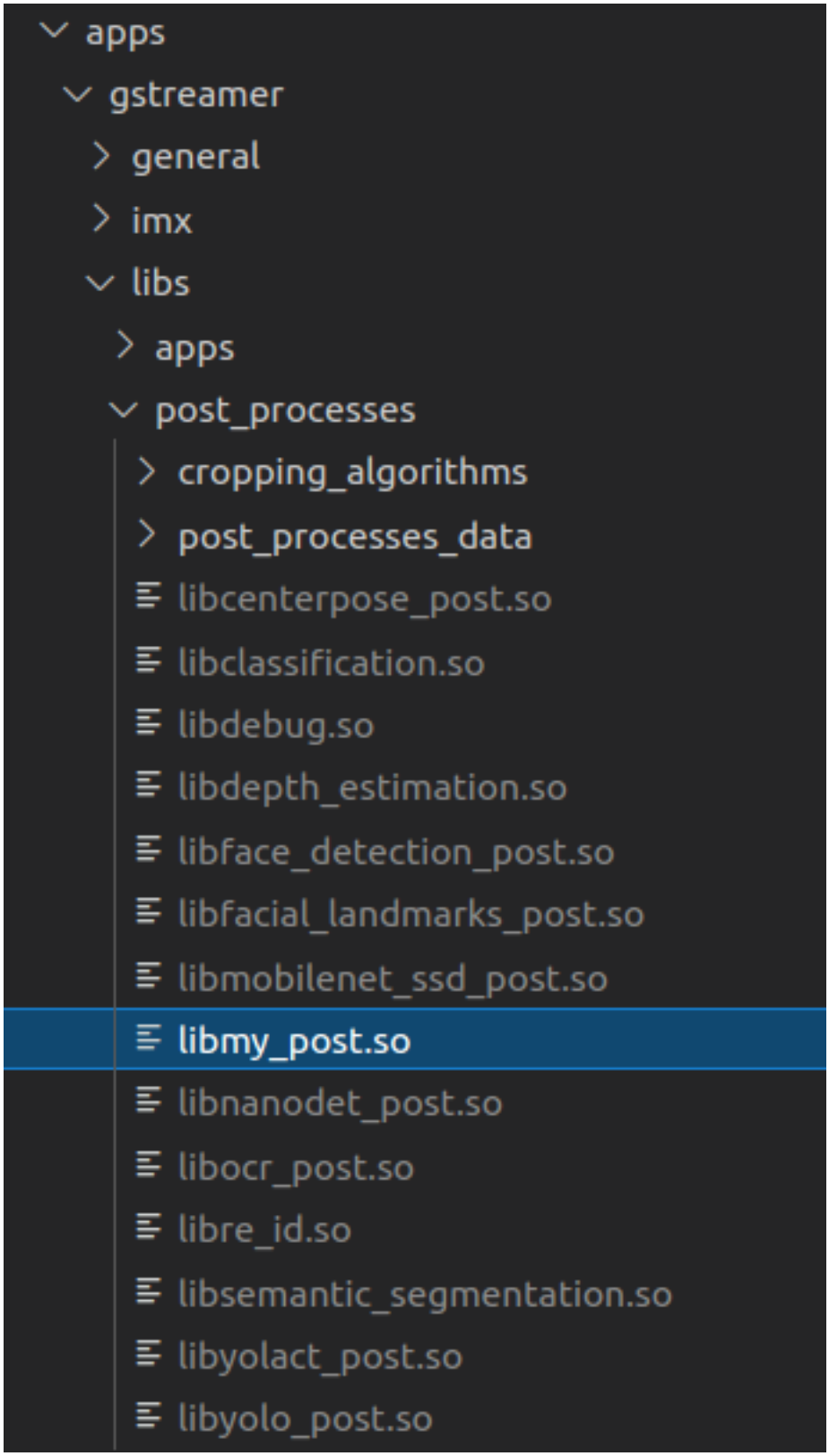
From the TAPPAS home directory folder you can run:

```
./scripts/gstreamer/install_hailo_gstreamer.sh
```



```
hailo@hailo_tappas:/local/workspace/tappas$ ./scripts/gstreamer/install_hailo_gstreamer.sh
/local/workspace/tappas/core/hailo/gstreamer /local/workspace/tappas
Directory already configured.

Just run your build command (e.g. ninja) and Meson will regenerate as necessary.
If ninja fails, run "ninja reconfigure" or "meson --reconfigure"
to force Meson to regenerate.

If build failures persist, run "meson setup --wipe" to rebuild from scratch
using the same options as passed when configuring the build.
To change option values, run "meson configure" instead.
ninja: Entering directory `build.release'
[0/1] Regenerating build files.
The Meson build system
Version: 0.58.0
Source dir: /local/workspace/tappas/core/hailo/gstreamer
Build dir: /local/workspace/tappas/core/hailo/gstreamer/build.release
Build type: native build
Project name: gst-hailo-tools
Project version: 3.16.0
C compiler for the host machine: gcc-9 (gcc 9.4.0 "gcc-9 (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0")
C linker for the host machine: gcc-9 ld.bfd 2.30
C++ compiler for the host machine: g++-9 (gcc 9.4.0 "g++-9 (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0")
C++ linker for the host machine: g++-9 ld.bfd 2.30
Host machine cpu family: x86_64
Host machine cpu: x86_64
Configuring config.h using configuration
Message: GCC >= 9.0.0 detected, applying extra arguments.
Dependency gstreamer-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-base-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-app-1.0 found: YES 1.14.5 (cached)
Dependency gstreamer-video-1.0 found: YES 1.14.5 (cached)
Dependency opencv4 found: YES 4.5.2 (cached)
Dependency glib-2.0 found: YES 2.56.4 (cached)
Dependency gmodule-2.0 found: YES 2.56.4 (cached)
Dependency gobject-2.0 found: YES 2.56.4 (cached)
Dependency pygobject-3.0 found: YES 3.26.1 (cached)
Program python3.6 found: YES (/usr/bin/python3.6)
Found pkg-config: /usr/bin/pkg-config (0.29.1)
Dependency python found: YES (pkgconfig)
Dependency python found: YES (pkgconfig)
Dependency blas found: YES 3.7.1 (cached)
Dependency lapack found: YES 3.7.1 (cached)
Library dl found: YES
Library stdc++fs found: YES
Dependency threads found: YES unknown (cached)
Build targets in project: 29
```

If all runs correctly a green `YES`, and our .so should appear in `apps/h8/gstreamer/libs/post_processes/`!



Running the .so

Now that the user has successfully compiled their first postprocess, they can continue to run the postprocess and view the results. Since it is still generic, run this test pipeline in the terminal to see if it works:

```
gst-launch-1.0 videotestsrc ! hailofilter so-path=$TAPPAS_WORKSPACE/apps/h8/
↪gstreamer/libs/post_processes/libmy_post.so ! fakesink
```

Note in the above pipeline that we gave the `hailofilter` the path to `libmy_post.so` in the `so-path` property. So now every time a buffer is received in that `hailofilter`'s sink pad, it calls the `filter()` function in `libmy_post.so`. The resulting app should print our chosen text `"My first postprocess!"` in the console:

```
hailo@hailo_tappas:/local/workspace/tappas$ gst-launch-1.0 videotestsrc ! hailofilter2 so-path=$TAPPAS_WORKSPACE/apps/gstreamer/x86/libs/libmy_post.so ! fakesink
Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
My first postprocess!
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
My first postprocess!
My first postprocess!
My first postprocess!
My first postprocess!
My first postprocess!
My first postprocess!
My first postprocess!
```

Filter Basics

Working with Tensors

Printing statements on every buffer is useful, however we would like a postprocess that can actually do operations on inference tensors. We will now describe how this can be achieved. Go back to `my_post.cpp` and replace the print statement with the following:

```
// Get the output layers from the hailo frame.
std::vector<HailoTensorPtr> tensors = roi->get_tensors();
```

The `HailoROI` has two ways of providing the output tensors of a network: via the `get_tensors()` and `get_tensor(std::string name)` functions. The first (which is used here) returns an `std::vector` of `HailoTensorPtr` objects. These are an `std::shared_ptr` to a `HailoTensor`: a class that represents an output tensor of a network. `HailoTensor` holds all kinds of important tensor metadata besides the data itself; such as the width, height, number of channels, and even quantization parameters. A full implementation for this class can be viewed at core/hailo/general/hailo_tensors.hpp. `get_tensor(std::string name)` also returns a `HailoTensorPtr`, but only the one with the given name output layer name. This can be convenient for performing operations on specific layers whose names are known in advance. Now that we have a vector of `HailoTensorPtr` objects, lets examine the information that can be obtained from it. Add the following lines to our `filter()` function:

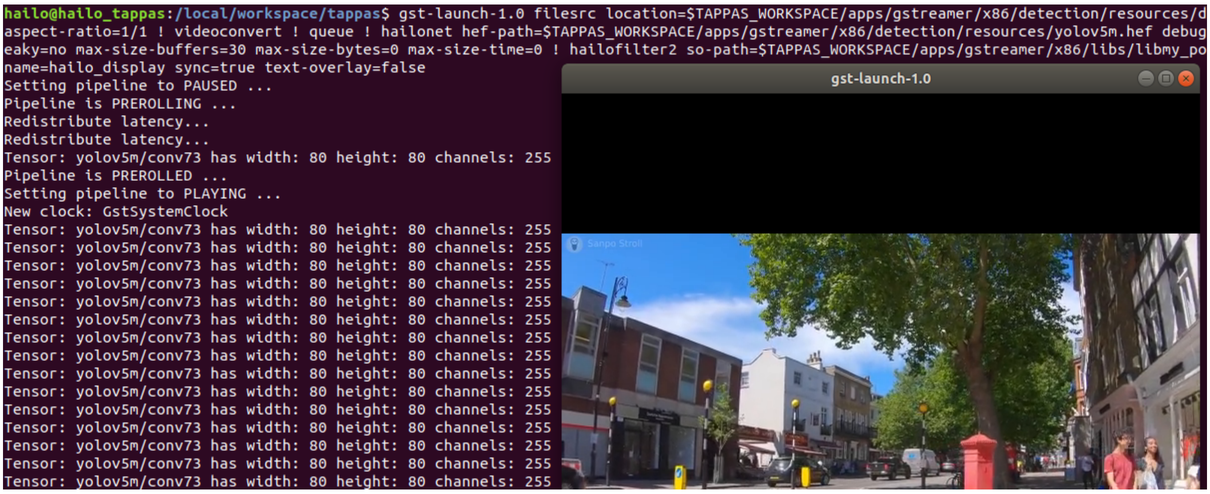
```
// Get the first output tensor
HailoTensorPtr first_tensor = tensors[0];
std::cout << "Tensor: " << first_tensor->name();
std::cout << " has width: " << first_tensor->shape()[0];
std::cout << " height: " << first_tensor->shape()[1];
std::cout << " channels: " << first_tensor->shape()[2] << std::endl;
```

Recompile with the same *script we used earlier*. Run a test pipeline, and this time see actual parameters of the tensor printed out:

```
gst-launch-1.0 filesrc location=$TAPPAS_WORKSPACE/apps/h8/gstreamer/general/
↪detection/resources/detection.mp4 name=src_0 ! decodebin ! videoscale ! video/x-
↪raw, pixel-aspect-ratio=1/1 ! videoconvert ! queue ! hailonet hef-path=$TAPPAS_
↪WORKSPACE/apps/h8/gstreamer/general/detection/resources/yolov5m_wo_spp_60p.hef
↪is-active=true ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! hailofilter so-path=$TAPPAS_WORKSPACE/apps/h8/gstreamer/libs/post_
↪processes/libmy_post.so qos=false ! videoconvert ! fpsdisplaysink video-
↪sink=ximagesink name=hailo_display sync=true text-overlay=false
```

(continues on next page)

(continued from previous page)



With a HailoTensorPtr the user has everything needed to perform postprocess operations. The actual tensor values can be accessed from the HailoTensorPtr with:

```
auto first_tensor_data = first_tensor->data();
```

Remember at this point the data is of type uint8_t, for full precision you will have to dequantize the tensor to a float. To aid this the quantization parameters (scale and zero point) are stored in the HailoTensorPtr and can be applied through tensor->fix_scale(uint8_t num).

Attaching Detection Objects to the Frame

Now that you know how to create a basic filter and access your inference tensor, we will learn how to add a detection object to the hailo_frame.Remove the prints from the filter() function and replace them with the following function call:

```
std::vector<HailoDetectionPtr> detections = demo_detection_objects();
```

Here the function is being called demo_detection_objects() which will return some detection objects. Copy the following function definition into my_post.cpp:

```
std::vector<HailoDetection> demo_detection_objects()
{
 std::vector<HailoDetection> objects; // The detection objects we will eventually return
 HailoDetection first_detection = HailoDetection(HailoBBox(0.2, 0.2, 0.2, 0.2), "person", 0.99);
 HailoDetection second_detection = HailoDetection(HailoBBox(0.6, 0.6, 0.2, 0.2), "person", 0.89);
 objects.push_back(first_detection);
 objects.push_back(second_detection);

 return objects;
}
```

In this function two instances of HailoDetection are being created and pushed into a vector that we return. Note that when creating a HailoDetection, we give a series of parameters. The expected parameters are as follows:

```
HailoDetection(HailoBBox bbox, const std::string &label, float confidence)
```


Where `HailoBBox` is a class that represents a bounding box, it is initialized as `HailoBBox(float xmin, float ymin, float width, float height)`.
NOTE: It is assumed that the `xmin, ymin, width, and height` given are a **percentage of the image size** (meaning, if the box is **half** as wide as the width of the image, then `width=0.5`). This protects the pipeline's ability to resize buffers without compromising the correct relative size of the detection boxes.

