# PYTHON module for TINE video

Davit Kalantaryan
Stefan Weisse

HELMHOLTZ | ASSOCIATION

DESY

## Content

## Introduction

PHYTON module has been implemented for decoding TINE video images. The name of main function of the module is "tonumpy". This function converts a TINE image (consisting of headers and byte array) to a numpy array. TINE image can be returned for example from TINE PYTHON module "get" call (which in turn got the image from a video server). The numpy array returned by this module can be plotted by matplotlib functions. In case the video server provides JPEG data instead of raw data, "tonumpy" function first decompress jpeg data to raw data, then converts it to numpy array. At the moment, all the image formats used in DESY (according to Stefan) are implemented and tested: JPEG-RGB24, JPEG-GRAY8, raw GRAY with 1 or 2 bytes per pixel, raw RGB24 color.

## Implementation

> Sources of library IJG (Independent JPEG Group) were used for creating BLOB (Binary Large OBject) decompressor API.

> To make the PYTHON extension smaller and faster, only IJG sources responsible for decompression are included in the module.

> Some modifications were done to decompress directly from PYTHON process memory instead of reading and decompressing a file.

> Some modifications were done to minimize memory allocation and deallocation (in the case if it is possible to use buffers for JPEG data and RAW data from previous calls). This makes decompress function faster and excludes memory fragmentation problem due to work with huge memories.

## Tests

- For initial tests the module is compiled and tested on WINDOWS and LINUX (ubuntu14.04).
- In order to build the Python module, Visual Studio 13 project for WINDOWS and qtproject for LINUX have been created. These projects one can find in the following svn repository ("https://svnsrv.desy.de/desy/ers/tools/blob_decompressor/").

## Example of steps for testing the module

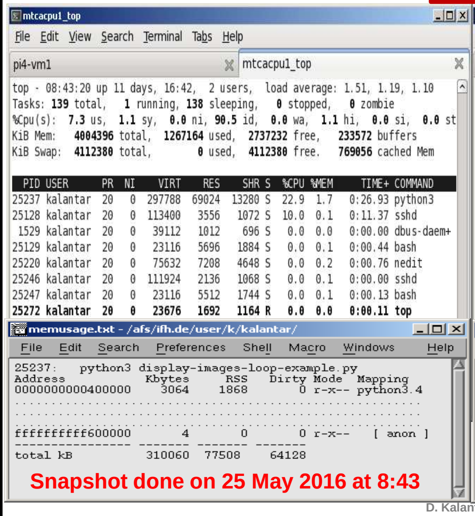
```
#jpeg rgb
import pytinevideo as ptv;
import PyTine as tine;
#m = tine.get('/TEST/Z41L.R2J/Output', 'Frame'); #  Zeuthen
m=tine.get('/DESY2/D2JPEG.Webcam/Output', 'Frame'); #HH
#import prettier_print_format_of_pytine_return as pp;
#print(pp.formatX(m));
rdt_np=ptv.tonumpy(m)
import matplotlib.pyplot as plt
implot=plt.imshow(rdt_np);
plt.show();
```

```
#raw gray
import pytinevideo as ptv;
import PyTine as tine;
#m = tine.get('/TEST/Z41.DShow/Output', 'Frame'); #  Zeuthen
m = tine.get('/DESY2/D2GRAB.Webcam/Output', 'Frame'); #  HH
#import prettier_print_format_of_pytine_return as pp;
#print(pp.formatX(m));
rdt_np=ptv.tonumpy(m)
import matplotlib.pyplot as plt
implot=plt.imshow(rdt_np);
plt.show();
```

## PYTHON script for testing module

The name of script is "**display-images-loop-example.py**". Script calls periodically complete cycle described in previous slide. The period is 100ms.
For testing following command could be called
**$python3 display-images-loop-example.py**

