# DirectDemod Documentation

**Vinay C K (7andahalf)**

**Aug 19, 2019**

# CONTENTS:

DirectDemod is a set of python libraries that allow for easy handling, demodulation and decoding of raw IQ.wav (or IQ.dat) files directly captured from RTLSDRs. All the tools such as file readers, filters, chunking etc. are implemented and can be used as per the user's needs. Currently application specific demodulators are implemented for NOAA satellites (Image and sync detection), Funcube (similar cubesats) and Meteor M2 satellite (sync detection).

To get started on directly using this software for decoding: NOAA or demodulating: funcubes or meteor m2 satellites, look at the getting started guide. Some tutorials on how to use the modules and write your own scripts or to extend existing libraries, can be found at the tutorial folder in the repo.

CHAPTER

ONE

# DIRECTDEMOD: GETTING STARTED

## 1.1 Installation

DirectDemod is written in python3 and uses the following libraries:

Mandatory:

- scipy
- numpy
- matplotlib
- PIL
- colorsys

Optional (used for map overlay, georeferencing and image merger):

- pyorbital
- Basemap
- cartopy
- GeographicLib
- GDAL
- dateutil

Please make sure you have all the mandatory libraries installed.

Clone the repo into a folder and run "python main.py". If you get a usage statement, you are good to go. The usage statement has all the commands that can be given to the program.

## 1.2 Specific applications

Following are the application specific guides. Assuming you already know how to record RTLSDR data to a .wav or a .dat file.

## 1.3 To decode NOAA image

If you want to decode a NOAA IQ data into images you can run the command:

python main.py -c 137000000 -f 137100000 -d noaa "file.wav" here 13700000 is the centre frequency of the input file. 137100000 is the frequency of the satellite. "-d noaa" tells the program to use a noaa decoder on this. You should change these to match the file you have. When you run this, it will continuously print the status of decoding.

If you are skeptical if these settings are right and just want to test a portion of your file you can use the -s and -e options. For example if I want to just decode the file from 1000000 sample number to 2000000 sample number I can use the command,

> python main.py -c 137000000 -f 137100000 -s 1000000 -e 2000000 -d noaa "file.wav"

This is especially helpful to just do a small test run to make sure it has found the signal.

This will just generate a black and white image, and a color image if right channels are detected. You can have a look at other commands from the usage statement.

In case the signal is not found or is very noisy you can do the following trouble shooting:

- **sometimes I and Q channels migt be swapped, so use the -q flag to try to un-swap and try decoding.** e.g.
  python main.py -c 137000000 -f 137100000 -q -d noaa "file.wav"
- **If the signal is very noisy, you can play around with the bandwidth of the main filter by using the -b option**
  e.g. python main.py -c 137000000 -f 137100000 -b 1000000 -d noaa "file.wav"
- Try opening the file in a gui like SDRSHARP and make sure you can see and hear the characteristic NOAA waterfall. Note down the frequency and make sure you are providing accurate inputs to the program.

## 1.4 To get sync locations in IQ recordings

Currently the program has implementations of NOAA, Meteor M2 and Funcube (similar cubesats) so that accurate sync locations within the file could be found.

Similar to NOAA image extraction, if you provide the flag -sync, the program will generate a .csv file with the corresponding sync locations. For Funcube or Meteor satellites, the process is similar, but no need to pass -sync flag, the .csv file will be automatically generated.

CHAPTER

TWO

# DIRECTDEMOD: MODULES DOCUMENTATION

## 2.1 Signal object

**class** directdemod.comm.**commSignal**(*sampRate*, *sig=array([], dtype=float64)*, *chunker=None*)

This is an object used to store a signal and its properties

**__init__**(*sampRate*, *sig=array([], dtype=float64)*, *chunker=None*)

Initialize the object

**Parameters**

- **sampRate** (int) – sampling rate in Hz, will be forced to be an integer
- **sig** (numpy array, optional) – must be one dimentional, will be forced to be a numpy array
- **chunker** (chunker, optional) – Chunking object, if this signal is going to be processed in chunks

**bwLim**(*tsampRate*, *strict=False*, *uniq='abcd'*)

Limit the bandwidth by downsampling

**Parameters**

- **tsampRate** (int) – target sample rate
- **strict** (bool, optional) – if true, the target sample rate will be matched exactly
- **uniq** (str, optional) – in case chunked signal, uniq is to differentiate different bwLim funcs