Looking back at the demo function we just introduced, we are adding two instances of `HailoDetection`: `first_detection` and `second_detection`. According to the parameters we saw, `first_detection` has an `xmin` 20% along the x axis, and a `ymin` 20% down the y axis. The `width` and `height` are also 20% of the image. The last two parameters, `label` and `confidence`, show that this instance has a 99% `confidence` for `label` person.

Now that we have a couple of `HailoDetections` available, add to them the original `HailoROIPtr`. There is a helper function we need in the core/hailo/general/hailo_common.hpp file that we included earlier in `my_post.hpp`.
This file has other features that will be useful, so it is recommended to keep the file readily available.
With the include in place, add the following function call to the end of the `filter()` function:

```
// Update the frame with the found detections.
hailo_common::add_detections(roi, detections);
```

This function takes a `HailoROIPtr` and a `HailoDetection` vector, then adds each `HailoDetection` to the `HailoROIPtr`. Now that our detections have been added to the `hailo_frame` our postprocess is done!
To recap, our whole `my_post.cpp` should look like this:

```
#include <iostream>
#include "my_post.hpp"

std::vector<HailoDetection> demo_detection_objects()
{
 std::vector<HailoDetection> objects; // The detection objects we will eventually
↪return
 HailoDetection first_detection = HailoDetection(HailoBBox(0.2, 0.2, 0.2, 0.2),
↪"person", 0.99);
 HailoDetection second_detection = HailoDetection(HailoBBox(0.6, 0.6, 0.2, 0.2),
↪"person", 0.89);
 objects.push_back(first_detection);
 objects.push_back(second_detection);
 return objects;
}

// Default filter function
void filter(HailoROIPtr roi)
{
  std::vector<HailoTensorPtr> tensors = roi->get_tensors();

  std::vector<HailoDetection> detections = demo_detection_objects();
  hailo_common::add_detections(roi, detections);
}
```

Recompile again and run the test pipeline, if all is correct then you should see the original video run with no problems. If you are unable to see any detections this is because they are attached to each buffer, however no overlay is drawing them onto the image itself. To see how our detection boxes can be drawn, read further in *Next Steps Drawing*.

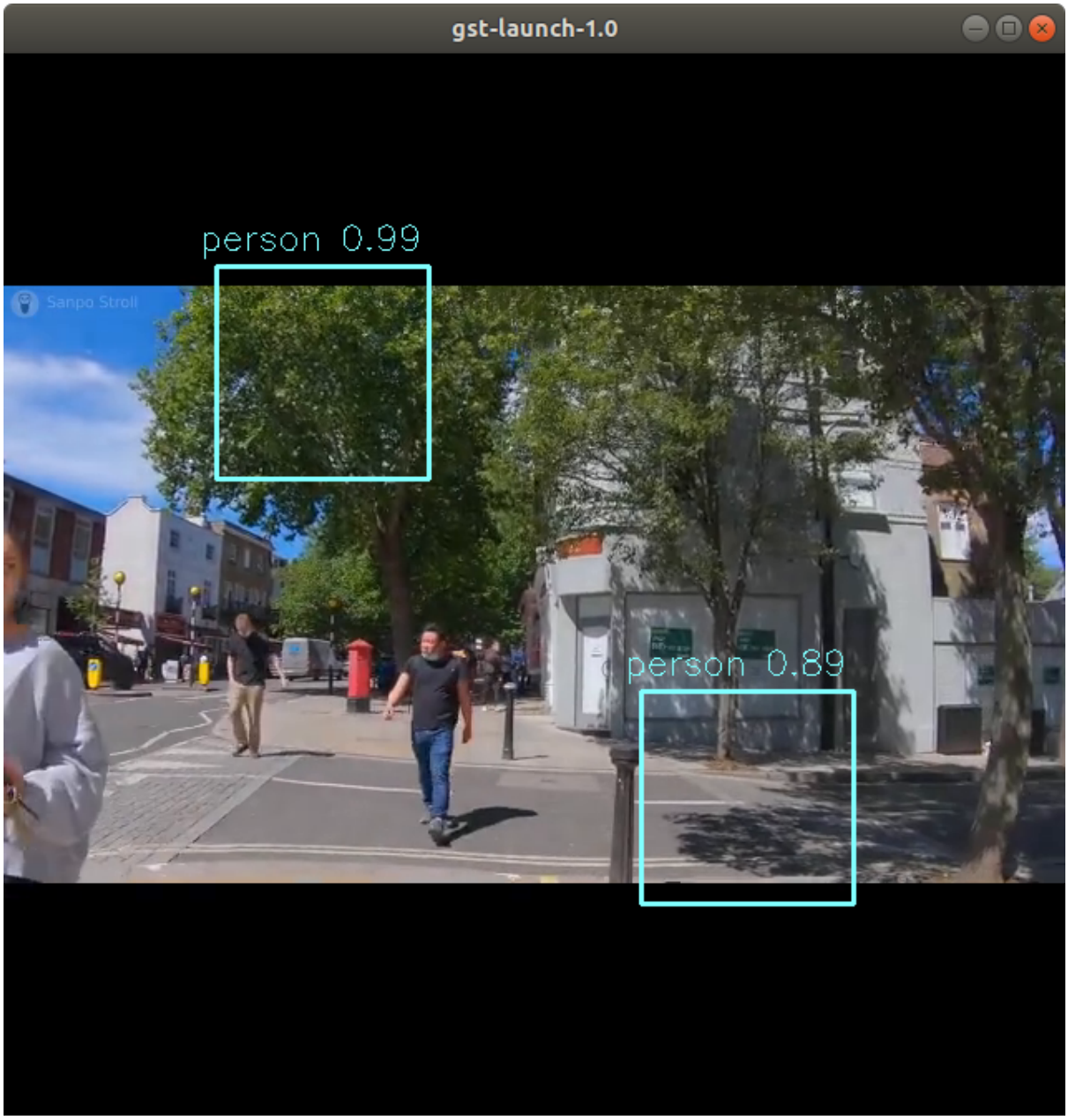
Next Steps

Drawing

At this point we have a working postprocess that attaches two detection boxes to each passing buffer. But to get the GStreamer pipeline to draw those boxes onto the image, We provide a GStreamer element - *hailooverlay* - that draws any Hailo provided output classes (detections, classifications, landmarks, etc..) on the buffer, to do this include it in your pipeline.
The element should be added in the pipeline after the `hailofilter` element with our postprocess.
Now the pipeline should look like:

```
gst-launch-1.0 filesrc location=$TAPPAS_WORKSPACE/apps/h8/gstreamer/general/
↪detection/resources/detection.mp4 name=src_0 ! decodebin ! videoscale ! video/x-
↪raw, pixel-aspect-ratio=1/1 ! videoconvert ! queue ! hailonet hef-path=$TAPPAS_
↪WORKSPACE/apps/h8/gstreamer/general/detection/resources/yolov5m_wo_spp_60p.hef
↪is-active=true ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! hailofilter so-path=$TAPPAS_WORKSPACE/apps/h8/gstreamer/libs/post_
↪processes/libmy_post.so qos=false ! queue ! hailooverlay ! videoconvert !
↪fpsdisplaysink video-sink=ximagesink name=hailo_display sync=true text-
↪overlay=false
```

Run the expanded pipeline above to see the original video, but this time with the two detection boxes we added.



Both boxes will be labeled as `person`, and each is shown with the assigned `confidence`. Obviously, the two boxes don't move or match any object in the video; this is because for the benefit of this tutorial the values are hardcoded. It is up to the user to extract the correct numbers from the inferred tensor of their network, as can be seen among the postprocesses already implemented in the TAPPAS each network can be different. This guide provides a strong starting point for further development.

Multiple Filters in One .so

While the `hailofilter` always calls on a `filter()` function by default, the user can provide the element access to other functions in `.so` to call instead. This may be of interest for developing a postprocess that applies to mutliple networks, but each network needs slightly different starting parameters (in the TAPPAS case, mutliple flavors of the Yolo detection network are handled via the same .so). This can be achieved by declaring the extra functions in the header file, then pointing the `hailofilter` to that function via the `function-name` property. Taking the Yolo networks as an example, open up libs/postprocesses/detection/yolo_postprocess.hpp to see what functions are made available to the `hailofilter`:


```
#pragma once
#include "hailo_objects.hpp"
#include "hailo_common.hpp"


__BEGIN_DECLS
void filter(HailoROIPtr roi);
void yolov5(HailoROIPtr roi, void *params_void_ptr);
void yolox(HailoROIPtr roi, void *params_void_ptr);
void yoloxx(HailoROIPtr roi, void *params_void_ptr);
void yolov3(HailoROIPtr roi, void *params_void_ptr);
void yolov4(HailoROIPtr roi, void *params_void_ptr);
void tiny_yolov4_license_plates(HailoROIPtr roi, void *params_void_ptr);
void yolov5_no_persons(HailoROIPtr roi, void *params_void_ptr);
void yolov5_no_faces(HailoROIPtr roi, void *params_void_ptr);
void yolov5_counter(HailoROIPtr roi, void *params_void_ptr);
void yolov5_vehicles_only(HailoROIPtr roi, void *params_void_ptr);
void yolov5_personface(HailoROIPtr roi, void *params_void_ptr);
void yolov5_personface_letterbox(HailoROIPtr roi, void *params_void_ptr);
void yolov5_no_faces_letterbox(HailoROIPtr roi, void *params_void_ptr); //should
↪add to python too
void yolov5_adas(HailoROIPtr roi, void *params_void_ptr);
__END_DECLS
```

Any of the functions declared here can be given as a `function-name` property to the `hailofilter` element. Condsider this pipeline for running the `Yolov5` network:

```
gst-launch-1.0 filesrc location=/local/workspace/tappas/apps/h8/gstreamer/general/
↪detection/resources/detection.mp4 name=src_0 ! decodebin ! videoscale ! video/x-
↪raw, pixel-aspect-ratio=1/1 ! videoconvert ! queue leaky=no max-size-buffers=30
↪max-size-bytes=0 max-size-time=0 ! hailonet hef-path=/local/workspace/tappas/
↪apps/h8/gstreamer/general/detection/resources/yolov5m_wo_spp_60p.hef is-
↪active=true ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0
↪! hailofilter function-name=yolov5 so-path=/local/workspace/tappas/apps/h8/
↪gstreamer/libs/post_processes//libyolo_post.so qos=false ! queue leaky=no max-
↪size-buffers=30 max-size-bytes=0 max-size-time=0 ! hailooverlay ! videoconvert !
↪fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=false text-
↪overlay=false
```

The `hailofilter` above that performs the post-process points to `libyolo_post.so` in the `so-path`, but it also includes the property `function-name=yolov5`. This lets the `hailofilter` know that instead of the default `filter()` function it should call on the `yolov5` function instead.

5.3.6. Writing Your Own Python Postprocess

Overview

To add a network to the TAPPAS that is not already supported, requires the implementation of a new post-process. Fortunately with the use of the *hailopython*, there is no need to create any new GStreamer elements, only to provide a Python module that applies your post-processing! This section will review how to create a python module and what mechanisms/structures are available for creating a post-process.

Getting Started

hailopython requires a module and a Python function.

Python Module Template

Below is a template for a Python module for the hailopython element.

```
import hailo

# Importing VideoFrame before importing GST is must
from gsthailo import VideoFrame
from gi.repository import Gst

# Create 'run' function, that accepts one parameter - VideoFrame, more about
↪VideoFrame later.
# `run` is default function name if no name is provided
def run(video_frame: VideoFrame):
  print("My first Python postprocess!")

  return Gst.FlowReturn.OK
```

To call it, create a pipeline with hailopython:

```
gst-launch-1.0 videotestsrc ! hailopython module=$PATH_TO_MODULE/my_module.py !
↪autovideosink
```

Extracting the Tensors

One of the first steps in each post-process is to get the output tensors. This can be done by one of two methods `video_frame.roi.get_tensor(tensor_name)` or `video_frame.roi.get_tensors()`

```
def run(video_frame: VideoFrame):
  for tensor in video_frame.roi.get_tensors():
    print(tensor.name())
  my_tensor = roi.get_tensor("output_layer_name")
  print(f"shape is {my_tensor.height()}X{my_tensor.width()}X{my_tensor.features()}
↪)
```

After doing this it is possible to convert this object of type HailoTensor to a numpy array on which perform post-processing operations can be performed more conveniently. This is a fairly simple step, just use np.array on a given HailoTensor.

> Notice that np.array has a parameter that determines whether to copy the memory or use the original buffer.

```
def run(video_frame: VideoFrame):
  my_tensor = roi.get_tensor("output_layer_name")
  # To create a numpy array with new memory
  my_array = np.array(my_tensor)
  # To create a numpy array with original memory
  my_array = np.array(my_tensor, copy=False)
```

There are some other methods in HailoTensor, that can be performed `dir(my_tensor)` or `help(my_tensor)`.

Adding Results

After processing the net results and obtaining the post-processed results, they can be used as preferred. Below demonstrates how to add them to the original image in order to draw them later with the hailooverlay element. In order to add the post-processed result to the original image - use the `video_frame.roi.add_object` method. This method adds a HailoObject object to the image. There are several types of objects that are currently supported: hailo.HailoClassification - Classification of the image. hailo.HailoDetection - Detection in the image. hailo.HailoLandmarks - Landmarks in the image.