Iterations~924000
Δmemory=98208kB
=> Δmem_per_iter≈**106**B

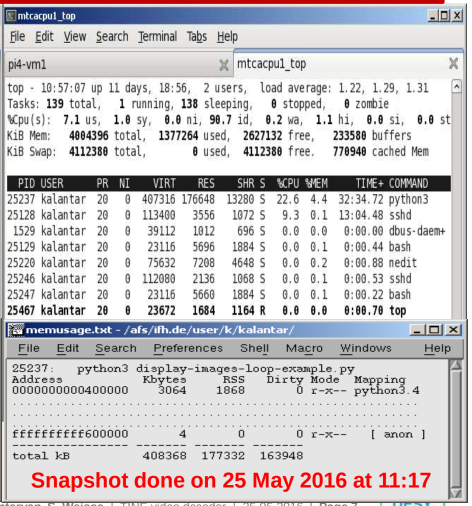


```
mtcacpu1_top
top - 08:43:20 up 11 days, 16:42,  2 users,  load average: 1.51, 1.19, 1.10
Tasks: 139 total,   1 running, 138 sleeping,   0 stopped,   0 zombie
%Cpu(s):  7.3 us,  1.1 sy,  0.0 ni, 90.5 id,  0.0 wa,  1.1 hi,  0.0 si,  0.0 st
KiB Mem:   4004396 total,  1267164 used,  2737232 free,   233572 buffers
KiB Swap:  4112380 total,        0 used,  4112380 free.   769056 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
25237 kalantar  20   0  297788  69024  13280 S  22.9  1.7   0:26.93 python3
25128 kalantar  20   0  113400   3556   1072 S  10.0  0.1   0:11.37 sshd
 1529 kalantar  20   0   39112   1012    696 S   0.0  0.0   0:00.00 dbus-daem+
25129 kalantar  20   0   23116   5696   1884 S   0.0  0.1   0:00.44 bash
25220 kalantar  20   0   75632   7208   4648 S   0.0  0.2   0:00.76 nedit
25246 kalantar  20   0  111924   2136   1068 S   0.0  0.1   0:00.00 sshd
25247 kalantar  20   0   23116   5512   1744 S   0.0  0.1   0:00.13 bash
25272 kalantar  20   0   23676   1692   1164 R   0.0  0.0   0:00.11 top
```

```
memusage.txt - /afs/ifh.de/user/k/kalantar/
25237:   python3 display-images-loop-example.py
Address           Kbytes     RSS   Dirty Mode  Mapping
0000000000400000    3064    1868       0 r-x-- python3.4
..........................................................
..........................................................
..........................................................
ffffffffff600000       4       0       0 r-x--   [ anon ]
---------------- ------- ------- -------
total kB          310060   77508   64128
```

Snapshot done on 25 May 2016 at 8:43



```
mtcacpu1_top
top - 10:57:07 up 11 days, 18:56,  2 users,  load average: 1.22, 1.29, 1.31
Tasks: 139 total,   1 running, 138 sleeping,   0 stopped,   0 zombie
%Cpu(s):  7.1 us,  1.0 sy,  0.0 ni, 90.7 id,  0.2 wa,  1.1 hi,  0.0 si,  0.0 st
KiB Mem:   4004396 total,  1377264 used,  2627132 free,   233580 buffers
KiB Swap:  4112380 total,        0 used,  4112380 free.   770940 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
25237 kalantar  20   0  407316 176648  13280 S  22.6  4.4  32:34.72 python3
25128 kalantar  20   0  113400   3556   1072 S   9.3  0.1  13:04.48 sshd
 1529 kalantar  20   0   39112   1012    696 S   0.0  0.0   0:00.00 dbus-daem+
25129 kalantar  20   0   23116   5696   1884 S   0.0  0.1   0:00.44 bash
25220 kalantar  20   0   75632   7208   4648 S   0.0  0.2   0:00.88 nedit
25246 kalantar  20   0  112080   2136   1068 S   0.0  0.1   0:00.53 sshd
25247 kalantar  20   0   23116   5660   1884 S   0.0  0.1   0:00.22 bash
25467 kalantar  20   0   23672   1684   1164 R   0.0  0.0   0:00.70 top
```

```
memusage.txt - /afs/ifh.de/user/k/kalantar/
25237:   python3 display-images-loop-example.py
Address           Kbytes     RSS   Dirty Mode  Mapping
0000000000400000    3064    1868       0 r-x-- python3.4
..........................................................
..........................................................
..........................................................
ffffffffff600000       4       0       0 r-x--   [ anon ]
---------------- ------- ------- -------
total kB          408368  177332  163948
```

Snapshot done on 25 May 2016 at 11:17

## Possible sources of the memory leaking

> TINEVIDEO module PYTHON part: creation and deletion of PYTHON variables are not done properly or something else in TINEVIDEO PYTHON code.

> BLOB decompressor API (it is already tested with stand alone application and it seems this part of code is not guilty).

> Decoder API

> TINE module PYTHON part: creation and deletion of PYTHON variables are not done properly or something else in TINE PYTHON code

> TINE library itself. But I guess this part is also not the source of leaking, while TINE is used in a lot of other applications and this phenomena was not seen

**For finding the source of memory leaking stand alone applications should be prepared to test all functionalities independently (for decompressor API it has been done)**

## Summary

- First release is ready
- Some tests have been done on WINDOWS and LINUX
- Tests shows that the module is stable. There is small memory growing (~100B pro one call of tine.get(…)+tinevideo.tonumpy(…)).
- As the memory growing is too small we think that the module already can be used.
- The stand alone application for testing BLOB decompressor API has been created and it showed that the memory growing does not come from here.
- It would be nice if somebody will take care of building and deploying the module in Hamburg (on MAC and Linux)

## Acknowledgment

Thank you for your attention!