**Returns** Updated signal (self)

**Return type** *commSignal*

**extend**(*sig*)

Adds another signal to this one at the tail end

**Parameters** **sig** (*commSignal*) – Signal to be added

**Returns** Updated signal (self)

**Return type** *commSignal*

**filter**(*filt*)

Apply a filter to the signal

**Parameters** **filt** (*filter*) – filter object

**Returns** Updated signal (self)

**Return type** *commSignal*

**funcApply** (*func*)
Applies a function to the signal

**Parameters** **func** (*function*) – function to be applied

**Returns** Updated signal (self)

**Return type** *commSignal*

**property length**
get length of signal

**Type** int

**offsetFreq** (*freqOffset*)
Offset signal by a frequency by multiplying a complex envelope

**Parameters** **freqOffset** (float) – offset frequency in Hz

**Returns** Signal offset by given frequency (self)

**Return type** *commSignal*

**property sampRate**
get sampling rate of signal

**Type** int

**property signal**
get signal

**Type** numpy array

**updateSignal** (*sig*)
Updates the signal

**Parameters** **sig** (numpy array) – New signal array

**Returns** Updated signal (self)

**Return type** *commSignal*

## 2.2 Specific applications

**class** directdemod.decode_noaa.**decode_noaa** (*sigsrc*, *offset*, *bw=None*)
Object to decode NOAA APT

**__init__** (*sigsrc*, *offset*, *bw=None*)
Initialize the object

**Parameters**

- **sigsrc** (commSignal) – IQ data source
- **offset** (float) – Frequency offset of source in Hz
- **bw** (int, optional) – Bandwidth

**property channelID**
get channel ID's

**Returns** [channelIDA, channelIDB]

**Return type** `list`

**getAccurateSync**(*useNormCorrelate=True*)
Get the sync locations: at highest sampling rate

**Parameters** **useNormCorrelate** (`bool`, optional) – Whether to use normalized correlation or not

**Returns** A list of locations of sync in sample number (start of sync)

**Return type** `list`

**property getAudio**
Get the audio from data

**Returns** An audio signal

**Return type** `commSignal`

**property getColor**
Get false color image (EXPERIMENTAL)

**Returns** A matrix list of pixel

**Return type** `numpy array`

**getCrudeSync**()
Get the sync locations: at constants.NOAA_CRUDESYNCSAMPRATE sampling rate

**Returns** A list of locations of sync in sample number (start of sync)

**Return type** `list`

**property getImage**
Get the image from data

**Returns** A matrix of pixel values

**Return type** `numpy array`

**property getImageA**
Get Image A from the extracted image

**Returns** A matrix list of pixel

**Return type** `numpy array`

**property getImageB**
Get Image B from the extracted image

**Returns** A matrix list of pixel

**Return type** `numpy array`

**getMapImage**(*cTime, destFileRot, destFileNoRot, satellite, tleFile=None*)
Get the map overlay of the image

**Parameters**

- **cTime** (`datetime`) – Time of start of capture in UTC
- **tleFile** (`str`, optional) – TLE file location, pulls latest from internet if not given
- **destFile** (`str`) – location where to store the image
- **satellite** (`str`) – Satellite name, ex: NOAA 19 etc.

**property useful**

10 consecutive syncs apart by 0.5s+-error

**Returns** 0 if not found, 1 if found

**Return type** `int`

**Type** See if some data was found or not

**class** `directdemod.decode_afsk1200.`**`decode_afsk1200`**(*sigsrc*, *offset*, *bw*)

Object to decode AFSK1200

**`__init__`**(*sigsrc*, *offset*, *bw*)

Initialize the object

**Parameters**

- **sigsrc** (`commSignal`) – IQ data source
- **offset** (`float`) – Frequency offset of source in Hz
- **bw** (`int`, optional) – Bandwidth

**decode_nrzi**()

Decode NRZI

**Parameters** **nrzi** (`list`) – the NRZI bits

**Returns** decoded NRZI bits

**Return type** `list`

**find_bit_stuffing**()

To find bit stuffing

**Parameters** **code_bit** (`list`) – the bits

**Returns** bit stuffing status

**Return type** `list`

**property getMsg**

Get the message from data

**Returns** string: A string of message data

**reduce_stuffed_bit**(*stuffed_bit*)

To remove stuffed bits

**Parameters**

- **code_bit** (`list`) – the bits
- **stuffed_bit** (`list`) – the result from find_bit_stuffing()