It is possible to create one of these objects and then add it with the roi.add_object method.

```
def run(video_frame: VideoFrame):
  classification = hailo.HailoClassification(type='animal', index=1, label='horse',
↪ confidence=0.67)

  # You can also create a classification without class id (index).
  classification = hailo.HailoClassification(type='animal', label='horse',
↪confidence=0.67)

  video_frame.roi.add_object(classification)
```

One can also add objects to detections:

```
def run(video_frame: VideoFrame):
  # Adds a person detection in the bottom right quarter of the image. (normalized only)
  person_bbox = hailo.HailoBBox(xmin=0.5, ymin=0.5, width=0.5, height=0.5)
  person = hailo.HailoDetection(bbox=person_bbox, label='person', confidence=0.97)
  video_frame.roi.add_object(person)

  # Now, Adds a face to the person, at the top of the person. (normalized only)
  face_bbox  = hailo.HailoBBox(xmin=0.0, ymin=0.0, width=1, height=0.2)
  face = hailo.HailoDetection(bbox=face_bbox, label='face', confidence=0.84)
  person.add_object(face)
  # No need to add the face to the roi because it is already in the person that is in the
↪roi.
```

Next Steps

Drawing

In order to draw the post-processed results on the original image use the hailooverlay element. It is already familiar with our HailoObject types and knows how to draw classifications, detections, and landmarks onto the image.

```
gst-launch-1.0 filesrc location=$TAPPAS_WORKSPACE/apps/h8/gstreamer/general/
↪detection/detection.mp4 name=src_0 ! decodebin \
! videoscale ! video/x-raw, pixel-aspect-ratio=1/1 ! videoconvert ! queue leaky=no
↪max-size-buffers=30 \
max-size-bytes=0 max-size-time=0 ! hailonet hef-path=$TAPPAS_WORKSPACE/apps/h8/
↪gstreamer/general/detection/yolov5m_wo_spp_60p.hef \
is-active=true ! queue leaky=no max-size-buffers=30 max-size-bytes=0 \
max-size-time=0 ! hailopython module=$TAPPAS_WORKSPACE/apps/h8/gstreamer/general/
↪detection/my_module.py qos=false ! queue \
leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! hailooverlay
↪qos=false ! videoconvert ! \
fpsdisplaysink video-sink=ximagesink name=hailo_display sync=true text-
↪overlay=false
```

This is the standard detection pipeline with a python module for post-processing.

Multiple Functions in One Python Module

There is an option to write several post-process functions in the same module. In order to run each of them just add the `function` property to the `hailopython` element:

```
import hailo

# Importing VideoFrame before importing GST is must
from gsthailo import VideoFrame
from gi.repository import Gst


def post_process_function(video_frame: VideoFrame):
  print("My first Python postprocess!")

def other_post_function(video_frame: VideoFrame):
  print("Other Python postprocess!")
```

```
gst-launch-1.0 videotestsrc ! hailopython module=$PATH_TO_MODULE/my_module.py
↪function=other_post_function ! autovideosink
```

VideoFrame Class

In addition to providing `buffer` and `HailoROI` access functions, the `VideoFrame` module provides helper functions for accessing the buffer through NumPy

List all Available Methods and Members

Running the following command would display a list of methods and members available:

```
python3 -c "import hailo; help(hailo)"
```

5.3.7. Retraining TAPPAS Models

Instructions

If you wish to use a TAPPAS pipeline/demo with your own model, the easiest solution is:

1. **Read the relevant TAPPAS pipeline/demo README page:**
    - It lists the ModelZoo models that are used on this pipeline
    - If Model Zoo retraining dockers are available for those models, links are given; In addition it describes how to reconfigure TAPPAS for the retrained models
2. Follow the retraining docker's instructions to retrain the model with your own data
3. Follow the retraining docker's instructions to compile the network
4. Reconfigure TAPPAS scripts to run your new .hef file(s)

Notes

- Easy post-processing JSON reconfiguration is currently only available for YOLO architecures. For other architectures, recreating post-processing .so file is required. The relevant code and header files are mentioned on the README pages.
- Models that use on-chip RGBX->RGB layers does not appear yet on the ModelZoo. Therefore, many models that are used for iMX demos (and others; see on the README of each app) should be added those layers manually to create models that will fit TAPPAS:
  - Use the non-rgbx network from the above table for retraining and compilation, with one simple modification - On the model alls file (on hailo_model_zoo/cfg/alls), add **right after normalization command**:

```
reshape_rgb = input_conversion(input_layer1, tf_rgbx_to_hailo_rgb)
```


6. Elements

6.1. Hailo Cropper

6.1.1. Overview

`HailoCropper` is an element providing cropping functionality, designed for application with cascading networks, meaning doing one task based on a previous task. It has 1 sink and 2 sources. `HailoCropper` receives a frame on its sink pad, then invokes it's `prepare_crops` method that returns the vector of crop regions of interest (crop_roi), For each crop_roi it creates a cropped image (representing it's x, y, width, height in the full frame). The cropped images are then sent to the second src. From the first src we push the original frame that the detections were cropped from.

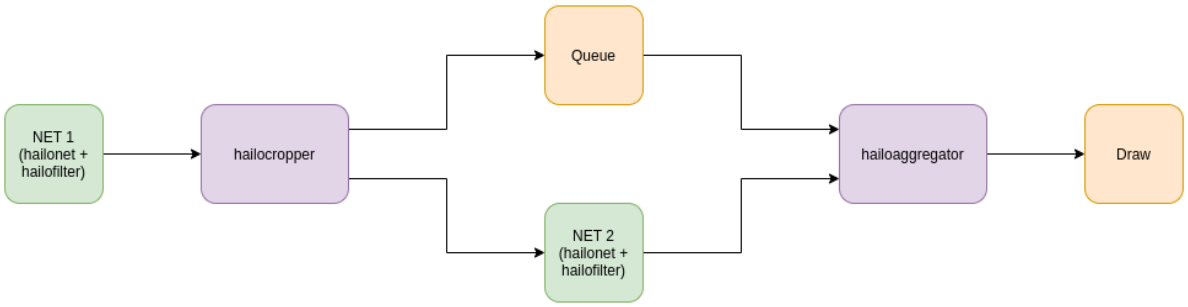
By default, `HailoCropper` receives a video frame that has detections (means a previous `HailoNet` + `HailoFilter` ran) on it's sinkpad. For each detection it creates a cropped image (using a specific algorithm to create a scaled image with the same aspect ratio). This is used by the cascading networks app *Face Landmarks based on Face Detection*.

Derived classes can override the default `prepare_crops` behaviour and decide where to crop and how many times. *hailotilecropper* element does this exact thing when splitting the frame into tiles by rows and columns.

Parameters

There is only one property for this element other than the common 'name' and 'parent'. The name of this boolean property is 'internal-offset' and it is used to determine whether we use the original offset* of the buffer or overwrite it with our own offset. The offset of the buffer is given to the original buffer and all the crops, and used by the hailoaggregator, to make sure the cropped detections we are 'muxing' with the original buffer are actually from the same buffer.*Offset is an attribute of buffer that determines on what offset this buffer is since the start of the pipeline run, represented by number of buffers. It's similar to frame-id in video. On some videos the offset attribute is not created by the filesrc element and it is set to -1 (casted to uint64), therefore if we want to use it to determine what the current frame is, we should somehow track the number of buffers and set this offset accordingly.

6.1.2. Example



6.1.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoCropper

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src_0'
  Pad Template: 'src'
 SRC: 'src_1'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
            flags: readable, writable
            String. Default: "hailocropper0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 internal-offset   : Whether to use Gstreamer offset of internal offset.
            flags: readable, writable, controllable
            Boolean. Default: false
```

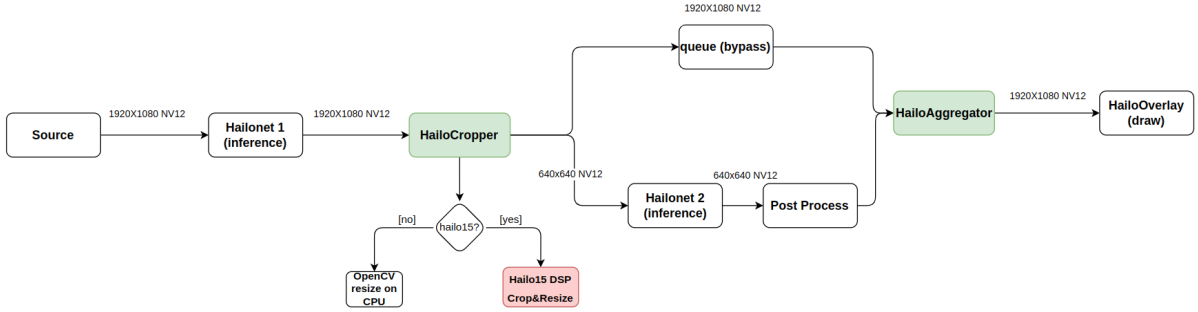
6.1.4. Hailo-15

HailoCropper can utilize the on-chip DSP (Digital Signal Processor), to perform resize and crop operations.

The DSP is used by default on the Hailo-15 machine, and can be disabled by setting the `use-dsp` property. When disabled OpenCV will be used (on the CPU) to perform the resize and crop operations.

HailoCropper holds a buffer pool (GstBufferPool) that manages the buffers, used by the DSP. The buffer pool is responsible for allocating and freeing the buffers when the reference count of a buffer reaches 0. Buffer pool size (maximum buffers that can be allocated simultaneously in the pool) can be controlled using the `pool-size` property.

It is recommended to make sure that a buffer is contiguous in memory, before being sent to the DSP. Non-contiguous buffers will be copied to a new buffer, before being sent to the DSP.



6.2. Hailo Overlay

6.2.1. Overview

HailoOverlay is a drawing element that can draw postprocessed results on an incoming video frame. This element supports the following results:

- Detection - Draws the rectengle over the frame, with the label and confidence (rounded).
- Classification - Draws a classification over the frame, at the top left corner of the frame.
- Landmarks - Draws a set of points on the given frame at the wanted coordintates.
- Tiles - Can draw tiles as a thin rectengle.

Parameters

As a member of the GstBaseTransform hierarchy, the hailooverlay element supports qos (Quality of Service). Although qos typically tries to guarantee some level of performance, it can lead to frames dropping. For this reason it is advised to always set `qos=false` to avoid either tensors being dropped or not drawn.

6.2.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstBaseTransform
             +----GstHailoOverlay

Pad Templates:
 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB }
```

(continues on next page)

(continued from previous page)

```
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
       framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
            flags: readable, writable
            String. Default: "hailooverlay0"
 parent      : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 qos         : Handle Quality-of-Service events
            flags: readable, writable
            Boolean. Default: false
```

6.3. Hailo DeviceStats

6.3.1. Overview

Hailodevicestats is an element that samples power and temperature. It doesn't have any pads, it just has to be part of the pipeline. An example for using this element could be found under the `detection` / `multi-stream_multidevice` app.

Parameters

Determine the time period between samples with the `interval` property.

Choose device with the `device-id` property.

6.3.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoDeviceStats


Factory Details:
 Rank          primary (256)
 Long-name        hailodevicestats element
 Klass          Hailo/Device
 Description       Log Hailo-8 device statistics
 Author         Omer Salem <omers@hailo.ai>
```

(continues on next page)

(continued from previous page)

```
Pad Templates:

 none


Element has no clocking capabilities.


Element has no URI handling capabilities.


Pads:

 none


Properties:

 name         : The name of the object
            flags: readable, writable
            String. Default: "hailodevicestats0"

 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"

 interval      : Time period between samples, in seconds
            flags: readable, writable
            Unsigned Integer. Range: 0 - 4294967295 Default: 1

 device-id      : Device ID ([<domain>]:<bus>:<device>.<func>, same as in lspci
↪command)
            flags: readable, writable
            String. Default: null

 silent       : Should print statistics
            flags: readable, writable
            Boolean. Default: false

 power-measurement  : Current power measurement of device
            flags: readable
            Float. Range:        0 -  3.402823e+38 Default:        0

 temperature     : Current temperature of device
            flags: readable
            Float. Range:        0 -  3.402823e+38 Default:        0
```

6.4. Hailo Filter

6.4.1. Overview

Hailofilter is an element which enables the user to apply a postprocess operation on hailonet's output tensors. It provides an entry point for a compiled .so file that the user writes, inside of which they will have access to the original image frame, the tensors output by the network for that frame, and any metadata attached. At first the hailofilter will read the buffer from the sink pad, then apply the filter defined in the provided .so, until finally sending the filtered buffer along the source pad to continue down the pipeline.

Parameters

The most important parameter here is the `so-path`. Here the user provides the path to your compiled .so that applies your wanted filter. By default, the hailofilter will call on a filter() function within the .so as the entry point. If your .so has multiple entry points, for example in the case of slightly different network flavors, then you can chose which specific filter function to apply via the `function-name` parameter. As a member of the GstVideoFilter hierarchy, the hailofilter element supports qos (Quality of Service). Although qos typically tries to garuantee some level of performance, it can lead to frames dropping. For this reason it is advised to always set `qos=false` to avoid either tensors being dropped or not drawn.

6.4.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstBaseTransform
             +----GstHailofilter

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
            flags: readable, writable
            String. Default: "hailofilter-0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 qos          : Handle Quality-of-Service events
            flags: readable, writable
            Boolean. Default: false
 so-path      : Location of the so file to load
            flags: readable, writable, changeable only in NULL or READY state
            String. Default: null
 function-name    : function-name
            flags: readable, writable, changeable only in NULL or READY state
            String. Default: "filter"
 use-gst-buffer    : use function with access to the Gst Buffer
            flags: readable, writable, controllable
            Boolean. Default: false
```


6.5. Hailo Tracker

6.5.1. Overview

HailoTracker is an element which enables the user to track *HailoDetection* objects using the Joint-Detection-and-Embedding (JDE) Tracking algorithm. JDE is a fast and high-performance multiple-object tracking algorithm that associates between detections in consecutive frames based on size/location/movement. Associations are made by linear assignment via a Kalman Filter.
For more information on JDE Tracking and Kalman Filtering read here.

