Udt Extractor

User Manual

UBERTONE S.A.S.
8A, rue Principale
67300 Schiltigheim
Tel : +33(0)3 67 100 883
E-mail: info@ubertone.fr
Internet: www.ubertone.fr

You are welcome to contact us!

Table of contents

1. Presentation **3**
2. Extracted data **3**
2.1. param_us_dicts 4
2.2. data_us_dicts 6
2.3. data_dicts 8
3. Usage example **9**
3.1. to read raw.udt file 9
3.2. to transform to Pandas dataframe 9
3.3. to plot a heatmap in Jupyter Notebook 10

1. Presentation

This is the user manual of the Udt Extractor library explains the format of the extracted data.

This data format is generated by extracting raw.udt binary files recorded by Ubertone's instruments through the https://github.com/ubertone/udt_extractor or https://github.com/ubertone/udt4_extractor library.

2. Extracted data

```
(   device_name,

    time_begin,

    time_end,

    param_us_dicts,

    data_us_dicts,

    data_dicts,

    settings_dict

) = raw_extract(path)
```

- **device_name**, if defined, gives the name of the device and its serial number.
- **time_begin** and **time_end** give the timestamps of the first measurement and of the last measurement, all config included
- **settings_dict** is the settings applied to the device for the concerned measurement sequence, with all the original parameter names. For a translated and homogene formatting to all instruments and versions, it is advised to use the **param_us_dicts**
- All the measured and recorded data for the concerned sequence (or run) are in the dictionaries **data_us_dicts** and **data_dicts**. We distinguish the data linked to the ultrasound measurement from the others, which could be measured in an asynchronous way, and which are not linked to a config number.

In param_us_dicts, data_us_dicts and data_dicts, we chose to uniformise the units and names of the variables (parameters or data) to get dictionaries with uniform keys, no matter which model of device or version of software was used to measure.

The keys we keep and the translations are listed in ***./udt_extract/translation.json***

2.1. param_us_dicts

Multi-level dictionary

Level 1 : number of configs [1..n], see GUI manual for the definition of a config (the device's Graphical User Interface).

Level 2 : number of channels {1..n}, ie. channel of the receiving transducer.

Level 3 : uniformised dictionary of parameters for a channel of a config.

Detail of level 3:

key string	description	type	unit
receiver	transducer channel used for the reception	string	
Common to all channels in a config			
emitter	transducer channel used for the emission	string	
f0	frequency of the ultrasonic wave sent into the medium	float	Hz
PRF	repetition frequency of the ultrasonic pulses	float	Hz
r_cell1	distance between the transducer surface and the first measurement cell. Translated from r_vol1	float	m
r_dcell	inter-cell distance: distance between two successive measurement cells. Translated from r_dvol, et calculé de r_dsubvol parfois	float	m
n_cell	number of cells in the profile. Translated from n_vol, et calculé de n_subvol parfois	int	
r_em	thickness of the measurement cell along the acoustic axis	float	m
n_p	number of pulses to calculate one instantaneous profile. Translated from n_ech	int	
phase_coding	phase coding activation	bool	
static_echo_filter	static echo filter activation	bool	
n_avg	number of profiles in one bloc, ie. To	int	

	calculate an average. Translated from n_profile		
v_min	start value for the velocity range	float	m/s
blind_ca0	constant coefficient of the blind zone gain limit	float	dB
blind_ca1	first degree coefficient of the blind zone gain limit	float	dB/m
a0	constant coefficient of the gain law. if key not available: gain auto activated.	float	dB
a1	first degree coefficient of the gain law. if key not available: gain auto activated.	float	dB/m
Common to all configs in a param_us_dicts			
operator		string	
comments		string	
sound_speed	sound speed used for the distances and velocity calculations.	float	m/s

Notes:

- temperature does not appear in this list as it is a parameter only given to the instrument to calculate the sound speed,
- if a parameter is not indicated or is variable in time, the field does not exist in param_us,
- if a parameter is variable in time, he may appear in the data dictionaries.