**Returns** bits free from stuffing

**Return type** `list`

**property useful**

See if atleast one message was found or not

**Returns** 0 if not found, 1 if found

**Return type** `int`

**class** `directdemod.decode_funcube.`**`decode_funcube`**(*sigsrc*, *offset*, *bw*, *center_frequency*, *signal_freq*, *corrfreq=False*)

Object to decode Funcube

`__init__`(*sigsrc*, *offset*, *bw*, *center_frequency*, *signal_freq*, *corrfreq=False*)
Initialize the object

**Parameters**

- **sigsrc** (`commSignal`) – IQ data source
- **offset** (`float`) – Frequency offset of source in Hz
- **bw** (`int`, optional) – Bandwidth

**property getSyncs**
Get syncs of Funcube

**Returns** list of detected syncs

**Return type** `list`

**property useful**
See if signal was found

**Returns** 0 if not found, 1 if found

**Return type** `int`

**class** `directdemod.decode_meteorm2.`**decode_meteorm2**(*sigsrc*, *offset*, *bw*)
Object to decode Meteor m2

`__init__`(*sigsrc*, *offset*, *bw*)
Initialize the object

**Parameters**

- **sigsrc** (`commSignal`) – IQ data source
- **offset** (`float`) – Frequency offset of source in Hz
- **bw** (`int`, optional) – Bandwidth

**property getSyncs**
Get syncs of Meteor M2

**Returns** list of detected syncs

**Return type** `list`

**property useful**
See if signal was found

**Returns** 0 if not found, 1 if found

**Return type** `int`

## 2.3 Filters

**class** `directdemod.filters.`**filter**(*b*, *a*, *storeState=True*, *zeroPhase=False*, *initOut=None*)
This is a parent object of all filters, it implements all the necessary properties. Refer to experiment 3 for details.

`__init__`(*b*, *a*, *storeState=True*, *zeroPhase=False*, *initOut=None*)
Initialize the object

**Parameters**

- **b** (`list`) – list of 'b' constants of filter

- **a** (`list`) – list of 'a' constants of filter
- **storeState** (`bool`, optional) – Whether the filter state must be stored. Useful when filtering a chunked signal to avoid border effects.
- **zeroPhase** (`bool`, optional) – Whether the filter has to provide zero phase error to the input i.e. no delay in the output (Note: Enabling this will disable 'storeState' and 'initOut')
- **initOut** (`list`, optional) – Initial condition of the filter

**applyOn** (*x*)

Apply the filter to a given array of signal

**Parameters** **x** (`numpy array`) – The signal array on which the filter needs to be applied

**Returns** Filtered signal array

**Return type** `numpy array`

**property getA**

Get 'a' of the filter

**Type** `list`

**property getB**

Get 'b' of the filter

**Type** `list`

**class** directdemod.filters.**rollingAverage** (*n=3*, *storeState=True*, *zeroPhase=False*, *initOut=None*)

A simple rolling average filter

**__init__** (*n=3*, *storeState=True*, *zeroPhase=False*, *initOut=None*)

Initialize the object

**Parameters**

- **n** (`int`, optional) – size of the rolling window
- **storeState** (`bool`, optional) – Whether the filter state must be stored. Useful when filtering a chunked signal to avoid border effects.
- **zeroPhase** (`bool`, optional) – Whether the filter has to provide zero phase error to the input i.e. no delay in the output (Note: Enabling this will disable 'storeState' and 'initOut')
- **initOut** (`list`, optional) – Initial condition of the filter

**class** directdemod.filters.**blackmanHarris** (*n*, *storeState=True*, *zeroPhase=False*, *initOut=None*)

Blackman Harris filter

**__init__** (*n*, *storeState=True*, *zeroPhase=False*, *initOut=None*)

Initialize the object

**Parameters**

- **n** (`int`) – size of the window
- **storeState** (`bool`, optional) – Whether the filter state must be stored. Useful when filtering a chunked signal to avoid border effects.
- **zeroPhase** (`bool`, optional) – Whether the filter has to provide zero phase error to the input i.e. no delay in the output (Note: Enabling this will disable 'storeState' and 'initOut')
- **initOut** (`list`, optional) – Initial condition of the filter