Parameters

The hailotracker element provides a series of properties that allow you to adjust the tracking algorithm. The most important property to set is `class-id`: this determines if the tracker will track all *HailoDetection* objects indiscriminately of class or focus only on detections of a specific class id (the default behavior is to track across-classes).

6.5.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstBaseTransform
             +----GstVideoFilter
                +----GstHailoTracker

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
            flags: readable, writable
            String. Default: "hailotracker0"
 parent      : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 qos         : Handle Quality-of-Service events
            flags: readable, writable
            Boolean. Default: true
 debug       : Enable debug mode.
```

(continues on next page)

(continued from previous page)

```
            flags: readable, writable, controllable
            Boolean. Default: false
class-id        : The class id of the class to track. Default -1 crosses classes.
            flags: readable, writable, changeable only in NULL or READY state
            Integer. Range: -2147483648 - 2147483647 Default: -1
kalman-dist-thr    : Threshold used in Kalman filter to compare Mahalanobis cost
↪matrix. Closer to 1.0 is looser.
            flags: readable, writable, controllable
            Float. Range:        0 -        1 Default:       0.7
iou-thr         : Threshold used in Kalman filter to compare IOU cost matrix. Closer to
↪1.0 is looser.
            flags: readable, writable, controllable
            Float. Range:        0 -        1 Default:       0.8
init-iou-thr     : Threshold used in Kalman filter to compare IOU cost matrix of newly
↪found instances. Closer to 1.0 is looser.
            flags: readable, writable, controllable
            Float. Range:        0 -        1 Default:       0.9
keep-tracked-frames : Number of frames to keep without a successful match before a
↪'tracked' instance is considered 'lost'.
            flags: readable, writable, controllable
            Integer. Range: 0 - 2147483647 Default: 2
keep-new-frames    : Number of frames to keep without a successful match before a 'new'
↪instance is removed from the tracking record.
            flags: readable, writable, controllable
            Integer. Range: 0 - 2147483647 Default: 2
keep-lost-frames   : Number of frames to keep without a successful match before a 'lost
↪' instance is removed from the tracking record.
            flags: readable, writable, controllable
            Integer. Range: 0 - 2147483647 Default: 2
```

6.6. Hailo Tile Aggregator

6.6.1. Overview

`HailoTileAggregator` is a derived element of *hailoAggregator* and it is used in the *Tiling* app. A complement to the *HailoTileCropper*, the two elements work together to form a versatile tiling apps.

The element extends two methods of the parent element:

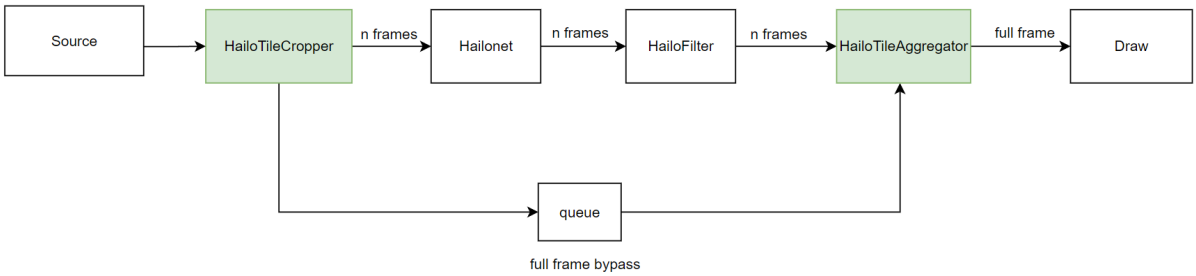
- `handle_sub_frame_roi`: Functionality to perform for each incoming sub frame. .. code-block:

  ```
  Performs ``remove_exceeded_bboxes`` (remove boxes close to boundary - using
  ↪given border_threshold) and then parent element performs flatten detections.
  ```

- `post_aggregation`: Functionality to perform after all frames are aggregated succesfully. .. code-block:

  ```
  Performs ``remove_large_landscape`` and ``NMS``.
  ```


6.6.2. Example



6.6.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoAggregator
             +----GstHailoTileAggregator

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink_0'
  Pad Template: 'sink'
 SINK: 'sink_1'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
            flags: readable, writable
            String. Default: "hailotileaggregator0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 flatten-detections : perform a 'flattening' functionality on the detection metadata
↪when receiving each frame
            flags: readable, writable, changeable only in NULL or READY state
            Boolean. Default: false
 iou-threshold    : threshold
            flags: readable, writable, changeable only in NULL or READY state
            Float. Range:        0 -        1 Default:        0.3
 border-threshold   : border threshold
            flags: readable, writable, changeable only in NULL or READY state
```

(continues on next page)

(continued from previous page)

```
            Float. Range:        0 -        1 Default:        0.1
 remove-large-landscape: remove large landscape objects when running in multi-scale
↪mode
            flags: readable, writable, changeable only in NULL or READY state
```

6.7. Hailo Tile Cropper

6.7.1. Overview

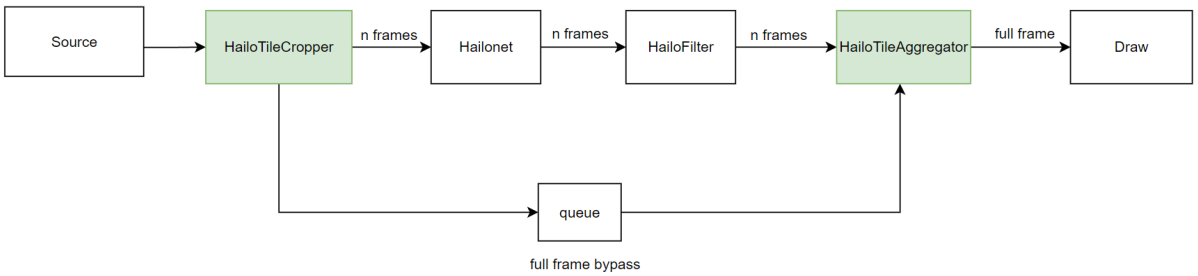
`HailoTileCropper` is a derived element of *hailoCropper* and it is used in the *Tiling* app. It overrides the default `prepare_crops` behaviour to return a vector of tile regions of interest, and allows splitting the incoming frame into tiles by rows and columns. Each tile stores their x, y, width, and height (with overlap between tiles included) in the full frame. Just like the base HailoCropper, the full original frame is sent to the first src pad while all the cropped images are sent to the second.

hailoaggregator wiil aggregate the cropped tiles and stitch them back to the original resolution.

Parameters

- tiles-along-x-axis : Number of tiles along x axis (columns) - default 2
- tiles-along-y-axis : Number of tiles along x axis (rows) - default 2
- overlap-x-axis : Overlap in percentage between tiles along x axis (columns) - default 0
- overlap-y-axis : Overlap in percentage between tiles along y axis (rows) - default 0
- tiling-mode : Tiling mode (0 - single-scale, 1 - multi-scale) - default 0
- scale-level : Scales (layers of tiles) in addition to the main layer 1: [(1 X 1)] 2: [(1 X 1), (2 X 2)] 3: [(1 X 1), (2 X 2), (3 X 3)]] - default 2

6.7.2. Example



6.7.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoBaseCropper
             +----GstHailoTileCropper
Pad Templates:
 SRC template: 'src'
```

(continues on next page)

(continued from previous page)

```
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB, (string)YUY2 }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB, (string)YUY2 }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src_0'
  Pad Template: 'src'
 SRC: 'src_1'
  Pad Template: 'src'

Element Properties:
 name         : The name of the object
            flags: readable, writable
            String. Default: "hailotilecropper0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 internal-offset   :
            Whether to use Gstreamer offset of internal offset.
            NOTE: If using file sources, Gstreamer does not generate offsets for
↪buffers,
            so this property should be set to true in such cases.
            flags: readable, writable, controllable
            Boolean. Default: false
 tiles-along-x-axis  : Number of tiles along x axis (columns)
            flags: readable, writable, changeable only in NULL or READY state
            Unsigned Integer. Range: 1 - 20 Default: 2
 tiles-along-y-axis  : Number of tiles along x axis (rows)
            flags: readable, writable, changeable only in NULL or READY state
            Unsigned Integer. Range: 1 - 20 Default: 2
 overlap-x-axis    : Overlap in percentage between tiles along x axis (columns)
            flags: readable, writable, changeable only in NULL or READY state
            Float. Range:        0 -        1 Default:        0
 overlap-y-axis    : Overlap in percentage between tiles along y axis (rows)
            flags: readable, writable, changeable only in NULL or READY state
            Float. Range:        0 -        1 Default:        0
 tiling-mode     : Tiling mode
            flags: readable, writable
            Enum "GstHailoTileCropperTilingMode" Default: 0, "single-scale"
              (0): single-scale   - Single Scale
              (1): multi-scale    - Multi Scale
 scale-level     : 1: [(1 X 1)] 2: [(1 X 1), (2 X 2)] 3: [(1 X 1), (2 X 2), (3 X 3)]]
            flags: readable, writable, changeable only in NULL or READY state
```

(continues on next page)

(continued from previous page)

```
            Unsigned Integer. Range: 1 - 3 Default: 2
```

6.8. Hailo Python

6.8.1. Overview

HailoPython is an element which enables the user to apply processing operations to an image via python. It provides an entry point for a python module that the user writes, inside of which they will have access to the Hailo raw-output (output tensors) and postprocessed-outputs (detections, classifications etc..) as well as the gstreamer buffer. The python function will be called for each buffer going through the hailopython element.

Parameters

The two parameters that define the function to call are `module` and `function` for the module path and function name respectively. In addition, as a member of the GstVideoFilter hierarchy, the hailofilter element supports qos (Quality of Service). Although qos typically tries to guarantee some level of performance, it can lead to frames dropping. For this reason it is advised to always set `qos=false` to avoid either tensors being dropped or not drawn.

6.8.2. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstBaseTransform
             +----GstVideoFilter
                +----GstHailoPython

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB, (string)YUY2 }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
       framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
         format: { (string)RGB, (string)YUY2 }
          width: [ 1, 2147483647 ]
         height: [ 1, 2147483647 ]
       framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'
```

(continues on next page)

(continued from previous page)

```
Element Properties:
 name        : The name of the object
              flags: readable, writable
              String. Default: "hailopython0"
 parent      : The parent of the object
              flags: readable, writable
              Object of type "GstObject"
 qos         : Handle Quality-of-Service events
              flags: readable, writable
              Boolean. Default: true
 module      : Python module name
              flags: readable, writable
              String. Default: "/local/workspace/tappas/processor.py"
 function    : Python function name
              flags: readable, writable
              String. Default: "run"
```

6.9. Hailo Net

For documentation, please refer to HailoRT

6.10. Hailo Muxer

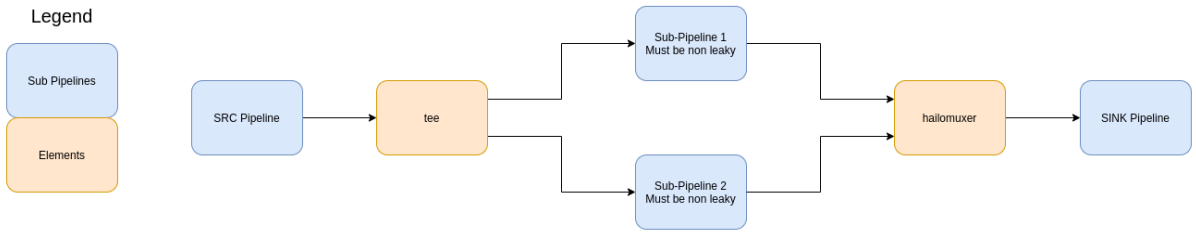
6.10.1. Overview

HailoMuxer is an element designed for our new multi-device application. It muxes 2 similar streams into 1 stream, holding both stream's metadata. It has 2 src elements and 1 sink, and whenever there are buffers on both src pads, it takes only 1 of the buffers and passes it on, with both buffer's metadata.

Parameters

There are no unique properties to hailomuxer. The only parameters are the baseclass parameters, which are 'name' and 'parent'.

6.10.2. Example



6.10.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoMuxer

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink_%u'
  Availability: On request
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
            flags: readable, writable
            String. Default: "hailomuxer0"
 parent      : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
```

6.11. Hailo Aggregator

6.11.1. Overview

HailoAggregator is an element designed for applications with cascading networks or cropping functionality, meaning to perform one task based on a previous task. It is a complement to the *HailoCropper*, the two elements work together to form versatile apps. It has 2 sink pads and 1 source: the first sinkpad receives the original frame from an upstream hailocropper, while the other receives cropped buffers from the other hailocropper. The HailoAggregator waits for all crops of a given orignal frame to arrive, then sends the original buffer with the combined metadata of all collected crops.

HailoAggregator also performs a 'flattening' functionality on the detection metadata when receiving each frame: detections are taken from the cropped frame, copied to the main frame and re-scaled/moved to their corresponding location in the main frame (x,y,width,height). As an example:

- *Face Landmarks based on Face Detection* - HailoCropper crops each face detection -> HailoNet + FaceLandmarks post for each face -> HailoAggregator aggregates the frames back.
- *Tiling* - *hailotilecropper* crops the image to tiles -> HailoNet + Detection post for each tile -> HailoAggregator aggregates the frames back and 'flatten' the detection objects in the metadata.

HailoAggregator exports two methods to extend or override in derived elements:

- `handle_sub_frame_roi`: Functionality to perform for each incoming sub frame.


