



Mamba Realtime Python Reference - Linux

Author:
Nicolas BEUCHER

February 16, 2010

Contents

1	Introduction	3
2	Getting started : a small example	3
3	Realtime commands	4
4	Functions reference	4
4.1	Functions	4
4.1.1	activateRealTime()	4
4.1.2	deactivateRealTime()	4
4.1.3	initializeRealTime(device, devType, seqlenght=10)	4
4.1.4	isActivatedRealTime()	5
4.1.5	resetPaletteRealTime()	5
4.1.6	resetProcessRealTime()	5
4.1.7	setFrequencyRealTime(f)	5
4.1.8	setPaletteRealTime(palette)	5
4.1.9	setProcessRealTime(process, type, args=(), kwargs=)	5
4.1.10	takePictureRealTime(path)	5

List of Figures

1	The Realtime window looking at herself (full display)	4
---	---	---

1 Introduction

The Mamba Realtime module is an extension to the Mamba library for python that allows you to test your algorithms in realtime on your webcam (or any video capture device supported) images. It is based on the directshow API on Windows or Video4Linux2 (with a very limited support for V4L) on Linux for capture and on SDL library for display.

This document gives you the python function reference along with the list of dynamic commands available in the realtime window.

Be warn that although we tried to make the functions system independant there are still some difference between Windows and Linux implementation of this module (Mainly with the initializing function). Make sure the documentation you are currently reading refer to the system you planned to target, the title should provide this information.

2 Getting started : a small example

To start using the realtime, import the module :

```
#a from-import is the best way to use the realtime module
#import works but is really painful to use since the name of the
# module is this long
from mambaRealtime import *
```

The second step is to initialize you capture device. On linux it will be any Video4Linux2 API supported device (the support for V4L is very limited and may not work. On windows, any directshow API compatible device will be supported.

You should make sure that the libraries are installed on your system. For windows, the directshow API is part of directX and should be present by default on any modern updated windows (XP, Vista, ...).

To initialize your V4L2 device:

```
# This is assuming your device is "/dev/video0", there are a wide range of
# possibilities here but you must always give the path to a V4L2 supported
# device.
initializeRealTime("/dev/video0", V4L2)
```

To initialize your V4L device:

```
# Again, this is assuming your device is "/dev/video0".
#
# V4L support in mambaRealtime is WEAK and not tested extensively !
# V4L should die anyways so we are not going to bother.
initializeRealTime("/dev/video0", V4L)
```

Now you can actually launch the realtime module:

```
# Will create the realtime window and start the acquisition on the device
activateRealTime()
```

This should have opened a new window like **1** in which you can see what your device is actually acquiring such as your smiling face (come on !) in front of your webcam.

As you can see, there is no color. Just like mamba, the realtime module only works with greyscale image. This is not a limitation as mathematical morphology goes so you shouldn't be disturbed by it.

Once the realtime is active, you can specify a process to apply to the acquired images. Process are python function with a specific prototype. Refer to the documentation of the function setProcessRealTime() (see [4.1.9](#)). Here are some examples using the mambaComposed gradientImage function :

```
# first import the gradient function
from mambaComposed.gradient import gradientImage

# Setting up a simple gradient as the realtime process
setProcessRealTime(gradientImage, INSTANT)

# Setting up a thick gradient as the realtime process
setProcessRealTime(gradientImage, INSTANT, (2,))
```



Figure 1: The Realtime window looking at herself (full display)

3 Realtime commands

Once the realtime window is active, you have access to commands that will display information, activate palette coloring, activate the process or allow you to close the window.

- `<esc>` Closes the realtime acquisition. Once pressed you must reinitialize and reactivate it with the correct functions.
- `p` toggle the color palette. You should first specify one using the appropriate function.
- `p` toggle on/off the computation process. You should first specify one using the appropriate function.
- `r` display framerate information in a white bar at the bottom of the display. If the bar is full, the framerate is equal to the one required (10 by default), half the bar is filled then the framerate is half the one required and so on... You can see an example in figure 1
- `h` display histogram of what is actually displayed. You can see an example in figure 1

4 Functions reference

This module provides an interface to the realtime module of mamba. It provides functions to open the realtime thread and communicate with it.

4.1 Functions

4.1.1 `activateRealTime()`

Activates the realtime module. Opens the acquisition device and create the display in a separate thread. The thread ensure the acquisition and display of an image at a frequency of 10Hz with no treatment applied.

4.1.2 `deactivateRealTime()`

Deactivate the realtime module. Close the acquisition device and the display.

4.1.3 `initializeRealTime(device, devType, seqlength=10)`

Initialize the realtime module using 'device' for the acquisition. 'devType' indicates the type of the device (either V4L or V4L2). This function must be called before any other.

'seqlength' controls the length of the image sequence the thread is filling with the acquisition image. This sequence can be used in computation as an input.

4.1.4 `isActivatedRealTime()`

Returns True if the realtime thread is active and alive.

4.1.5 `resetPaletteRealTime()`

Reset the palette used to display the acquired images in the realtime thread.

4.1.6 `resetProcessRealTime()`

Reset the realtime thread so that no process (treatment) is applied to the images acquired and displayed.

4.1.7 `setFrequencyRealTime(f)`

Sets the frequency 'f' of acquisition and display in the realtime thread. This is strongly limited by the acquisition module and the selected process.

4.1.8 `setPaletteRealTime(palette)`

Set the palette used to display the acquired images in the realtime thread. 'palette' can be any palette as defined in the mamba module.

4.1.9 `setProcessRealTime(process, type, args=(), kwargs=)`

Change the 'process' (treatment) applied to the images acquired and display by the realtime thread.

if 'type' is set to INSTANT, 'process' must be a function/method whom prototype must be `process(imOut, imIn, ...)` where imIn and imOut are mamba.imageMb object. imIn is given by the acquisition device and imOut will be displayed.

if 'type' is set to SEQUENTIAL, 'process' must be a function/method whom prototype must be `process(imOut, seqIn, seqIndex, ...)` where imOut is a mamba.imageMb object. seqIn is a sequence as defined in module mambaComposed.sequence that contains the last 10 image obtained from the acquisition device. seqIndex indicates the index of the most recent. imOut will be displayed.

Others arguments can be passed through 'args' and 'kwargs'.

If process is not compliant with this, the treatment will be deactivated in realtime.

Take also care to remove any display activation (`showDisplay` method).

4.1.10 `takePictureRealTime(path)`

Takes a picture of the current image displayed (at the moment the function is called) and puts it into file 'path'.