**class** directdemod.filters.**hamming**(*n*, *storeState=True*, *zeroPhase=False*, *initOut=None*)
Hamming filter

**__init__**(*n*, *storeState=True*, *zeroPhase=False*, *initOut=None*)
Initialize the object

**Parameters**

- **n** (int) – size of the window
- **storeState** (bool, optional) – Whether the filter state must be stored. Useful when filtering a chunked signal to avoid border effects.
- **zeroPhase** (bool, optional) – Whether the filter has to provide zero phase error to the input i.e. no delay in the output (Note: Enabling this will disable 'storeState' and 'initOut')
- **initOut** (list, optional) – Initial condition of the filter

**class** directdemod.filters.**gaussian**(*n*, *sigma*, *storeState=True*, *zeroPhase=False*, *initOut=None*)
Gaussian filter

**__init__**(*n*, *sigma*, *storeState=True*, *zeroPhase=False*, *initOut=None*)
Initialize the object

**Parameters**

- **n** (int) – size of the window
- **sigma** (float) – The standard deviation
- **storeState** (bool, optional) – Whether the filter state must be stored. Useful when filtering a chunked signal to avoid border effects.
- **zeroPhase** (bool, optional) – Whether the filter has to provide zero phase error to the input i.e. no delay in the output (Note: Enabling this will disable 'storeState' and 'initOut')
- **initOut** (list, optional) – Initial condition of the filter

**class** directdemod.filters.**butter**(*Fs*, *cutoffA*, *cutoffB=None*, *n=6*, *typeFlt=0*, *storeState=True*, *zeroPhase=False*, *initOut=None*)
Butterworth filter

**__init__**(*Fs*, *cutoffA*, *cutoffB=None*, *n=6*, *typeFlt=0*, *storeState=True*, *zeroPhase=False*, *initOut=None*)
Initialize the object

**Parameters**

- **Fs** (int) – Sampling frequency of signal
- **cutoffA** (float) – desired cutoff A of filter in Hz
- **cutoffB** (float, optional) – desired cutoff B of filter in Hz
- **n** (int, optional) – Order of filter
- **type** (constant, optional) – constants.FLT_LP to constants.FLT_BS, see constants module
- **storeState** (bool, optional) – Whether the filter state must be stored. Useful when filtering a chunked signal to avoid border effects.
- **zeroPhase** (bool, optional) – Whether the filter has to provide zero phase error to the input i.e. no delay in the output (Note: Enabling this will disable 'storeState' and 'initOut')
- **initOut** (list, optional) – Initial condition of the filter

**class** directdemod.filters.**remez**(*Fs*, *bands*, *gains*, *ntaps=128*, *storeState=True*, *zeroPhase=False*, *initOut=None*)

Remez band filter

**__init__**(*Fs*, *bands*, *gains*, *ntaps=128*, *storeState=True*, *zeroPhase=False*, *initOut=None*)

Initialize the object

**Parameters**

- **Fs** (`int`) – sampling frequency in Hz
- **bands** (`list`) – non-overlapping list of bands (in Hz) in increasing order. e.g [[0, 100], [400, 500], [600, 700]]
- **gains** (`float`) – Corresponding gains of the bands e.g. [0, 1, 0.5]
- **ntaps** (`int`, optional) – Number of taps of filter (number of terms in filter)
- **storeState** (`bool`, optional) – Whether the filter state must be stored. Useful when filtering a chunked signal to avoid border effects.
- **zeroPhase** (`bool`, optional) – Whether the filter has to provide zero phase error to the input i.e. no delay in the output (Note: Enabling this will disable 'storeState' and 'initOut')
- **initOut** (`list`, optional) – Initial condition of the filter

**class** directdemod.filters.**blackmanHarrisConv**(*n=151*)

Blackman Harris filter (by convolving, Not recommended for large signals)