```
Calls `flattening` method.
```

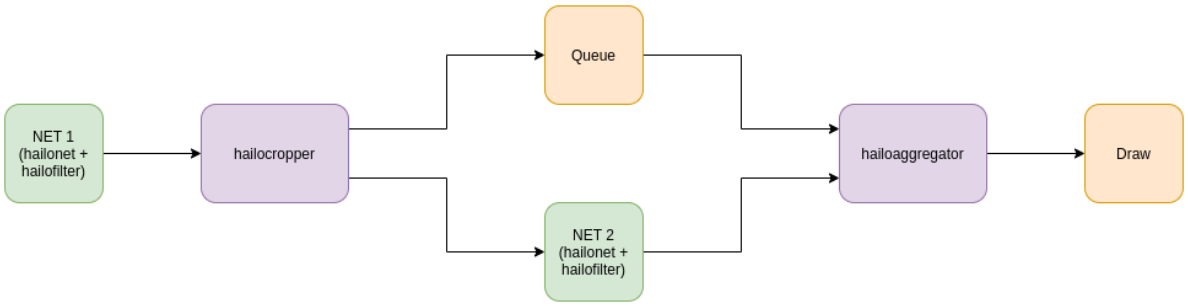
- `post_aggregation`: Functionality to perform after all frames are aggregated succesfully.

```
Base implementation does nothing.
```

Parameters

There are no unique properties to hailoaggregator. The only parameters are the baseclass parameters, which are 'name' and 'parent'.

6.11.2. Example



6.11.3. Hierarchy

```
GObject
 +----GInitiallyUnowned
    +----GstObject
       +----GstElement
          +----GstHailoAggregator

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink_0'
  Pad Template: 'sink'
 SINK: 'sink_1'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
```

(continues on next page)

(continued from previous page)

```
            flags: readable, writable
            String. Default: "hailoaggregator0"
 parent       : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 flatten-detections : perform a 'flattening' functionality on the detection metadata
            when receiving each frame.
            flags: readable, writable, changeable only in NULL or READY state
            Boolean. Default: false
```

6.12. Hailo Gallery

6.12.1. Overview

HailoGallery is an element which enables the user to save and compare embeddings (HailoMatrix) that represents recogintion, in order to track objects across multiple streams. It is also enables saving and loading these embeddings into a local JSON file (database like), in order to track pre-saved objects.

Parameters

The hailogallery element provides a series of properties that allow you to adjust the gallery comparison algorithm. The most important property to set is `class-id`: this determines if the gallery will track all *HailoDetection* objects indiscriminately of class or focus only on detections of a specific class id (the default behavior is to track across-classes).

6.12.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoGallery

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
```

(continues on next page)

(continued from previous page)

```
            flags: readable, writable
            String. Default: "hailogallery0"
parent        : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
qos           : Handle Quality-of-Service events
            flags: readable, writable
            Boolean. Default: false
class-id      : The class id of the class to update into the gallery. Default -1
↪crosses classes.
            flags: readable, writable, changeable only in NULL or READY state
            Integer. Range: -2147483648 - 2147483647 Default: -1
similarity-thr    : Similarity threshold used in Gallery to find New ID's. Closer to 1.
↪0 is less similar.
            flags: readable, writable, controllable
            Float. Range:        0 -        1 Default:      0.15
gallery-queue-size  : Number of Matrixes to save for each global ID
            flags: readable, writable, controllable
            Integer. Range: 0 - 2147483647 Default: 100
load-local-gallery  : Load Gallery from JSON file
            flags: readable, writable, controllable
            Boolean. Default: false
save-local-gallery  : Save Gallery to JSON file
            flags: readable, writable, controllable
            Boolean. Default: false
gallery-file-path   : Gallery JSON file path to load
            flags: readable, writable, controllable
            String. Default: null
```

6.13. Hailo RoundRobin

6.13.1. Overview

`HailoRoundRobin` is an element that provides muxing functionality. It receives input from one or more sink pads and forwards them into a single src pad in round-robin method.

It also adds metadata to each buffer with the input pad name it was received on, The metadata's pupose is to be able to de-mux it easily later on by *hailostreamrouter* . De-muxing by streamiddemux is not supported with this element.

It can work in 3 modes:

- Funnel mode - push every buffer when it is ready no matter which pad it came from.
- Blocking mode - push every buffer when it is its pad's turn, and if the buffer is not ready, block until ready. This is the default mode.
- Non Blocking mode - push every buffer when it is its pad's turn, and if the buffer is not ready, skip it. This mode is useful when the video sources are not stable and may stop sending buffers for a while. In this case, the pipeline should not be blocked and should continue to process the other streams.

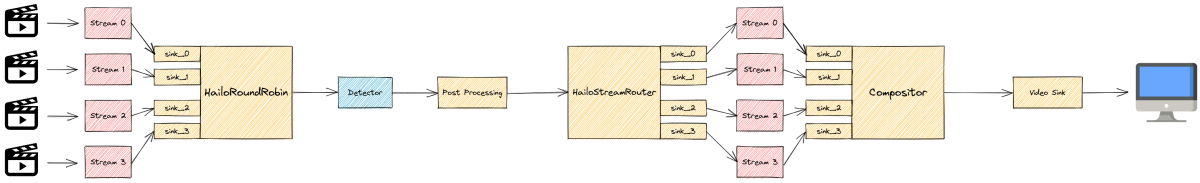
When using non-blocking mode, the element maintains a queue for sink pad that holds pointers to buffers. When a buffer is pushed to a sink pad, it is added to the queue. When the src pad wants to push a buffer, the element tries to get a buffer from the queue of the pad that is next in line. If the queue is empty, the element retries to get a buffer from the queue for a number of times (retries-num property). If the queue is still empty, the element skips the pad and tries to get a buffer from the next pad in line. The size of the queue and the number of retries can be configured by the properties:

- queue-size - Size of the queue for each pad.
- retries-num - Number of retries to get a buffer from a pad queue.

When using non-blocking mode, Compositor element is not supported, since it requires all the streams to be synchronized.

6.13.2. Example

Here's an example of a pesudo pipeline muxing 4 streams into one detection pipeline, and then de-muxing them into 2 separate pipelines - one for person attributes and one for face attributes.




```
for ((n = 0; n < 4; n++)); do
  filesrc location=video_$n ! decodebin ! roundrobin.sink_$n
hailoroundrobin name=roundrobin mode=1 !
<Rest of the pipeline>
```

6.13.3. Hierarchy

GObject

+—-GInitiallyUnowned

+—-GstObject

+—-GstElement +—-GstHailoRoundRobin

Pad Templates:

SINK template: 'sink_%u' Availability: On request Capabilities:

ANY

SRC template: 'src' Availability: Always Capabilities:

ANY

Element has no clocking capabilities. Element has no URI handling capabilities.

Pads:

SRC: 'src' Pad Template: 'src'

Element Properties: mode : Select the mode of the element (0 - funnel mode (push every buffer when it is ready), 1 - blocking mode (push every buf

fer when it is its pad's turn, and if the buffer is not ready, block until ready), 2 - non blocking mode(push every buffer when it is i

turn, and if the buffer is not ready, skip it)) flags: readable, writable Enum "GstHailoRoundRobinMode" Default: 1, "blocking-mode"

(0): funnel-mode - Funnel Mode (push every buffer when it is ready) (1): blocking-mode - Blocking Mode (push every buffer when it is its pad's turn, and if the buffer is not ready,

block until ready) (2): non-blocking-mode - Non Blocking Mode (push every buffer when it is its pad's turn, and if the buffer is not re

ady, skip it)

name [The name of the object] flags: readable, writable String. Default: “hailoroundrobin0”

parent [The parent of the object] flags: readable, writable Object of type “GstObject”

queue-size [Size of the queue for each pad (only relevant when using non-blocking mode)] flags: readable, writable, controllable Unsigned Integer. Range: 1 - 10 Default: 3

retries-num [Number of retries to get a buffer from a pad queue (only relevant when using non-blocking mode)] flags: readable, writable, controllable Unsigned Integer. Range: 1 - 20 Default: 3

6.14. Hailo Stream Router

6.14.1. Overview

`HailoStreamRouter` is an element that provides de-muxing functionality. It is designed to be used after muxing any number of streams using a *hailoroundrobin* element.

Note: *hailoroundrobin* tags each incoming frame with the input pad name it was received on (adding it to the metadata of the buffer).

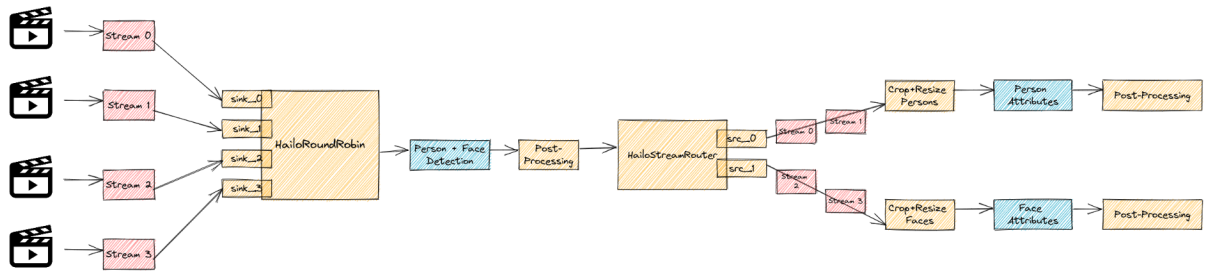
Each `HailoStreamRouter` has to be configured with a list of input names (In accordance with the `HailoRoundRobin` inputs). As an Example: .. code-block:

```
src_0::input-streams='<sink_0, sink_1>' src_1::input-streams='<sink_2, sink_3>'
```

`HailoStreamRouter` receives a frame on its sink pad, reads the input name from it’s metadata, and then passes the frame to pre configured source pads.

6.14.2. Example

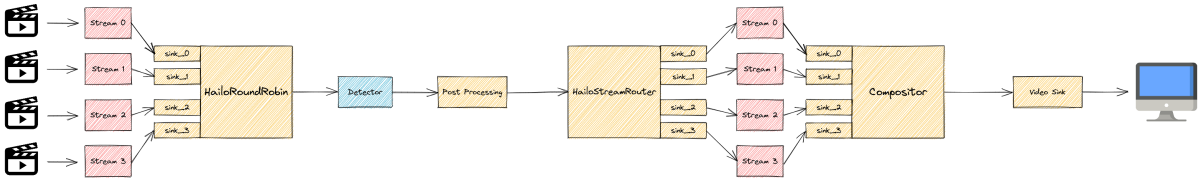
Here’s an example of a pesudo pipeline muxing 4 streams into one detection pipeline, and then de-muxing them into 2 separate pipelines - one for person attributes and one for face attributes.




```
for ((n = 0; n < 4; n++)); do
 filesrc location=video_$n ! decodebin ! roundrobin.sink_$n
hailoroundrobin name=roundrobin funnel-mode=false !
 ... Logic ...
hailostreamrouter name=router src_0::input-streams='<sink_0, sink_1>' src_1::input-
↪streams='<sink_2, sink_3>'
router.src_0 ! ... Logic ...
router.src_1 ! ... Logic ...
```

In this example HailoStreamRouter is configured with 2 source pads that each have a list of 2 input streams.

Another Example is using `HailoStreamRouter` as a classic de-muxer, where each input stream is mapped to a single output, and into compositor afterwards.



6.14.3. Hierarchy

```
 GObject
 --GInitiallyUnowned
   --GstObject
      --GstElement
         --GstHailoStreamRouter

Implemented Interfaces:
 GstChildProxy

Pad Templates:
 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

 SRC template: 'src_%u'
  Availability: On request
  Capabilities:
   ANY
  Type: GstHailoStreamRouterPad
  Pad Properties:
   input-streams    : Input streams of a srcpad
              flags: readable, writable, controllable
              GstValueArray of GValues of type "gchararray"

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'

Element Properties:
 name         : The name of the object
            flags: readable, writable, 0x2000
            String. Default: "hailostreamrouter0"
 parent       : The parent of the object
            flags: readable, writable, 0x2000
            Object of type "GstObject"
```


6.15. Hailo Export File

6.15.1. Overview

HailoExportFile is an element which provides an access point in the pipeline to export *HailoObjects meta* to a JSON file.
The meta itself is not changed or removed, and buffers continue onwards in the pipeline unchanged.

Parameters

The HailoExportFile element allows the user to change the output file name/path. The default is hailo_meta.json

6.15.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoExportFile

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)RGB, (string)YUY2 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)RGB, (string)YUY2 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
            flags: readable, writable
            String. Default: "hailoexportfile0"
 parent      : The parent of the object
            flags: readable, writable
            Object of type "GstObject"
 qos         : Handle Quality-of-Service events
            flags: readable, writable
```

(continues on next page)

(continued from previous page)

```
            Boolean. Default: false
 location       : Location of the JSON file to save
            flags: readable, writable, changeable only in NULL or READY state
            String. Default: "hailo_meta.json"
```

6.16. Hailo Export ZMQ

6.16.1. Overview

HailoExportZMQ is an element which provides an access point in the pipeline to export *HailoObjects meta* to a ZMQ socket.
The meta itself is not changed or removed, and buffers continue onwards in the pipeline unchanged.

Parameters

The HailoExportZMQ element allows the user to change the output port/protocol. The default is *tcp://*:5555*. Currently only PUB behvaior (PUB/SUB) is supported.

6.16.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoExportZMQ

Pad Templates:
 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)RGB, (string)YUY2 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)RGB, (string)YUY2 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'
```

(continues on next page)

(continued from previous page)

```
Element Properties:
 name        : The name of the object
               flags: readable, writable
               String. Default: "hailoexportzmq0"
 parent      : The parent of the object
               flags: readable, writable
               Object of type "GstObject"
 qos         : Handle Quality-of-Service events
               flags: readable, writable
               Boolean. Default: false
 address     : Address to bind the socket to.
               flags: readable, writable, changeable only in NULL or READY state
               String. Default: "tcp://*:5555"
```

6.17. Hailo Import ZMQ

6.17.1. Overview

HailoImportZMQ is an element which provides an access point in the pipeline to import *HailoObjects meta* from a ZMQ socket.
The meta is added to any pre-existing ROI in the buffer, and then the buffer continues onwards in the pipeline.

