pymepix Documentation

Release 1.1.dev0

CFEL Controlled Molecule Imaging group

Mar 24, 2021

CONTENTS

Pymepix documentation

CHAPTER

ONE

INTRODUCTION

Pymepix is intended to bridge the gap between Timepix3 and Python. The goal of the library is to allow a user without deep technical knowledge of Timepix3 to establish a connection, start acquisition, and retrieve and plot pixel and timing information in as few lines of code as possible; at the same time it provides all details to unleash the full power of Timepix3-SPIDR hardware. This is achieved by classes that act as a black-box, handling all of the low level TCP communication and decoding of the UDP data-stream, presenting them in a pythonic fashion. More advanced and lower-level control of SPIDR and Timepix3 is still available from these black-box classes or can be established directly by the user. For easy installation, it only depends on the standard python library, numpy and scikit-learn.

CHAPTER

TWO

GETTING STARTED

2.1 Installing

2.1.1 Installing from PyPI (platform-independent)

Simply to:

```
pip install pymepix
```

This should install all dependencies.

2.1.2 Installing from git source directly (platform-independent)

You can clone pymepix from our main git repository:

```
git clone https://github.com/CFEL-CMI/pymepix.git
```

Move into the pymepix library:

```
cd pymepix
```

Then, just do:

```
pip install .
```

To build documentation do:

```
python setup.py build_sphinx
```

2.2 Dependencies

The majority of pymepix only depends on numpy. To use centroiding, the sklearn package is required

CHAPTER

THREE

CONNECTING AND CONFIGURING

3.1 Connecting

For the camera to work you will have to set up the IP address on your machine, that the camera then communicates with. For Timepix3 with 10 Gb/s that is 192.168.100.1. Look up the official documentation for your camera to find out more.

Before using Pymepix, make sure your camera works properly with the SoPhy software.

The IP address of your TPX camera is the one seen on the OLED screen. Connecting to SPIDR can be done with:

```
>>> timepix = Pymepix(('192.168.100.10',50000))
```

The number of devices can be found using:

```
>>> len(timepix)
1
```

Meaning we have one device. To access this device directly, use:

```
tpx0 = timepix[0]
```

And to check the device name:

```
>>> tpx0.deviceName
W0026_K08
```

3.2 Configuring

To set the biasVoltage to 50 Volts in spidr you can do:

```
>>> timepix.biasVoltage = 50
```

Setting the we can manage its settings directly. To easily setup the device we can use a SoPhy config file (.spx):

```
tpx0.loadConfig('myFile.spx')
```

This sets up all the DAC setting and pixel configurations. Individual parameters can also be set for example. To set the fine threshold to 100 mV do:

```
>>> tpx0.Vthreshold_fine = 100
```

pixel threshold configurations can be set by passing a 256x256 numpy array:

```
import numpy as np
tpx0.pixelThreshold[...] = 0
```

The same for pixel masks, to set a checkboard mask do:

```
tpx0.pixelMask[::2] = 1
```

These need to be uploaded to timepix before they take effect:

```
>>> tpx0.uploadPixels()
```

The full list of parameters that can be set can be found in *pymepix.timepixdevice module*.

CHAPTER

FOUR

ACQUISITION

Acquisition can be started and stopped by:

```
import time
from pymepix import Pymepix

#Connect
timepix = Pymepix(('192.168.1.10',50000))

#Start acquisition
timepix.start()

#Wait
time.sleep(1.0)

#Stop acquisition
timepix.stop()
```

Pymepix provides data as a tuple given by (`MessageType`,data). These are explained in *Data Formats*. Retrieving the data can be done in to ways: Polling or Callback

4.1 Polling

Polling is where pymepix will place anything retrieved from Timepix into a ring polling buffer. This is the default mode but to reenable it you can use:

```
>>> timepix.enablePolling(maxlen=1000)
```

where *maxlen* describes the maximum number of elements in the buffer before older values are overwritten.

The user can retrieve this data by using:

```
>>> timepix.poll()
(MessageType.RawData,(array[9873240589723458980234 5,dtype=uint8],12348798))
```

If there is nothing in the polling buffer then a `PollBufferEmpty` exception is raised The poll buffer is limited in size but can be extended by doing:

```
>>> timepix.pollBufferLength = 5000
```

This will clear all objects using the polling buffer.

4.2 Callback

The callback method allows the user to deal with the data immediately when it is recieved. Setting this will clear the polling buffer of any contents.