**__init__**(*n=151*)

Initialize the object

**Parameters** **n** (`int`, optional) – size of the window

**applyOn**(*sig*)

Apply the filter to a given array of signal

**Parameters** **x** (`numpy array`) – The signal array on which the filter needs to be applied

**Returns** Filtered signal array

**Return type** `numpy array`

## 2.4 Demodulators

**class** directdemod.demod_fm.**demod_fm**(*storeState=True*)

Object for FM demodulation

**__init__**(*storeState=True*)

Initialize the object

**Parameters** **storeState** (`bool`) – Store state? Helps if signal is chunked

**demod**(*sig*)

FM demod a given complex IQ array

**Parameters** **sig** (`numpy array`) – numpy array with IQ in complex form

**Returns** FM demodulated array

**Return type** `numpy array`

**class** directdemod.demod_am.**demod_am**

AM demodulation by hilbert's transform

**demod** (*sig*)

AM demodulation by hilbert's transform

**Parameters** **sig** (`numpy array`) – Signal array to be demodulated

**Returns** Demodulated signal

**Return type** `numpy array`

**class** `directdemod.demod_fm.`**demod_fmAD** (*storeState=True*)

Object for FM demodulation (Alternative method using angle differentiation)

**__init__** (*storeState=True*)

Initialize the object

**Parameters** **storeState** (`bool`) – Store state? Helps if signal is chunked

**demod** (*sig*)

FM demod a given complex IQ array

**Parameters** **sig** (`numpy array`) – numpy array with IQ in complex form

**Returns** FM demodulated array

**Return type** `numpy array`

**class** `directdemod.demod_am.`**demod_amFLT** (*Fs*, *cutoff*)

AM demodulation by low pass filter

**__init__** (*Fs*, *cutoff*)

Initialize the object

**Parameters** **cutoff** (`int`) – lowpass cutoff frequency in Hz

**demod** (*sig*)

AM demodulation by low pass filter

**Parameters** **sig** (`numpy array`) – Signal array to be demodulated

**Returns** Demodulated signal

**Return type** `numpy array`

## 2.5 Sources

**class** `directdemod.source.`**IQwav** (*filename*, *givenSampFreq=None*)

An IQ.wav file source, typically an output recorded from SDRSHARP or other similar software

**__init__** (*filename*, *givenSampFreq=None*)

Initialize the object

**Parameters** **filename** (`str`) – filename of the IQ.wav file

**property length**

get source length

**Type** `int`

**limitData** (*initOffset=None*, *finalLimit=None*)

Limit source data

**Parameters**

- **initOffset** (`int`, optional) – starting index

- **finalLimit** (int, optional) – ending index

**read** (*fromIndex*, *toIndex=None*)
Read source data

**Parameters**

- **fromIndex** (int) – starting index
- **toIndex** (int, optional) – ending index. If not provided, the element at location given by fromIndex is returned

**Returns** Complex IQ numbers in an array

**Return type** numpy array

**property sampFreq**
get sampling freq of source

**Type** int

**property sourceType**
get source type

**Type** int

## 2.6 Sinks

**class** directdemod.sink.**wavFile** (*filename*, *sig*)
This object is used to write wav files

**__init__** (*filename*, *sig*)
Initialize the object

**Parameters**

- **filename** (str) – filename of the wav file
- **sig** (commSignal) – signal to be written

**property write**
writes the signal to file

**Type** sig (*wavFile*)

**class** directdemod.sink.**image** (*filename*, *mat*)
This object is used to display and write images

**__init__** (*filename*, *mat*)
Initialize the object

**Parameters**

- **filename** (str) – filename of the image file
- **mat** (list) – a matrix of pixel values

**property show**
shows the image

**Type** sig (*image*)

**property write**
writes the image to file

**Type** sig (`image`)

## 2.7 Chunking helper

**class** `directdemod.chunker.`**`chunker`**(*sigsrc*, *chunkSize=20000000*)
This object is just to help in chunking process

**`__init__`**(*sigsrc*, *chunkSize=20000000*)
Initialize the object

**Parameters**

- **sampRate** (`commSignal`) – commSignal object to be chunked
- **chunkSize** (`int`, optional) – chunk size

**`get`**(*name*, *init=None*)
get a variable value for to be used during chunking

**Parameters**

- **name** (`str`) – name of the variable
- **init** (`anything`) – initialize variable to this, if undefined previously

**Returns** value of variable

**Return type** `anything`

**property getChunks**
get the created chunks

**Type** `list`

**`set`**(*name*, *value*)
set a variable for to be used during chunking

**Parameters**

- **name** (`str`) – name of the variable
- **value** (`anything`, optional) – value of variable