Parameters

The HailoImportZMQ element allows the user to change the input port/protocol. The default is *tcp://localhost:5555*. Currently only SUB behvaior (PUB/SUB) is supported.

6.17.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoImportZMQ

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   ANY

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   ANY

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
```

(continues on next page)

(continued from previous page)

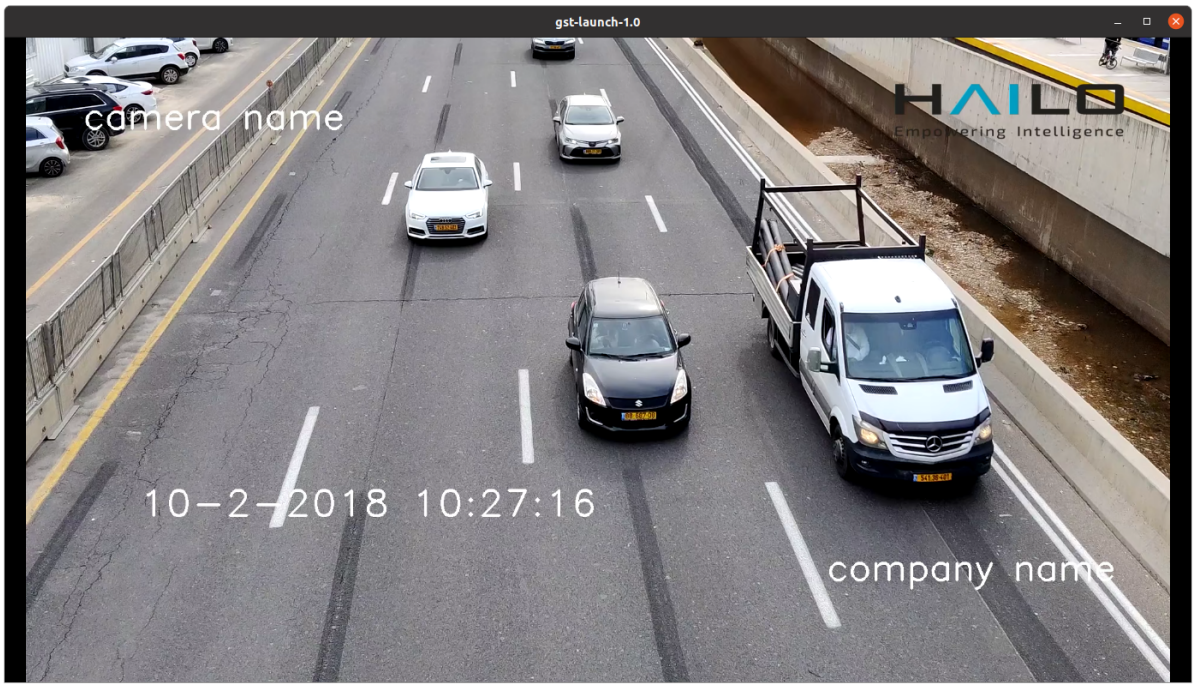
```
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
             flags: readable, writable
             String. Default: "hailoimportzmq0"
 parent      : The parent of the object
             flags: readable, writable
             Object of type "GstObject"
 qos         : Handle Quality-of-Service events
             flags: readable, writable
             Boolean. Default: false
 address     : Address to bind the socket to.
             flags: readable, writable, changeable only in NULL or READY state
             String. Default: "tcp://localhost:5555"
```

6.18. Hailo On-Screen Display

6.18.1. Overview

HailoOSD is an element which enables the user to draw static text, images, and timestamps on GstBuffers **using the DSP provided in Hailo-15**. By offloading the image blending to the DSP, high performance overlays can be achieved. The DSP also supports transparent blending, allowing HailoOSD to draw image files with transparency.

Currently only NV12 pipelines are supported by HailoOSD.



Parameters

The hailoosd element provides default behavior on what telemetry to draw.
The user can customize the overlay contents via json through the **config-file-path** property. This property accepts the path to a json that follows the following schema:
Any number of entries can be added to the **“image”**, **“text”**, and **“dateTime”** arrays in the json.

```
{
  ”image”: [
    {
      ”id”: ”image_id”,
      ”image_path”: ”/path/to/image”,
      ”width”: 0.2,
      ”height”: 0.13,
      ”x”: 0.76,
      ”y”: 0.05,
      ”z-index”: 1
    }
  ],
  ”dateTime”: [
    {
      ”id”: ”datetime_id”,
      ”font_size”: 2,
      ”text_color”: [0, 0, 255],
      ”x”: 0.1,
      ”y”: 0.7,
      ”z-index”: 3
    }
  ],
  ”text”: [
    {
      ”id”: ”text1_id”,
      ”label”: ”example text 1”,
      ”font_size”: 2,
      ”text_color”: [255, 0, 0],
      ”x”: 0.7,
      ”y”: 0.8,
      ”z-index”: 1
    },
    {
      ”id”: ”text2_id”,
      ”label”: ”example text 2”,
      ”font_size”: 2,
      ”x”: 0.05,
      ”y”: 0.1,
      ”z-index”: 1
    }
  ]
}
```


6.18.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailoOsd

Pad Templates:
 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)NV12 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)NV12 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 config-file-path    : json config file path
            flags: readable, writable, changeable only in NULL or READY state
            String. Default: "NULL"
 name          : The name of the object
            flags: readable, writable, 0x2000
            String. Default: "hailoosd0"
 parent        : The parent of the object
            flags: readable, writable, 0x2000
            Object of type "GstObject"
 qos           : Handle Quality-of-Service events
            flags: readable, writable
            Boolean. Default: false
```


6.19. Hailo Upload

6.19.1. Overview

`HailoUpload` is an element specifically designed for Hailo-15 system. It is responsible for transformation between memory spaces.

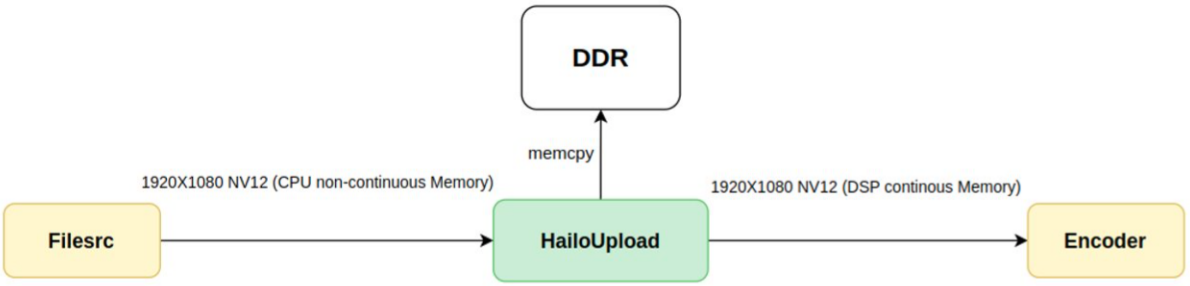
The element can be described by the following steps:

- Request an address (pointer) to a physically contiguous buffer from the kernel at a specifc size (determined by the format and resolution of the frame).
- Perform a memory copy to the target memory.
- Hold the address in a buffer pool - to free the buffer when its reference count is zero.
- Ensure that the output buffer is physically contiguous in memory and also virtually contiguous.

DSP and Encoder require physically contiguous buffers. It is recommended to make sure that a buffer is contiguous in memory, before being sent to an element that uses DSP- it is essential to avoid memcopies. (Like in *HailoCropper*). In Encoder - contiguous memory is mandatory. (see Media Library documentation for more information).

As an example, file source allocates non-contiguous buffers, meaning a hailoupload is required to use `hailoh265enc`:

```
gst-launch-1.0 filesrc location=video.raw name=src_0 ! rawvideoparse format=rgb
↪width=1920 height=1080 ! hailoupload ! hailoh265enc ! fakesink
```



Another example is to use hailoupload before hailonet, to make sure that the input buffer is virtually contiguous in memory:

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=mmap ! video/x-raw,format=NV12,
↪width=1920,height=1080 ! hailoupload ! hailonet hef-path=yolov5m_wo_spp_60p_nv12.
↪hef ! fakesink
```

`HailoUpload` inherits from `HailoDspBaseTransform` which is responsible for managing the allocation queries and buffer pool.

6.19.2. Hierarchy

```
  GObject
↪
+----GInitiallyUnowned
↪
  +----GstObject
↪
      +----GstElement
↪
        +----GstBaseTransform
↪
```

(continues on next page)

(continued from previous page)

```
            +----GstHailoDspBaseTransform
  ↪
              +----GstHailoUpload
  ↪

  ↪
Pad Templates:
  ↪
SINK template: 'sink'
  ↪
  Availability: Always
  ↪
  Capabilities:
  ↪
  ANY
  ↪

  ↪
SRC template: 'src'
  ↪
  Availability: Always
  ↪
  Capabilities:
  ↪
  ANY
  ↪

  ↪
Element has no clocking capabilities.
  ↪
Element has no URI handling capabilities.
  ↪

  ↪
Pads:
  ↪
SINK: 'sink'
  ↪
  Pad Template: 'sink'
  ↪
SRC: 'src'
  ↪
  Pad Template: 'src'
  ↪

  ↪
Element Properties:
  ↪
name         : The name of the object
  ↪
             flags: readable, writable, 0x2000
  ↪
             String. Default: ”hailoupload0”
  ↪
parent         : The parent of the object
  ↪
             flags: readable, writable, 0x2000
  ↪
             Object of type ”GstObject”
  ↪
pool-size      : Size of the pool of buffers to use for cropping. Default 10
  ↪
```

(continues on next page)

(continued from previous page)

```
              flags: readable, writable, changeable only in NULL or READY
 ↪state
              Unsigned Integer. Range: 1 - 2147483647 Default: 10
 ↪
qos           : Handle Quality-of-Service events
 ↪
              flags: readable, writable
 ↪
              Boolean. Default: false
```

6.20. Hailo Gray to NV12

6.20.1. Overview

Hailograytonv12 is an element that replaces the GRAY8 buffer with an NV12 buffer that is stored in the metadata of the GRAY8 buffer. It is a mirror to the hailonv12togray element.It is worth noting that this process does not involve any buffer copies, enabling high performance.This element is particularly useful for NV12 pipelines that want to use a GRAY8 formatted HEF.In such cases, the hailonv12togray element should be placed before the hailonet element, and the hailograytonv12 element should be placed after the hailonet element, to retrieve the original NV12 buffer.

6.20.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailograytonv12

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)NV12 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
     framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)GRAY8 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
     framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'
```

(continues on next page)

(continued from previous page)

```
Element Properties:
 name        : The name of the object
               flags: readable, writable
               String. Default: "hailograytonv12-0"
 parent      : The parent of the object
               flags: readable, writable
               Object of type "GstObject"
 qos         : Handle Quality-of-Service events
               flags: readable, writable
               Boolean. Default: false
```

6.21. Hailo NV12 to Gray

6.21.1. Overview

Hailonv12togray is an element that performs conversion of NV12 format to GRAY8 format, while preserving the initial NV12 buffer as the metadata of the resulting GRAY8 buffer. It is a mirror to the hailograytonv12 element.It is worth noting that this process does not involve any buffer copies, enabling high performance.This element is particularly useful for NV12 pipelines that want to use a GRAY8 formatted HEF.In such cases, the hailonv12togray element should be placed before the hailonet element, and the hailograytonv12 element should be placed after the hailonet element, to retrieve the original NV12 buffer.

6.21.2. Hierarchy

```
GObject
+----GInitiallyUnowned
   +----GstObject
      +----GstElement
         +----GstBaseTransform
            +----GstHailonv12togray

Pad Templates:
 SRC template: 'src'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)GRAY8 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

 SINK template: 'sink'
  Availability: Always
  Capabilities:
   video/x-raw
        format: { (string)NV12 }
         width: [ 1, 2147483647 ]
        height: [ 1, 2147483647 ]
      framerate: [ 0/1, 2147483647/1 ]

Element has no clocking capabilities.
Element has no URI handling capabilities.

Pads:
 SINK: 'sink'
```

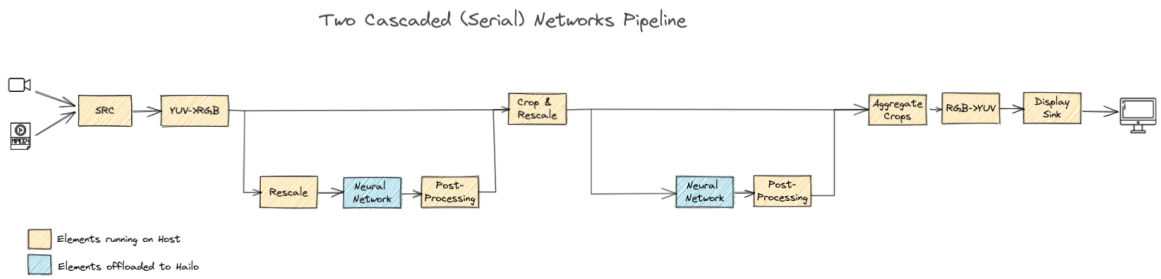
(continues on next page)

(continued from previous page)

```
  Pad Template: 'sink'
 SRC: 'src'
  Pad Template: 'src'

Element Properties:
 name        : The name of the object
             flags: readable, writable
             String. Default: "hailonv12togray0"
 parent      : The parent of the object
             flags: readable, writable
             Object of type "GstObject"
 qos         : Handle Quality-of-Service events
             flags: readable, writable
             Boolean. Default: false
```


7. Pipelines

7.1. Cascaded Networks Structure



This section provides a drill-down into the template of our cascaded network pipelines with a focus on explaining the `GStreamer` pipeline.