To set a callback, first you need a function, for example:

```
def my_callback(data_type,data):
    print('My callback is running!!!!')
```

The format of the function must accept two parameters, `MessageType` and an extra data parameter. These are explained in *Data Formats*. Now to make pymepix use it simply do:

```
>>> timepix.dataCallback = my_callback
```

Now when acquisition is started:

```
>>> timepix.start()
```

The output seen is:

```
.. code-block:: sh
```

> My callback is running!!!! My callback is running!!!! My callback is running!!!! My callback is running!!!! My callback is running!!!!

4.3 Pipelines

Pymepix uses pipelines objects in order to process data. Each pipeline is set for each timepix device so each timepix can have a different data pipeline. You can configure them to postprocess or output data in certain ways. For example the `PixelPipeline` object will read from a UDP packet stream and decode the stream into *pixel x*, *pixel y*, *time of arrival* and *time over threshold* arrays. All data is progated forward through the pipeline so both UDP packets and decoded pixels are output.

To use the (default) `PixelPipeline` pipeline on the first connected timepix device you can do:

```
from pymepix.processing import PixelPipeline,CentroidPipeline

timepix[0].setupAcquisition(PixelPipeline)
```

If you need centroid you instead can do:

```
>>> timepix[0].setupAcquisition(CentroidPipeline)
```

Configuring the pipelines can be done using the acquisition property for the timepix device, for example to enable TOFs you can do:

```
>>> timepix[0].acquisition.enableEvents = True
```

A list of pipelines and setting can be found in *pymepix.processing.acquisition module*

CHAPTER

FIVE

DATA FORMATS

Contains a list of possible data formats output during acquisition. Each entry of the data section represents another element in the tuple. Example shows how to read the data through polling

5.1 UDP Packets

Data Type: `MessageType.RawData`

Data:

array(uint64) list of UDP packets

uint64 global timer from Timepix at time packets were recieved

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.RawData:
    packets,longtime = data
```

5.2 Decoded Pixels

Data Type: `MessageType.PixelData`

Data:

array(uint64) pixel x position

array(uint64) pixel y position

array(float) global time of arrival in seconds

array(uint64)) time over threshold in nanoseconds

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.PixelData:
    x,y,toa,tot = data
```

5.3 Decoded Triggers

Data Type: `MessageType.TriggerData`

Data:

array(uint64) trigger number

array(float) global trigger time in seconds

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.TriggerData:
    t_num,t_time = data
```

5.4 Time of Flight/Event

Data Type: `MessageType.EventData`

Data:

array(uint64) trigger number

array(uint64) pixel x position

array(uint64) pixel y position

array(float) time of flight relative to its trigger in seconds

array(uint64)) time over threshold in nanoseconds

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.EventData:
    trigger,x,y,tof,tot = data
```

5.5 Centroid Data

Data Type: `MessageType.CentroidData`

Data:

array(uint64) trigger number

array(uint64) center of mass x position

array(uint64) center of mass y position

array(uint64) total area

array(uint64) total time over threshold

array(uint64) Ignore (used in future)

array(uint64) Ignore (used in future)

array(uint64)) time of flight

Example:

```
data_type,data = timepix.poll()
if data_type is MessageType.CentroidData:
    trigger,x,y,area,integral,nu,nu,tof = data
```

CHAPTER

SIX

EXAMPLES

Starting timepix and polling data:

```
import pymepix
from pymepix.processing import MessageType
import numpy as np

#Connect to SPIDR
timepix = pymepix.Pymepix(('192.168.1.10',50000))

#Set bias voltage
timepix.biasVoltage = 50

#Set pixel masks
timepix[0].pixelThreshold = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].pixelMask = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].uploadPixels()

#Start acquisition
timepix.start()

while True:
    try:
        #Poll
        data_type,data = timepix.poll()
    except pymepix.PollBufferEmpty:
        #If empty then just loop
        continue

    #Handle Raw
    if data_type is MessageType.RawData:

        print('UDP PACKET')

        packets,longtime = data

        print('Packet ',packets)
        print('Time', longtime)

    #Handle Pixels
    elif data_type is MessageType.PixelData:

        print('I GOT PIXELS!!!!')

        x,y,toa,tot = data
```

(continues on next page)

(continued from previous page)

```
        print('x',x)
        print('y', y)
        print('toa', toa)
        print('tot',tot)

#Stop
timepix.stop()
```

