Table of Contents

CoCoMR Documentation Book

This document provides examples and documentation for CoCo practitioners following a by example approach.

The original document is here: http://coco.readthedocs.io/en/latest/

The publication presenting CoCo is the following:

Ruffaldi E. & Brizzi F. (2016). **CoCo - A framework for multicore visuo-haptics in mixed reality**. In SALENTO AVR, 3rd International Conference on Augmented Reality, Virtual Reality and Computer Graphics (pp. 339-357). Springer. doi:10.1007/978-3-319-40621-3_24 isbn:978-3-319-40620-6

Concepts

CoCo is based on the idea of **Composability** that is the possibility of interfacing components in different means depending on their needs, with the possibility of replacing given component with another that provides the same interface. This allows to replace a camera with another, or introduce debugging interface, or switch from live processing to offline processing.

CoCo Components are interfaced via Data-flow and Remote Procedures: the former allows the exchange of data between components with different rates and strategies, while the latter execute direct operations between components. The CoCo design is not excluding a Publisher-Subscriber approach, although at the moment it has not been implemented.

The other CoCo concept is the flexibility in the **Execution** that is in the way a given Component is mapped to operating system Threads. The association of a CoCo Component to its Execution is decided when the component is first running.

Execution patterns

CoCo is interested in several application scenarios in which there is a common aspect of soft real-time execution, high-computing requirements, possibly large data exchanged between components. In this scenario it is possible to identify several patterns of

execution.

First the simplest ones:

- Sink
- Filter
- Source

These simple cases can be aggregated in a general Computational Graph in which two patterns are common:

- Pipeline
- Farm

Starting from all these cases we can describe the type of execution partitioning that is provided by CoCo:

Launch XML

In most situation a CoCo Application is based on a XML specification of the Components and in their interconnection, and this XML file is launched by the CoCo Launcher (or the ROS CoCo Launcher). In the following will prove examples of the CoCo launch files leaving the details to the Reference Documentation and to the XML Schema.

Cameras and Vision

CoCo provides the interface to various types of Cameras and a series of internal vision based processing that are employed for the tracking and visualization of entities as in Augmented and Mixed Reality Scenarios.

Refer also to the Online Documentation:
http://coco.readthedocs.io/en/latest/computervision/

The Components in this Library are built around the exchange of video frames with the following features:

- Single Images (type ImageBuffer)
- Twin Images: e.g. stereo or RGB-D (type ?)
- Reference frame: used for the transformations
- Camera calibration info (type CameraInfo): used for the calibration

The Images exchanged can be represented via Pixel Formats that comprise the most common without the need to being exhaustive. Pixel Format variability is relevant for reducing the data conversion from sensors, from standard compression libraries and toward given processing types that are interested mainly in some channels of the image.

Pixel Formats

We support three families of Pixel Formats: RGB, YUV and Depth. In the YUV family we also comprise the Grayscale cases at 8 and 16bit. In the YUV we have only support for I420 in which Y, U and V planes are separates, each 1 byte per pixel but with U and V half the size.

The full listing is the following:

- RGB color: RGB, BGR, RGBA, BGRA
- YUV: I420P, GRAY8,GRAY16
- Depth: DEPTH16, DEPTH32F, INVDEPTH16

RGB Camera Sources

Two main RGB source:

- OpenCV based: image_source
- GStreamer based: gstreamer_source

The OpenCV source wraps cv::VideoCapture and in the same way allows to specify a string (imagepath) or a number (cameraid). Example of OpenCV-based source:

```
<component>
    <name>imagesrc</name>
    <task>image_source</task>
    <library>cocoar_opencv_camera</library>
    <attributes>
        <attribute name="image_path" value="$(arg image)" type="file"/>
        <attribute name="display" value="0"/>
    </attributes>
</component>
```

Example of GStreamer source with the pipeline specified. Note that "appsink name=sink" is so far required. The output can be RGB, YUV or GRAY8 and it has to be specified manually together with the size (**[Dev]** automatic extraction).

```
<component>
    <name>imagesrc</name>
    <task>GStreamerSource</task>
    <library>cocoar_gstreamer</library>
    <attributes>
        <attribute name="width" value="$(arg width)" />
        <attribute name="height" value="$(arg height)" />
        <attribute name="gray" value="$(arg gray)" />
        <attribute name="pipeline" value="avfvideosrc  ! videoscale ! videoconvert !
         video/x-raw,width=$(arg width),height=$(arg height),format=RGB !
          appsink name=sink" unless="$(arg gray)"/>
        <attribute name="pipeline" value="avfvideosrc  ! videoconvert !
        video/x-raw,width=$(arg width),height=$(arg height),format=GRAY8 !
        appsink name=sink" if="$(arg gray)"/>
    </attributes>
</component>
```

Additional we have the following three sources:

- Video 4 Linux Native
- Shared Memory
- [Dev] libuv for accessing specific cameras under OSX

[ROS] When using ROS we can read ROS Image sources as CoCo sources, specifying the RGB topic and the CameraInfo topic.

```
<component>
    <task>CameraInterfaceTask</task>
    <library>camera_interface</library>
    <attributes>
        <attribute name="rgb_sub_topic" value="/cameras/right_hand_camera/image" />
        <attribute name="camera_info_sub_topic" value="/cameras/right_hand_camera/camera_i
nfo" />
    </attributes>
</component>
```