## 2.8 Logging

**class** `directdemod.log.`**`log`**(*file=None*, *console=False*)
Object for logging

**`__init__`**(*file=None*, *console=False*)
Initialize the object

**Parameters**

- **file** (`str`, optional) – Filename, if log is to be stored into a file
- **console** (`bool`, optional) – Enables console logging

CHAPTER

THREE

# VISUALIZATIONS ROUTINE

The software presents several ways of visualizing NOAA images:

- as simple decoded image
- as georeferenced raster
- as an interactive web map with world map in the background
- plotted on virtual globe

The visualization process is as follows:

1. Decode the signal using one of the *directdemod* decoders.
2. Preprocess the image using *preprocess* function from *directdemod.georeferencer* package.
3. Georeference the image (see docs on georeferencer).
4. Generate map and globe visualizations using *generate_map.py* CLI interface, it will create tiles and then generate *map.html* and *globe.html* files. You can open the map directly in browser. To view the virtual globe you have to start a server *python -m http.server 8000* (python3), then go to *https://localhost:8000/globe.html*.

In the section below, are presented classes that are related to visualization of satellite imagery, along with some helper classes, which provide IO operations.

## 3.1 Image merger

Merger provides functionality, along with CLI interface, for merging several raster images. Merger supports several methods for overlapping parts of the images: *average*, *max*, *first*, *last*.

```
python merger.py -o o.tif -r average --files a.tif b.tif
```

Console options:

| | |
|---|---|
| **-f, --files** | list of files to merge |
| **-o, --output** | name of output file |
| **-r, --resample** | name of resample algorithm |

This module provides an API for merging multiple images. It extracts needed information and projects images onto mercator projection.

`directdemod.merger.`**`add_pixel_fn`**(*filename: str*, *resample_name: str*) → None

inserts pixel-function into vrt file named 'filename'

**Parameters**

- **`filename`** (`string`) – name of file, into which the function will be inserted

- **resample_name** (`string`) – name of resampling method

`directdemod.merger.`**build_vrt**(*vrt: str, files: List[str], resample_name: str*) → None

builds .vrt file which will hold information needed for overlay

**Parameters**

- **vrt** (`string`) – name of vrt file, which will be created
- **files** (`list`) – list of file names for merging
- **resample_name** (`string`) – name of resampling method

`directdemod.merger.`**get_resample**(*name: str*) → str

retrieves code for resampling method

**Parameters** **name** (`string`) – name of resampling method

**Returns** code of resample method

**Return type** method `string`

`directdemod.merger.`**main**() → None

CLI interface for satellite image merger

`directdemod.merger.`**merge**(*files: List[str], output_file: str, resample: str = 'average'*) → None

merges list of files using specific resample method for overlapping parts

**Parameters**

- **files** (`list[string]`) – list of files to merge
- **output_file** (`string`) – name of output file
- **resample** (`string`) – name of resampling method

## 3.2 Georeferencer

This class provides an API for image georeferencing. Sample command to run georeferencer.py, first generate tif raster with metadata, then georeference it using *georeferencer.py* interface. The first command will extract the capture date from the name of wav file, and then will compute the coordinates of the satellite based on this date. Computed data will be stored in new file in '.tif' format. This file could be then used for georeferencing.

```
python misc.py -f ../samples/SDRSharp_20190521_170204Z_137500000Hz_IQ.wav -i
../samples/decoded/SDRSharp_20190521_170204Z_137500000Hz.png
```

```
python georeferencer.py -m -i ../samples/decoded/SDRSharp_20190521_170204Z_137500000Hz.
tif
```

Console options:

| | |
|---|---|
| **-m, --map** | flag to create map overlay |
| **-i, --image** | path to image file |

**class** `directdemod.georeferencer.`**Georeferencer**(*tle_file: str = ''*)

This class provides an API for image georeferencing. It extracts the information from descriptor file, translates and warps the image to defined projection.