Example:

{

1: {

1: {'f0': 500000.0, 'emitter': 'tr1', 'prf': 137.32491073880803, 'r_cell1': 0.00723644499968, 'r_dcell': 0.02170933499904, 'n_cell': 200, 'r_em': 0.01447288999936, 'n_p': 16, 'phase_coding': True, 'static_echo_filter': False, 'n_avg': 4, 'v_min': -0.022866654440836246, 'blind_ca0': 1241, 'blind_ca1': 0, 'receiver': 'tr1', 'operator': 'sf', 'comments': 'mesure rhone - transect', 'sound_speed': 1447.288999936}

},

2: {

2: {'f0': 1000000.0, 'emitter': 'tr2', 'prf': 137.32491073880803, 'r_cell1': 0.00723644499968, 'r_dcell': 0.02170933499904, 'n_cell': 200, 'r_em': 0.01447288999936, 'n_p': 16, 'phase_coding': True, 'static_echo_filter': False, 'n_avg': 4, 'v_min': -0.022866654440836246, 'blind_ca0': 1241, 'blind_ca1': 0, 'receiver': 'tr2', 'operator': 'sf', 'comments': 'mesure rhone - transect', 'sound_speed': 1447.288999936}

},

3: {

2: {'f0': 1500000.0, 'emitter': 'tr2', 'prf': 137.32491073880803, 'r_cell1': 0.00723644499968, 'r_dcell': 0.022191764665685336, 'n_cell': 200, 'r_em': 0.01447288999936, 'n_p': 16, 'phase_coding': True, 'static_echo_filter': False, 'n_avg': 4, 'v_min': -0.02261595018348313, 'blind_ca0': 1241, 'blind_ca1': 0, 'receiver': 'tr2', 'operator': 'sf', 'comments': 'mesure rhone - transect', 'sound_speed': 1447.288999936}

},

4: {

3: {'f0': 2250000.0, 'emitter': 'tr3', 'prf': 137.3291015625, 'r_cell1': 0.00723644499968, 'r_dcell': 0.022191764665685336, 'n_cell': 200, 'r_em': 0.01447288999936, 'n_p': 16, 'phase_coding': True, 'static_echo_filter': False, 'n_avg': 4, 'v_min': -0.022615292454024567, 'blind_ca0': 1241, 'blind_ca1': 0, 'receiver': 'tr3', 'operator': 'sf', 'comments': 'mesure rhone - transect', 'sound_speed': 1447.288999936}

},

5: {

4: {'f0': 4500000.0, 'emitter': 'tr4', 'prf': 137.33329264198736, 'r_cell1': 0.0073168499441208885, 'r_dcell': 0.004985106555335111, 'n_cell': 200, 'r_em': 0.004985106555335111, 'n_p': 16, 'phase_coding': True, 'static_echo_filter': False, 'n_avg': 4, 'v_min': -0.022084214540871287, 'blind_ca0': 1241, 'blind_ca1': 0, 'receiver': 'tr4', 'operator': 'sf', 'comments': 'mesure rhone - transect', 'sound_speed': 1447.288999936}

}

}

2.2. *data_us_dicts*

Here are the recorded data (measured or variable parameters in time), which have a direct link to the ultrasound measurement.

Multi-level dictionary, with level 1 and 2 identical to ***param_us_dicts***, as those data are potentially directly linked to a config or even a channel.

Level 3: the keys are character strings representing the measured or recorded data types.

The standard types:

key string	Description	Linked to param_us	format	type	unit
echo_profile	Backscattered echo	yes	vector	float	V