Using callbacks to acquire:

```
import pymepix
from pymepix.processing import MessageType
import numpy as np
import time

#Connect to SPIDR
timepix = pymepix.Pymepix(('192.168.1.10',50000))

#Set bias voltage
timepix.biasVoltage = 50

#Set pixel masks
timepix[0].pixelThreshold = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].pixelMask = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].uploadPixels()

#Define callback
def my_callback(data_type,data):
    print('MY CALLBACK!!!!')
    #Handle Raw
    if data_type is MessageType.RawData:

        print('UDP PACKET')

        packets,longtime = data

        print('Packet ',packets)
        print('Time', longtime)

    #Handle Pixels
    elif data_type is MessageType.PixelData:

        print('I GOT PIXELS!!!!')

        x,y,toa,tot = data

        print('x',x)
        print('y', y)
        print('toa', toa)
        print('tot',tot)

#Set callback
timepix.dataCallback = my_callback

#Start acquisition
timepix.start()
```

(continues on next page)

(continued from previous page)

```
#Wait 5 seconds
time.sleep(5.0)
#Stop
timepix.stop()
```

CHAPTER

SEVEN

PYMEPIXACQ - COMMAND LINE

Included with pymepix is a command line code using the pymepix library to acquire from timepix. It is run using:

```
pymepix-acq --time 10 --output my_file
```

Doing:

```
pymepix-acq --help
```

Outputs the help:

```
usage: pymepix-acq [-h] [-i IP] [-p PORT] [-s SPX] [-v BIAS] -t TIME -o OUTPUT
                   [-d DECODE] [-T TOF]

Timepix acquisition script

optional arguments:
-h, --help                  show this help message and exit
-i IP, --ip IP              IP address of Timepix
-p PORT, --port PORT        TCP port to use for the connection
-s SPX, --spx SPX           Sophy config file to load
-v BIAS, --bias BIAS        Bias voltage in Volts
-t TIME, --time TIME        Acquisition time in seconds
-o OUTPUT, --output OUTPUT output filename prefix
-d DECODE, --decode DECODE Store decoded values instead
-T TOF, --tof TOF           Compute TOF if decode is enabled
```

TODO: MORE DOCS

CHAPTER

EIGHT

TROUBLESHOOTING

- **Whenever there are problems when working with the camera** First make sure you can ping the Timepix camera to ensure a working connection.

  Next try starting the SoPhy software and see if it can communicate properly with the camera. Remember to close SoPhy afterwards, as there can only be one process using the address.
- **Make sure to load the correct config file.** If the parameters are off, you might not be able to see anything.
- **Use a flashlight!** When the camera is properly connected and set up, you may use a flashlight to shine directly into the lens and next to it in quick succession.
- **You can see ToA but no ToF data** Check and maybe reconfigure the trigger.
- **Error: Address is already in use** If you get this error, look for any other process that is running and uses the corresponding IP and port. Also try restarting the camera
- genindex
- modindex
- search

CHAPTER

NINE

PYMEPIX DEVELOPER DOCUMENTATION

This developer documentation contains the API reference for *pymepix*.

9.1 API reference

9.1.1 General overview

The main Pymepix library is built up in several different submodules which each tackle a different task in working with the Timepix camera. As seen in the API index those are

- *config*
- *core*
- *processing*
- *SPIDR*
- *util*

The top layer Pymepix consists of *pymepix*, *timepixdef* and *timepixdevice*.

9.1.1.1 pymepix

pymepix provides the highest level of interaction with the library. A single *Pymepix* object will hold all connected Timepix devices and manage the users' interaction with those.

9.1.1.2 timepixdevice and timepixdef

A *timepixdevice* object holds all the communication with a single camera. It basically configures, starts and stops. The *timepixdef* has multiple enums to encode all kinds of parameters for Timepix.

9.1.1.3 config module

The *config* module gets the information for config parameters.
timepixconf is the base class for the possible configurations.
defaultconfig holds hardcoded config parameters for the camera to initialize.
sophyconfig imports information from SoPhy (.spx) config files. It reads and transforms that information to be used by Pymepix.

9.1.1.4 core module

The *core* module consists of only the log class. It defines functionality for Pymepix' needs and uses the basic python logging module.

9.1.1.5 processing module

The *processing* module provides the data pipeline to process the incoming camera data. Pymepix can use different acquisition pipelines to process the data. Those are defined in *acquisition* with the base functionality provided by *baseacquisition*. An acquisition pipeline determines which steps work in what order on the incoming data and connects those.