**__init__**(*tle_file: str = ''*)

Georeferencer constructor

**Parameters** **tle_file** (`string`, optional) – file with orbit parameters

**static compute_angle**(*long1: float*, *lat1: float*, *long2: float*, *lat2: float*) → float

compute angle between 2 points, defined by latitude and longitude

**Parameters**

- **long1** (`float`) – longitude of start point
- **lat1** (`float`) – latitude of start point
- **long2** (`float`) – longitude of end point
- **lat2** (`float`) – latitude of end point

**Returns** angle between points

**Return type** `float`

**static compute_gcp**(*long: float*, *lat: float*, *angle: float*, *distance: float*, *width: float*, *height: float*) → osgeo.gdal.GCP

compute coordinate of GCP, using longitude and latitude of starting point, azimuth angle and distance to the point

**Parameters**

- **long** (`float`) – longitude of start point
- **lat** (`float`) – latitude of start point
- **angle** (`float`) – azimuth between start point and GCP
- **(** (*distance*) – obj: *float*): distance to point in meters
- **width** (`float`) – w-axis coordinate
- **height** (`float`) – height-axis coordinate

**Returns** instance of GCP object

**Return type** `gdal.GCP`

**compute_gcps**(*descriptor: dict*, *image: numpy.ndarray*) → List[osgeo.gdal.GCP]

compute set of Ground Control Points

**Parameters**

- **descriptor** (`dict`) – descriptor dictionary, which describes the image
- **image** (`np.ndarray`) – image as np.ndarray

**Returns** list of GCPs

**Return type** `list`

**static create_desc**(*descriptor: dict*, *output_file: str*) → None

create descriptor for *output_file* file

**Parameters**

- **descriptor** (`dict`) – descriptor dictionary
- **output_file** (`string`) – name of the output file

**georef**(*descriptor: dict*, *output_file: str*, *resample_alg=0*) → None

georeferences the satellite image from descriptor file using GDAL Python API

**Parameters**

- **descriptor** (`dict`) – descriptor dictionary
- **output_file** (`string`) – name of the output file

- **resample_alg** (gdalconst, optional) – algorithm for resampling

**georef_os** (*descriptor: dict*, *output_file: str*) → None
georeferences the satellite image from descriptor file, using GDAL compiled binaries. Can be used when gdal binaries are available only

**Parameters**

- **descriptor** (dict) – descriptor dictionary
- **output_file** (string) – name of the output file

**georef_tif** (*image_name: str*, *output_file: str*, *resample_alg=0*) → None
georeferences the satellite image from tif file using GDAL Python API. Descriptor is extracted directly from tif file

**Parameters**

- **image_name** (string) – path to tiff file, which contains needed metadata
- **output_file** (string) – path to output file
- **resample_alg** (gdalconst) – resampling algorithm (nearest, bilinear, cubic)

**static to_string_gcps** (*gcps: List[osgeo.gdal.GCP]*) → str
create string representation of gcp points

**Parameters** **gcps** (list) – list of gcp points

**Returns** gcp points represented as a string

**Return type** string

## 3.3 Map generation

To generate visualization of raster use *generate_map.py* interface. The following command will generate a TMS (Tile Map Service) and 2 visualization files in *samples/tms* directory.

```
python generate_map.py --raster ../samples/decoded/raster.tif --tms ../
samples/tms
```

You can run *map.html* by opening in the browser.

To use *globe.html* go to tms directory and type the following command to start http server on port 8000 (for python3):

```
python -m http.server 8000
```

Then open browser and go to *http://localhost:8000/globe.html*.

## 3.4 Json encoder

Json encoder, which handles encoding numpy array and datetime objects.

**class** directdemod.misc.**Encoder** (**, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)
JSON encoder, which handles *np.ndarray* and *datetime* objects

**`__init__`**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an OverflowError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

**`default`**(*obj*) → Any

Encode the object

**Parameters** **`obj`** (`object`) – oject to encode

**Returns** encoded object

**Return type** `object`

CHAPTER

FOUR

# TUTORIALS

This section presents several usage examples of directdemod package. Each usage example is accompanied with thorough explanation and an appropriate data, which is stored in *tutorial/data/* folder.

*Note:* python version should be higher then 3.6, preferably 3.7. In tutorials below *python* command refers to python3.

*Note on warnings*: depending on your python version you can see the following warnings, which is ok, they are not the errors of the program.