or *echo_avg_profile*	amplitude profile, instantaneous or averaged over a block				
velocity_profile or *velocity_avg_profile*	Doppler velocity profile, instantaneous or averaged over a block	yes	vector	float	m/s
velocity_std_profile	Standard deviation profile over the average doppler velocity profile	yes	vector	float	m/s
snr_doppler_profile or *snr_doppler_avg_profile*	Doppler SNR profile (Signal to Noise Ratio), instantaneous or averaged over a block	yes	vector	float	dB
turbidity_profile or *turbidity_avg_profile*	Acousitc turbidity profile, instantaneous or averaged over a block	yes	vector	float	1/m
saturation_profile or *saturation_avg_profile*	- profile of 1 and 0 (for udt005 and instantaneous profiles of udt004 and previous: 1 as soon as one occurrence within the instantaneous) - percentage (for udt004 and previous, it is a percentage (convert in the scale 0..255) over the number of profiles.")	yes	vector	?	
ny_jump_profile or *ny_jump_avg_profile*	works the same are saturation profile	yes	vector	?	
sound_speed	named like this if the sound speed is processed from the acoustic measurement. Elsewise it should be stored under the key sound_speed_param (calculated from the temperature)	depend on the used method		float	m/s

noise_g__high	Average echo amplitude without emission at high gain (ex: gmax for UB-Lab P)	yes	scalar	float	V
noise_g_low	Average echo amplitude without emission at low gain (ex: gmax-10dB for UB-Lab P)	yes	scalar	float	V
XXX_param	parameters defined in the settings in automatic modus and are thus changing in time (ex: gain_a0_param or sound_speed_param)	yes	scalar	param dependent	

Note :

- "*_avg*" : average in the general meaning: the velocities are obtained by arithmetic averaging and the echoes by quadratic averaging
- "*profile*" : allows the distinction between vectors and scalars
- the suffix "*_param*" allows the distinction with variable parameters (modified automatically by the system.
  Some parameters may vary along time (ex: mode auto). They have no fixed value in the settings and their evolution during the sequence is sometimes recorded by the instrument; they may then be found as a scalar in the data dictionaries with type name "**_param*".
- keys for types are lowercase and the only allowed special character is '_'.

Level 4:

For each type, there is a two-key dictionary:

- "*data*" : list or time series of the concerned data, which may be a scalar or a profile (vector).
- "*time*" : list of UTC timestamps

2.3. data_dicts

Here are the recorded data (measured or variable parameters in time), which have no direct link to the ultrasound measurement.

The first level is here directly associated to the data type. There is not config or channel key.

Level 2 corresponds to the level 4 of the ***data_us_dicts***.

Standard types:

key	Description	Linked to param_us	format	type	unit
roll		no	scalar	float	rad
pitch		no	scalar	float	rad
temperature		no	scalar	float	K

3. Usage example

3.1. to read raw.udt file

```
python ./test.py
```

3.2. to transform to Pandas dataframe

Example for acoustic backscattered echo amplitude data from UB-Lab P:

```
data_type = 'echo_avg_profile'

df = {}

for config in param_us_dict.keys():

      channel = list(param_us_dicts[config].keys())[0]

      positions       =       ar(param_us_dicts[config][channel]["r_cell1"]       +
ar(range(param_us_dicts[config][channel]["n_cell"]))                              *
param_us_dicts[config][channel]["r_dcell"])

      df[config] = pd.DataFrame(
            data_us_dicts[config][channel][data_type]['data'],

            index=data_us_dicts[config][channel][data_type]['time'],

            columns=positions)
```

3.3. to plot a heatmap in Jupyter Notebook

Example for acoustic backscattered echo amplitude data from UB-Lab P:

```
fig = make_subplots(
	rows=len(param_us_dicts.keys()),	cols=1,	shared_xaxes=True,
vertical_spacing=0.01
	)

i = 1
for config in param_us_dicts.keys():
	print(config)
	channel = list(param_us_dicts[config].keys())[0]
	fig.add_trace(
		go.Heatmap(
			x=np.array(data_us_dicts[config][channel][data_type]["time"]),
			y=ar(param_us_dicts[config][channel]["r_cell1"] +
ar(range(param_us_dicts[config][channel]["n_cell"])) *
param_us_dicts[config][channel]["r_dcell"]),
			z=np.array(data_us_dicts[config][channel][data_type]["data"]).T
,
			name=str(config)
		),
		row=i, col=1
	)
	fig.update_yaxes(title_text=data_type, type="log", row=i, col=j)
	i = i+1
fig.update_xaxes(title_text="Distance <br> (in m)", row=len(param_us_dicts.keys()),
col=1)
fig.show(renderer="notebook")
```