Each pipeline consists of acquisition stages (*baseacquisition*), where one stage holds the information about one logical step in the pipeline. Those tasks are currently *udpsampler* (capturing the packets), *rawtodisk* (saving the raw data), *packetprocessor* (interpreting the raw packets) and *centroiding* (compress data by finding blob centers). Each of these specific pipeline steps overwrites the *BasePipelineObject*, which is in fact a python *multiprocessing.Process*.

Each stage knows the task it has to fulfill and then creates one or multiple processes to work on that task in parallel.

datatypes provides an enum to classify the data that is passed through the pipeline at each step.

9.1.1.6 SPIDR module

This module communicates with the SPIDR chip of the Timepix. One *spidrcontroller* knows about one or more *spidrdevices*. *spidrcmds* lists the known commands to pass information and instructions to a chip. *spidrdefs* extends those commands by constants that can be passed. error contains information on possible errors from SPIDR.

9.1.1.7 util module

storage provides some functionality to save data.
spidrDummyTCP and -*UDP* can be used to simulate a timepix camera. Both are still rudimentary but helpful for debugging.
spidrDummyTCP accepts packets in so the configuration of timepix can be tested.
spidrDummyUDP samples and sends packets from a given file into the void. This can be used to test the pipeline functionality by capturing those packets with Pymepix.

9.1.2 pymepix

9.1.2.1 pymepix package

Subpackages

pymepix.SPIDR package

Submodules

pymepix.SPIDR.error module

pymepix.SPIDR.spidrcmds module

pymepix.SPIDR.spidrcontroller module

pymepix.SPIDR.spidrdefs module

pymepix.SPIDR.spidrdevice module

Module contents

pymepix.config package

Submodules

pymepix.config.sophyconfig module

pymepix.config.timepixconfig module

Module contents

pymepix.core package

Submodules

pymepix.core.log module

Module contents

pymepix.processing package

Submodules

pymepix.processing.acquisition module

9.2 Known issues and planned improvements

Todo: Synchronize the different (near)"stale" feature branches into develop and get into an actual (agile) git-flow development mode with progress being accumulated on *develop/*

Todo: TimePix doesn't get configured correctly and such SoPhy is still required for this, but compare branch XFEL-Nov2019 for current progress.

Todo: Centroiding is in many cases too slow for long data acquisition and not all data received gets saved to disk. Implement facilities to ensure data is saved to disk at highest priority and 'real-time' analysis, e.g., centroiding, is performed on as much data as possible – but nor more.

Todo: Following the previous ToDo: improve performance of centroiding.

Todo: Implement the FLASH/EuXFEL specific trainID reader on develop using programmable logic to turn it on (only) when available and wanted.

CHAPTER

TEN

REFERENCES

CHAPTER

ELEVEN

TODO LIST

Todo: Synchronize the different (near)"stale" feature branches into develop and get into an actual (agile) git-flow development mode with progress being accumulated on *develop/*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymepix/checkouts/latest/doc/api.rst, line 25.)

Todo: TimePix doesn't get configured correctly and such SoPhy is still required for this, but compare branch XFEL-Nov2019 for current progress.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymepix/checkouts/latest/doc/api.rst, line 28.)

Todo: Centroiding is in many cases too slow for long data acquisition and not all data received gets saved to disk. Implement facilities to ensure data is saved to disk at highest priority and 'real-time' analysis, e.g., centroiding, is performed on as much data as possible – but nor more.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymepix/checkouts/latest/doc/api.rst, line 31.)

Todo: Following the previous ToDo: improve performance of centroiding.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymepix/checkouts/latest/doc/api.rst, line 36.)

Todo: Implement the FLASH/EuXFEL specific trainID reader on develop using programmable logic to turn it on (only) when available and wanted.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymepix/checkouts/latest/doc/api.rst, line 38.)

CHAPTER

TWELVE

PYMEPIX DOCUMENTATION

Pymepix is a python library for interfacing, controlling and acquiring from SPIDR-Timepix detectors.

See also the accompanying Pymepix-viewer for an example user tool.

See [AlRefaie2019] for a description of version 1.0 of both tools and as a formal, citeable reference and the online manual for further information on later versions.

BIBLIOGRAPHY

[AlRefaie2019] A. Al-Refaie, M. Johny, J. Correa, D. Pennicard, P. Svihra, A. Nomerotski, S. Trippel, J. Küpper: PymePix: a python library for SPIDR readout of Timepix3, *Journal of Instrumentation* **14**, P10003–P10003 (2019), DOI: 10.1088/1748-0221/14/10/p10003; arXiv:1905.07999 [physics]