7.1.1. Example Pipeline

First, it is necessary to declare two sub-pipelines, these pipelines are derived from our *Single network template*

```
NETWORK_ONE_PIPELINE="videoscale qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailonet net-name=$NETWORK_ONE_NAME \
  hef-path=$hef_path is-active=true qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailofilter so-path=$NETWORK_ONE_SO function-name=$NETWORK_ONE_FUNC_NAME
↪qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0"

NETWORK_TWO_PIPELINE="queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-
↪time=0 ! \
  hailonet net-name=$NETWORK_TWO_NAME \
  hef-path=$hef_path is-active=true qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailofilter so-path=$NETWORK_TWO_SO function-name=$NETWORK_TWO_FUNC_NAME
↪qos=false ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0"
```

Next, insert them into the full pipeline:

```
gst-launch-1.0 \
  $source_element ! \
  tee name=t \
  \
  hailomuxer name=hmux \
  \
  t. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! hmux. \
  t. ! $NETWORK_ONE_PIPELINE ! hmux. \
  hmux. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailocropper internal-offset=$internal_offset name=cropper \
  \
  hailoaggregator name=agg \
  \
  cropper. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 !
↪agg. \
  \
  cropper. ! $NETWORK_TWO_PIPELINE ! agg. \
```

(continues on next page)

(continued from previous page)

```
  agg. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay ! \
  queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! videoconvert
↪! \
  fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=false text-
↪overlay=false ${additional_parameters}
```

To clarify the user's understanding pipeline is described section by section:

```
$source_element ! \
tee name=t \
\
hailomuxer name=hmux \
\
t. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! hmux. \
t. ! $NETWORK_ONE_PIPELINE ! hmux. \
hmux. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
```

This section of the pipeline is taken from our: *Single network template*. in brief: run an inference while keeping the original image resolution.

```
hailocropper internal-offset=$internal_offset name=cropper hailoaggregator name=agg
```

Hailocropper Hailocropper splits the pipeline into 2 branches: the first passes the original frame while the other passes crops from that original frame one at a time. The hailocropper chooses what crops to take from the original frame based on an so file that you can provide (typically a set of detections from a prior hailofilter postprocess). The hailocropper also scales all crops based on caps negotiation. This way if a hailonet is placed after the cropper (such as in this example), then the hailocropper will scale all crops to that hailonet's input network size. The hailoaggregator gets the original frame and then knows to wait for all related cropped buffers: aggregating all metadata to the original frame, then sending it forward.

```
cropper. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! agg.
↪\
```

The first part of the cascading network pipeline, passes the original frame on the bypass pads to hailoaggregator.

```
 cropper. ! $NETWORK_TWO_PIPELINE ! agg.

The second part of the cascading network pipeline, performs a second network on all
↪objects, which are cropped and scaled to the needed resolution by the HEF in the
↪hailonet.
```

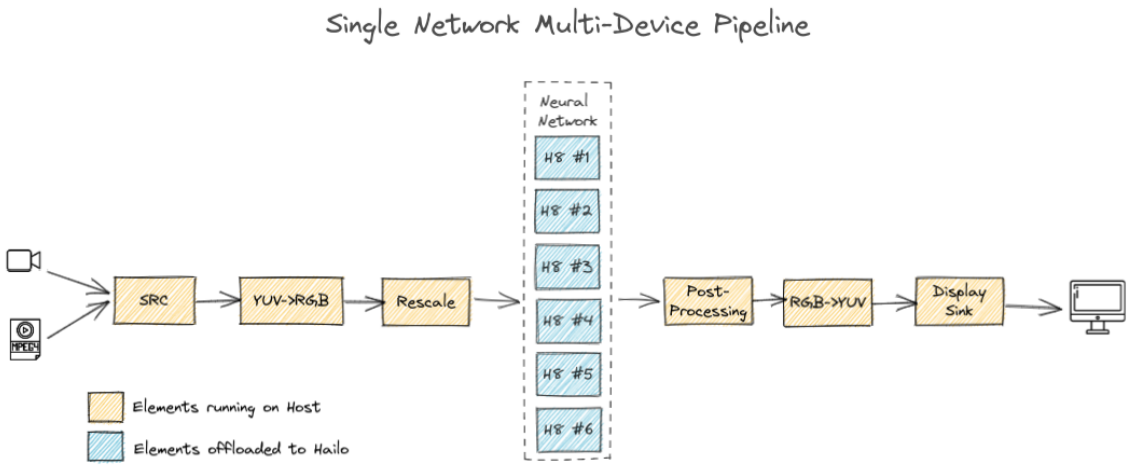
```
agg. ! queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! \
hailooverlay ! \
```

Aggregates all objects to the original frame, and draws them over the frame using the *hailooverlay* with specific drawing function.

```
queue leaky=no max-size-buffers=3 max-size-bytes=0 max-size-time=0 ! videoconvert ! \
fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=false text-
↪overlay=false
```

Display the final image using `fpsdisplaysink`.

7.2. Single Network Multi-Device Pipeline Structure



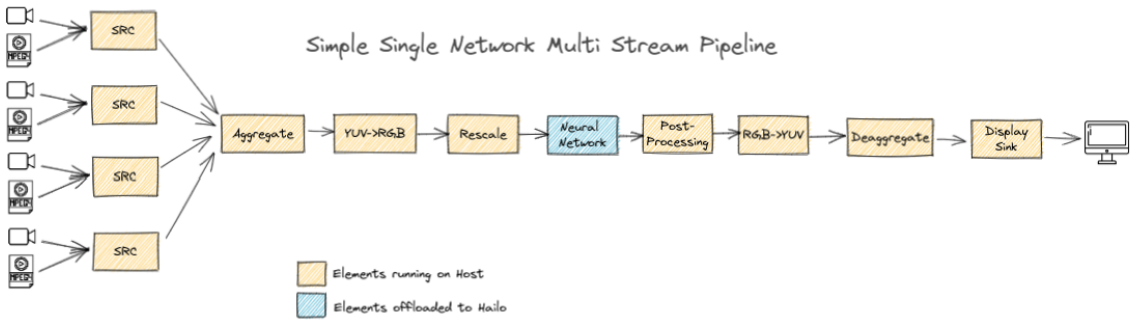
This page provides a drill-down into the template of our multi-device pipelines with a focus on explaining the `GStreamer` pipeline.

7.2.1. Example Pipeline

```
gst-launch-1.0 \
  $source_element ! videoconvert ! \
  videoscale ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailonet hef-path=$hef_path device-count=$device_count is-active=true ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter function-name=$network_name so-path=$postprocess_so qos=false ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay qos=false ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  videoconvert ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display sync=$sync_
↪pipeline text-overlay=false
```

This pipeline is based on-top of our *single network pipeline*. The number of physical devices to utilize is set via the `hailonet` device-count property (The default is 4 in this app)

7.3. Multi Stream Pipeline Structure



This page provides a drill-down into the template of our multi stream pipelines with a focus on explaining the `GStreamer` pipeline.

7.3.1. Example Pipeline

The first stage is to create the pipeline sources.

```
n=4
sources=''

for ((n = $start_index; n < $num_of_src; n++)); do
  sources+="$source_element ! \
      queue name=hailo_preprocess_q_$n leaky=no max-size-buffers=5 max-size-
↪bytes=0 max-size-time=0 ! \
      videoconvert ! videoscale ! fun.sink_$n \
      sid.src_$n ! queue name=comp_q_$n leaky=downstream max-size-buffers=30 \
      max-size-bytes=0 max-size-time=0 ! comp.sink_$n "
done
```

Each source is a sub-pipeline

```
pipeline="gst-launch-1.0 \
    funnel name=fun ! \
    queue name=hailo_pre_infer_q_0 leaky=no max-size-buffers=30 max-size-bytes=0
↪max-size-time=0 ! \
    hailonet hef-path=$HEF_PATH is-active=true ! \
    queue name=hailo_postprocess0 leaky=no max-size-buffers=30 max-size-bytes=0
↪max-size-time=0 ! \
    hailofilter so-path=$POSTPROCESS_SO qos=false ! \
    queue name=hailo_draw0 leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! \
    hailooverlay ! \
    streamiddemux name=sid \
    compositor name=comp start-time-selection=0 $compositor_locations ! \
    queue name=hailo_video_q_0 leaky=no max-size-buffers=30 max-size-bytes=0 max-
↪size-time=0 ! \
    videoconvert ! queue name=hailo_display_q_0 leaky=no max-size-buffers=30 max-
↪size-bytes=0 max-size-time=0 ! \
    $video_sink_element name=hailo_display sync=false \
    $sources
```

They can then be combined together

- `funnel` takes multiple input sinks and outputs one source. an N-to-1 funnel that attaches a streamid to each stream, can later be used to demux back into separate streams. this lets you queue frames from multiple

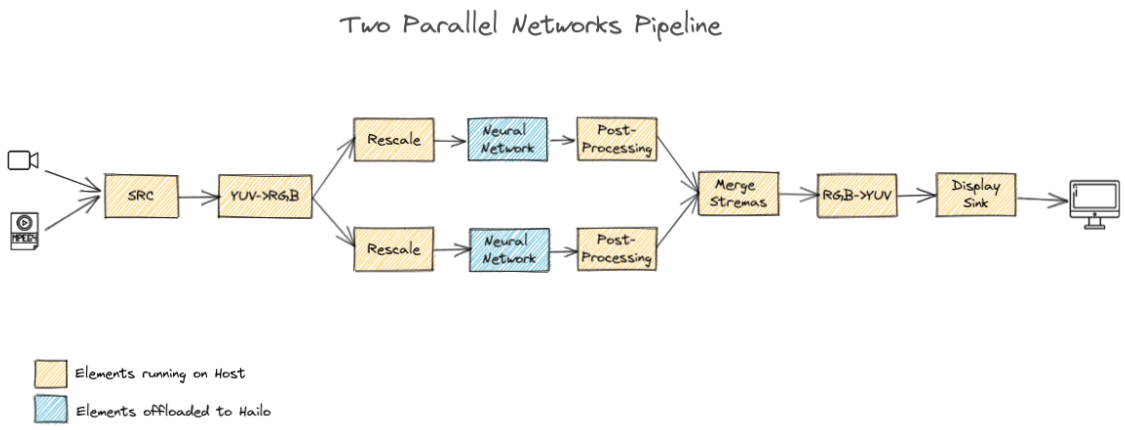
streams to send to the hailo device one at a time.

- `streamiddemux` a reverse to the funnel. It is a 1-to-N demuxer that splits a serialized stream based on stream id to multiple outputs.
- `compositor` composites pictures from multiple sources. handy for multi-stream/tiling like applications, as it lets you input many streams and draw them all together as a grid.

7.4. Parallel Networks Structure

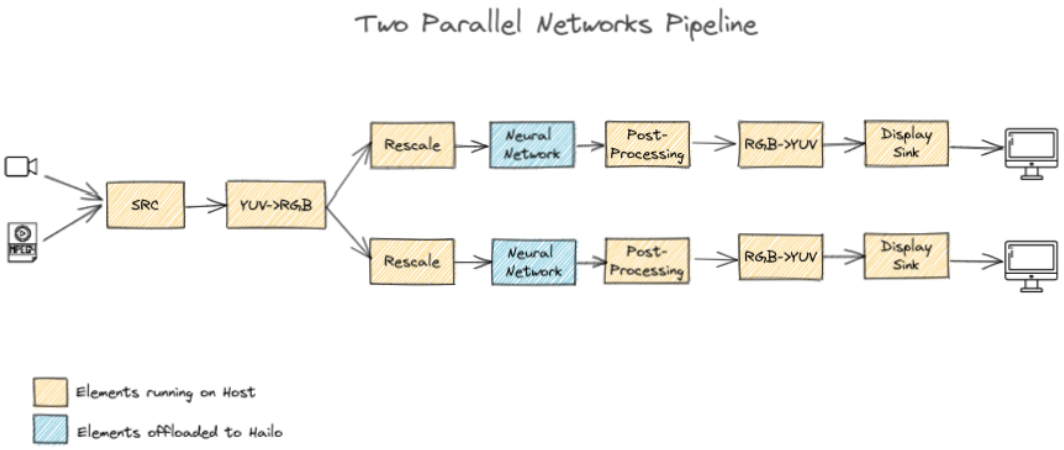
This page provides a drill-down into the template of our parallel networks pipeline with a focus on explaining the `GStreamer` pipeline.