- YAMLLoadWarning: calling yaml.load() without Loader=… is deprecated
- Warning 1: TIFFReadDirectoryCheckOrder:Invalid TIFF directory; tags are not sorted in ascending order

## 4.1 Data extraction (misc.py)

*misc.py* script is used to perform data extraction of satellite parameters. When running, *misc.py* will extract data from SDR file, create copy of provided image with .tif extension and embed extracted data as json into it. CLI interface receives following console options:

| | |
|---|---|
| **-f, --file_sdr** | path to recorded SDR file |
| **-i, --image_name** | path to decoded and preprocessed image |
| **-t, --tle** | path to tle file |
| **-s, --sat_type** | satellite type |

Tle and satellite type parameters are optional. The *tutorial/data/metadata* directory contains sample files - sdr file and the decoded image. Sample command:

```
python directdemod/misc.py -f tutorial/data/metadata/SDRSharp_20190521_170204Z_
↪137500000Hz_IQ.wav \
-i tutorial/data/metadata/image.png
```

Created *image.tif* file will contain the satellite data (orbit parameters, satellite type etc. ) in json format along with the image itself; it will be ready for performing georeferencing.

## 4.2 Georeferencer

Georeferencer class is intended to provide methods for georeferencing NOAA images. It provides CLI interface for running the program from command line. CLI interface takes following options (map, resample and output_file are optional):

| | |
|---|---|
| **-i, --image_name** | path to image file |

**-o, --output_file** name of output file

**-m, --map** flag to create map overlay

**-r, --resample** resample algorithm

Georeferencer assumes that the image passed via *–image_name* option contains a descriptor file embedded within it. If the file doesn't contain it, the processing will result in an error.

As an example usage let's say we have a decoded and preprocessed NOAA image *start.png* and the file it was extracted from *SDRSharp_20190521_170204Z_137500000Hz_IQ.wav*. To receive a georeferenced image we need to do the following:

1. Extract the information from *.wav* file name and save it to *start.tif* file.
2. Georeference tif file.

To extract data *misc.py* command is used (see misc.py docs).

```
python directdemod/misc.py -f tutorial/data/georef/SDRSharp_20190521_170204Z_
↪137500000Hz_IQ.wav \
-i tutorial/data/georef/start.png
```

To georeference the file we use *georeferencer.py* file. *start.tif* will contain georeferenced image.

```
python directdemod/georeferencer.py -i tutorial/data/georef/start.tif
```

## 4.3 Map Overlay

Map overlay can be created using *–map* option of the georeferencer. After the georeferencing is done map borders shapefile will be overlayed on top of it.

To create an overlay over image use the following command.

```
python directdemod/georeferencer.py -m -i tutorial/data/overlay/no_overlay.tif \
-o tutorial/data/overlay/with_overlay.tif
```

## 4.4 Merger

Merge is used to combine several georeferenced images into one single raster, taking care of overlapping regions. Merger CLI interface has following console options:

**-f, --files** list of input files

**-o, --output** name of output file

**-r, --resample** resample algorithm

Resample option receives one of the four merging method names:

1. first
2. last
3. average
4. max

The *tutorial/data/merge* directory contains several example usage files. *image1.tif* and *image2.tif* are sample files for merging. Use following command to merge them (resample average):

```
python directdemod/merger.py -o tutorial/data/merge/merged.tif -r average \
--files tutorial/data/merge/image1.tif tutorial/data/merge/image2.tif
```

The *tutorial/data/merge/merged.tif* file will be created after running the above command. You can compare it with other merging methods (*average.tif*, *max.tif*, *first.tif*, *last.tif*).

## 4.5 Map generation tutorial

To generate visualization of raster use *generate_map.py* interface. The following command will generate a TMS (Tile Map Service) and 2 visualization files in *samples/tms* directory.

```
python directdemod/generate_map.py --raster samples/decoded/raster.tif --tms samples/
↪tms
```

You can run *map.html* by opening it directly in the browser. To run *globe.html* go to tms directory and start the http server on port 8000 (python3):

```
python -m http.server 8000
```

Then open browser and go to *http://localhost:8000/globe.html*.

## 4.6 Help

If you encountered an error or want to add a fix, you can contact us directly on `github.com/aerospaceresearch/DirectDemod`.

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## d

# INDEX

## Symbols



## A



## B



## C



## D

# E



# F



# G



# H



# I



# L



# M



# O



# R

## S



## T



## U



## W