7.4.1. Example Pipeline of One Display Multi Source




```
gst-launch-1.0 \
  $source_element ! videoconvert ! \
  queue name=hailo_pre_split leaky=no max-size-buffers=30 max-size-bytes=0 max-
↪size-time=0 ! \
  tee name=splitter \
  hailomuxer name=hailomuxer ! \
  queue name=hailo_draw0 leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! \
  hailooverlay qos=false ! \
  splitter. ! queue name=hailo_pre_infer_q_1 leaky=no max-size-buffers=5 max-size-
↪bytes=0 max-size-time=0 ! \
  hailonet hef-path=$NETWORK_ONE_HEF_PATH is-active=true ! \
  queue name=hailo_postprocess0 leaky=no max-size-buffers=30 max-size-bytes=0 max-
↪size-time=0 ! \
  hailofilter so-path=$NETWORK_ONE_POSTPROCESS_SO function-name=$NETWORK_ONE_
↪POSTPROCESS_FUNCTION_NAME qos=false ! hailomuxer. \
  splitter. ! queue name=hailo_pre_infer_q_0 leaky=no max-size-buffers=5 max-size-
↪bytes=0 max-size-time=0 ! \
  hailonet hef-path=$NETWORK_TWO_HEF_PATH is-active=true ! \
  queue name=hailo_postprocess1 leaky=no max-size-buffers=30 max-size-bytes=0 max-
↪size-time=0 ! \
  hailofilter so-path=$NETWORK_TWO_POSTPROCESS_SO function-name=$NETWORK_TWO_
↪POSTPROCESS_FUNCTION_NAME qos=false ! hailomuxer. \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display sync=false text-
↪overlay=false \
```

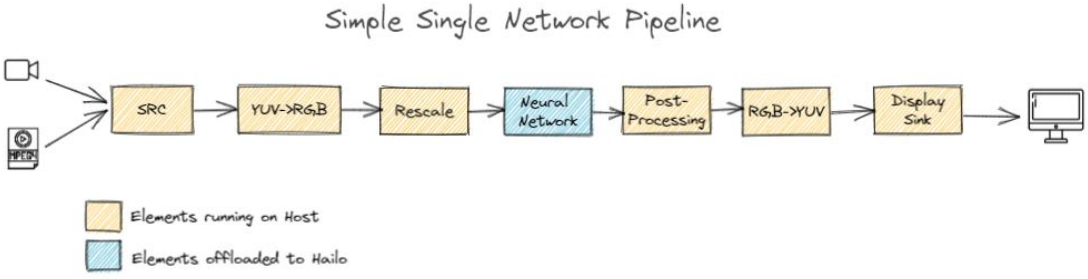

7.4.2. Example Pipeline of Two Displays




```
gst-launch-1.0 \
  $source_element ! videoconvert !
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  tee name=t ! queue ! videoscale ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailonet hef-path=$hef_path is-active=true net-name=$network_one_name ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter so-path=$network_one_so qos=false ! videoconvert ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display sync=false text-
↪overlay=false \
  t. ! \
  videoscale ! queue ! \
  hailonet hef-path=$hef_path is-active=true net-name=$network_two_name ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter so-path=$network_two_so function-name=mobilenet_ssd_merged
↪qos=false ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay ! videoconvert ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display2 sync=false
↪text-overlay=false ${additional_parameters}
```

This pipeline is based on-top of the *single network pipeline*, the modification amounts to using the GStreamer built in the tee element.

7.5. Single Network Pipeline Structure



This page provides a drill-down into the template of our single network pipelines with a focus on explaining the

GStreamer pipeline.

7.5.1. Example Pipeline

```
gst-launch-1.0 \
  filesrc location=$video_device ! decodebin ! videoconvert ! \
  videoscale ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailonet hef-path=$hef_path is-active=true ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter function-name=yolov5 so-path=$POSTPROCESS_SO qos=false ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay ! \
  videoconvert ! \
  fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=true text-
↪overlay=false
```

The pipeline functionality will be explained section by section:

```
filesrc location=$video_device ! decodebin ! videoconvert !
```

Specifies the location of the video used, then decodes and converts to the required format.

```
 videoscale ! \

Re-scale the video dimensions to fit the input of the network. The video scale finds out
↪the requires width and height using caps negotiation with ``hailonet``.
```

```
queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
```

Before sending the frames into the `hailonet` element, set a queue so no frames are lost (Read more about queues here)

```
hailonet hef-path=$hef_path is-active=true ! \
queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
```

Performs the inference on the Hailo-8 device.

```
hailofilter function-name=$POSTPROCESS_NAME so-path=$POSTPROCESS_SO qos=false ! \
queue name=hailo_draw0 leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! \
hailooverlay ! \
```

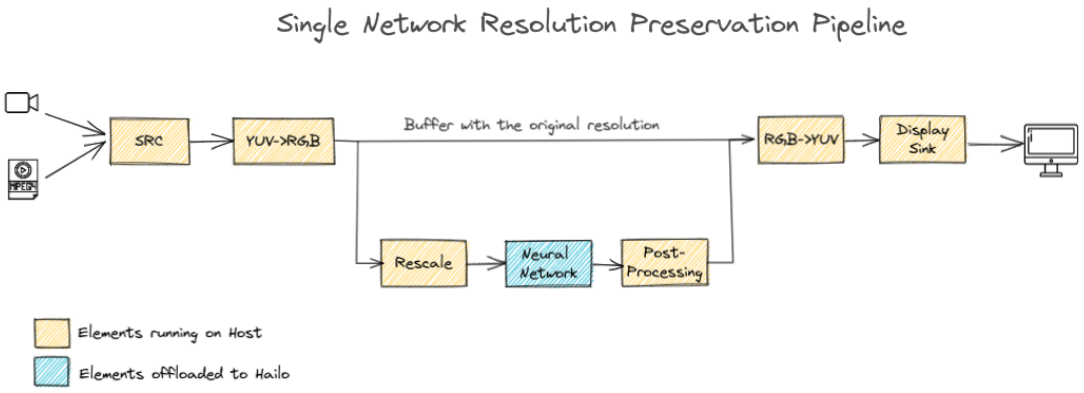
hailofilter performs a given post-process to extract the objects. The following *hailooverlay* element is able to draw standard `HailoObjects` to the buffer.

```
videoconvert ! \
fpsdisplaysink video-sink=xvimagesink name=hailo_display sync=true text-
↪overlay=false
```

Apply the final convert to let GStreamer utilize the format required by the `fpsdisplaysink` element

7.5.2. Example Pipeline with Resolution Preservation

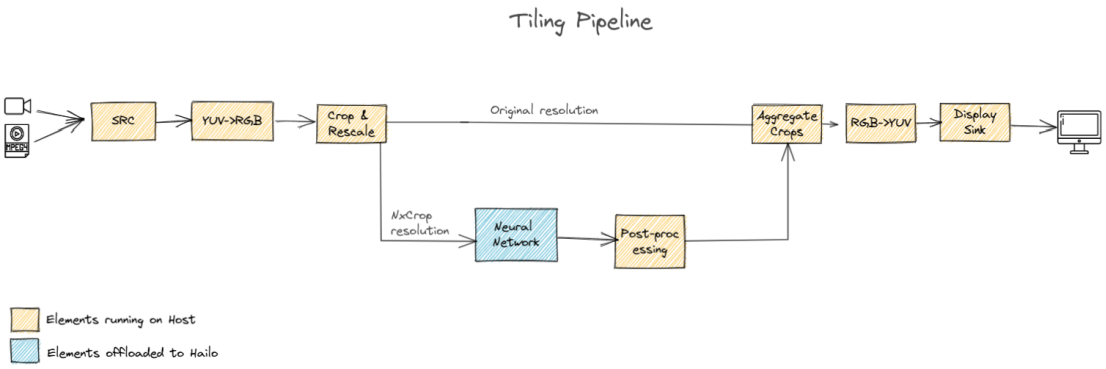
Using this template the source resolution would be preserved, this is an extension to our *Example pipeline*



An example for pipelines who preserve the original resolution:

```
gst-launch-1.0 \
  $source_element ! videoconvert ! \
  tee name=t hailomuxer name=mux \
  t. ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! mux. \
  t. ! videoscale ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailonet hef-path=$hef_path is-active=true ! \
  queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailofilter function-name=$network_name so-path=$postprocess_so qos=false ! mux. \
  mux. ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-time=0 ! \
  hailooverlay ! queue leaky=no max-size-buffers=30 max-size-bytes=0 max-size-
↪time=0 ! \
  videoconvert ! \
  fpsdisplaysink video-sink=$video_sink_element name=hailo_display sync=$sync_
↪pipeline text-overlay=false
```

7.5.3. Example Pipeline Single Network with Tiling



Tiling introduces two new elements:

- `hailotilecropper` **which splits the frame into tiles by separating the frame into rows and columns** (given as parameters to the element).
- `hailotileaggregator` which aggregates the cropped tiles and stitches them back to the original resolution.

An example for the pipeline itself could be found on our *Tiling app*.

8. Tools

8.1. Dot Visulaizer

8.1.1. How to use the tool?

```
# Run `gst_set_debug`
$ gst_set_debug

# Optional, adjust the TRACERS
# Important: Keep the `graphic` tracer
$ export GST_TRACERS="proctime;interlatency;framerate;queuelevel;graphic"

# Run the application
$ gst-launch-1.0 ...

# Prepare the tracers dir
$ gst_prepare_tracers_dir
# output should be in the following format:
Splited tracers dir: /local/workspace/tappas/tappas_traces_<date>

# Run the tool
$ hailo_tappas_dot_visualizer /local/workspace/tappas/tappas_traces_<date>
```

8.2. Cross-compile Hailo's GStreamer plugins

8.2.1. Overview

Hailo recommended method at the moment for cross-compilation is using Yocto SDK (aka Toolchain). We provide wrapper scripts whose only requirement is a Yocto toolchain to make this as easy as possible.

8.2.2. Preparations

In order to cross-compile you need to run TAPPAS container on a X86 machine and copy the Yocto toolchain to the container.

Toolchain

What is Toolchain?

A standard Toolchain consists of the following:

- Cross-Development Toolchain: This toolchain contains a compiler, debugger, and various miscellaneous tools.
- Libraries, Headers, and Symbols: The libraries, headers, and symbols are specific to the image (i.e. they match the image).
- Environment Setup Script: This *.sh file, once run, sets up the cross-development environment by defining variables and preparing for Toolchain use.

You can use the standard Toolchain to independently develop and test code that is destined to run on some target machine.

What should I add to my image?

For this example we will add the recipes to an NXP-IMX based image

Must add

```
# GStreamer plugins
IMAGE_INSTALL_append += "           \
  imx-gst1.0-plugin             \
  gstreamer1.0-plugins-bad-videoparsersbad \
  gstreamer1.0-plugins-good-video4linux2  \
  gstreamer1.0-plugins-base         \
"

# Opencv requirements for the hailo gstreamer plugin's postprocess
CORE_IMAGE_EXTRA_INSTALL += "   \
   libopencv-core-dev     \
   libopencv-highgui-dev    \
"
```

Nice to add

```
# GStreamer plugins
IMAGE_INSTALL_append += "           \
  gstreamer1.0-python           \
  gst-instruments             \
"

# Enable trace hooks for GStreamer
PACKAGECONFIG_append_pn-gstreamer1.0 += " gst-tracer-hooks"
```

Prepare the Toolchain

In order to generate a Yocto toolchain, use this following command

```
# Generate the Toolchain
bitbake -c do_populate_sdk <image name>
```

Copy the `toolchain` directory to the container using `docker cp`

```
docker cp toolchain hailo_tappas_container:/local/workspace/tappas
```

8.2.3. Components

TAPPAS

This script cross-compiles `TAPPAS` components. This script first unpack and installs the toolchain (if not already installed), and then cross-compiles TAPPAS core/.

Flags

```
$ ./cross_compile_tappas.py --help
usage: cross_compile_tappas.py [-h] [--remote-machine-ip REMOTE_MACHINE_IP] [--
↪build-lib {all,apps,plugins,libs,tracers}] [--clean-build-dir]
              [--install-to-rootfs]
              {aarch64,armv7l,armv7lhf,armv8a} {imx8,hailo15} {debug,release}
↪toolchain_dir_path

Cross-compile TAPPAS

positional arguments:
{aarch64,armv7l,armv7lhf,armv8a}
            Arch to compile to
{imx8,hailo15}  Target platform to compile to
{debug,release}     Build and compilation type
toolchain_dir_path  Toolchain directory path

optional arguments:
-h, --help       show this help message and exit
--remote-machine-ip REMOTE_MACHINE_IP
            remote machine ip
--build-lib {all,apps,plugins,libs,tracers}
            Build a specific tappas lib target (default all)
--clean-build-dir   Delete previous build cache (default false)
--install-to-rootfs  Install to rootfs (default false)
```

Example

Run the compilation script

Note: In this example we assume that the toolchain is located under toolchain-raw/hailo-dartmx8m-dunfell-aarch64-toolchain

```
$ ./cross_compile_tappas.py aarch64 imx8 debug toolchain
INFO:./cross_compile_gsthailotools.py:Building hailofilter plugin and post
↪processes
INFO:./cross_compile_gsthailotools.py:Running Meson build.
INFO:./cross_compile_gsthailotools.py:Running Ninja command.
```

Check the output directory

```
$ ls aarch64-gsthailotools-build/
build.ninja compile_commands.json config.h libs meson-info meson-logs meson-
↪private plugins
```

`libgsthailotools.so` is stored under libs

```
$ ls aarch64-gsthailotools-build/plugins/*.so
libgsthailotools.so
```

And the post-processes are stored under plugins

```
$ ls aarch64-gsthailotools-build/libs/*.so
libcenterpose_post.so libmobilenet_ssd_post.so
libclassification.so  libsegmentation_draw.so
libdebug.so       libyolo_post.so
libdetection_draw.so
```


8.2.4. Copy the cross-compiled files

Find out where the GStreamer plugins are stored in your embedded device by running the following command:

```
gst-inspect-1.0 filesrc | grep Filename | awk '{print $2}' | xargs dirname
```

Copy libgsthailo.so + libgsthailotools.so to the path found out above. Copy the post-processes so files under libs to the embedded device under /usr/lib/hailo-post-processes (create the directory if it does not exist)

Run gst-inspect-1.0 hailo and gst-inspect-1.0 hailotools and make sure that no error raises

9. Scripts

9.1. Using record_perf script to get records from tappas

1) Enter core/hailo/meson.build
2) Add to the Compiler Arguments:

```
add_global_arguments('-ggdb', language : 'cpp')
```

3) Compile tappas in debug mode
4) Run:

```
./scripts/misc/internals/record_perf/record_perf.sh <time>
```

5) run specific pipeline you want to debug
6) upload the output file to https://flamegraph.com/

