CUDRDC Documentation

Release 0.1.4

Jason Gao

Jul 21, 2019

CONTENTS

CHAPTER

ONE

ABOUT PROJECT

This project is funded by both the Defense Research and Development Canada (DRDC) and the National Research Council (NRC). These organizations are working alongside Carleton University students to develop a pip installable library that contains all the parsers used to convert radar and lidar data to readable csv files. The library also contains methods for determining radar thresholds. The radars used in this project are the Novelda X4M03, X4M300, X4M200 and the TSW1400. The lidar used is the Ouster OS1-16.

Contents:

1.1 Radar Information

1.1.1 Novelda X4

The X4 radars are IR-UWB and can work at frequencies ranging from 6 GHz to 10.2 GHz. The total number of bins that can be sampled is 1536.

Specifications

X4M300 Specs

- Detection Time: 1.5 - 3.0 seconds
- Range: 9.4 meters
- Antenna: Tx for transmission and Rx for receiving
- Baseband data output: 17 baseband/ssecond
- System on chip: Novelda UWB X4

X4M200 Specs

- Detection Time: 3.0 - 5.0 seconds
- Range: 5 meters
- Antenna: Tx for transmission and Rx for receiving
- Baseband data output: 17 baseband/ssecond
- System on chip: Novelda UWB X4

Configuring X4 radar

1. Begin by initializing to default values using prebuilt function *x4driver_init()*
2. Set PRF using function *x4driver_set_prf_div(...)*

Note: The common PLL value of 243 MHz is divided by the arguemnent passed in to *x4driver_set_prf_div(...)* to get a PRF value

---

Note: Make sure that when changing the PRF that frame length is shorter than 1/PRF and avoid sampling previous pulse when transmitting next pulse.

---

3. Set DAC sweep range minimum and maximum using *x4driver_set_dac_min()* and *x4driver_set_dac_max()*
4. Set 0 reference using *x4driver_set_frame_area_offset()*
5. Set frame area using function *x4driver_set_frame_area()* that takes two arguements, one for start of frame and one for end of frame.

Setting radar FPS

To set the radar FPS the following parameters are required, PRF, iterations, pulse per step, dac max and dac min range as well as duty cycle.

$$FPS = \frac{PRF}{iteration * pulse_per_step * (dac_max - dac_min + 1)} * dutycycle$$

Our Novelda radar is configured to a FPS of 17 pulse/second so if you wanted to change FPS then the above parameter would need to be changed.

Note: The resulting FPS can be read using the built-in function *x4driver_get_fps()*.

Example pulse_per_step calculation

- PRF: 16 MHz
- X4_duty_cycle: 95%
- dac_max: 1100
- dac_min: 949
- iteration: 64
- FPS: 17

$$pulse\_per\_step = \frac{PRF}{iteration * FPS * (dac_max - dac_min + 1} * D$$
$$pulse\_per\_step = \frac{16MHz}{64 * 17 * 150} * 0.95$$
$$pulse\_per\_step = 87$$

1.1.2 TSW1400

The TSW1400 board is used to interface with TI radars.

Specifications

- Operates using 5 V power source and is controlled by the SW7 switch.
- 11 LEDS used to indicate presence of power and state of FPGA.
- Control of the TSW1400 is via USB cable to a Windows PC.

Refer to `TSW1400 Comprehensive Guide` for full guide on hardware and software installations

Required softwares

Google Drive download

1.2 X4 Radar

1.2.1 Parser for iq data

`X4_parser.`**`iq_data`**(*filename*, *csvname*)

Reads in a binary file and data from range bins is taken and complex iq data is stored in a csv file specified by csvname.

Parameter:

filename: str The .dat binary file name.

csvname: str User defined .csv file name

Example:

```
>>> iq_data('X4data.dat','X4iq_data')
>>> 'converted'
```

Returns:

Readable csv file containing complex values.

1.2.2 Parser for raw data

`X4_parser.`**`raw_data`**(*filename*, *csvname*)

Reads in a binary file and data from range bins is taken and raw data is stored in a csv file specified by csvname.

Parameters:

filename: str The .dat binary file name.

csvname: str User defined .csv file name

Example:

```
>>> raw_data('X4data.dat','X4raw_data')
>>> 'converted'
```

Returns:

Readable csv files containing raw data.

1.2.3 X4 Record and playback code

Target module: X4M200,X4M300,X4M03

Introduction:

XeThru modules support both RF and baseband data output. This is an example of radar raw data manipulation. Developer can use Module Connecter API to read, record radar raw data, and also playback recorded data.

Command to run: *python X4_record_playback.py -d com3-b -r*

- *-d com3* represents device name and can be found when starting Xethru Xplorer.
- *-b* to use baseband to record, default is radio frequency.
- *-r* to start recording.

X4_record_playback.**clear_buffer**(*mc*)
Clears the frame buffer

Parameter:

mc: object module connector object

X4_record_playback.**main**()
Creates a parser with subcatergories.

Return:

A simple XEP plot of live feed from X4 radar.

X4_record_playback.**on_file_available**(*data_type*, *filename*)
Returns the file name that is available after recording.

Parameters:

data_type: str data type of the recording file.

filename: str file name of recording file.

X4_record_playback.**on_meta_file_available**(*session_id*, *meta_filename*)
Returns the meta file name that is available after recording.

Parameters:

session_id: str unique id to identify meta file

filename: str file name of meta file.

X4_record_playback.**playback_recording**(*meta_filename*, *baseband=False*)
Plays back the recording.

Parameters:

meta_filename: str Name of meta file.

baseband: boolean Check if recording with baseband iq data.

X4_record_playback.**reset**(*device_name*)
Resets the device profile and restarts the device

Parameter:

device_name: str Identifies the device being used for recording using it's port number.

X4_record_playback.**simple_xep_plot**(*device_name*, *record=False*, *baseband=False*)

Plots the recorded data.

Parameters:

device_name: str port that device is connected to.

record: boolean check if device is recording.

baseband: boolean check if recording with baseband iq data.

Return:

Simple plot of range bin by amplitude.

1.2.4 X4 Threshold detection

To use these functions first take the data recorded from the X4 radar and pass it into the iq_data() function found in X4_parser.py to get a comlex csv file. The file received will be used wherever *filename* is an arguement.

X4_threshold.**csv_into_list**(*filename*)

Converts data from a CSV file into a numpy array.

Data is collected from "XEP_X4M200_X4M300_plot_record_playback.py". Then .dat file is converted into a CSV file using the library.

Parameters:

filename: str Csv file name.

Example:

```
>>> csv_into_list("Heli150040.csv")
>>> array([[2.12019056e-04, 2.66481591e-04, 3.51781533e-04, ...,
    2.51499279e-04, 2.63311858e-04, 2.49837321e-04],
   [1.36839763e-03, 1.13654408e-03, 7.92290507e-04, ...,
    3.49299137e-04, 1.30311682e-04, 1.87399805e-04],
   [7.57068081e-04, 6.54611670e-04, 6.83445863e-04, ...,
    6.95117789e-05, 2.17249015e-04, 3.85946482e-04],
   ...,
   [9.57032957e-04, 5.99047943e-04, 5.37280418e-04, ...,
    2.95999810e-04, 2.42217122e-04, 3.25605109e-04],
   [2.37110777e-03, 1.98986895e-03, 1.28061167e-03, ...,
    2.95363991e-04, 1.50057104e-04, 2.33757752e-04],
   [6.45687293e-04, 5.61334516e-04, 1.63285784e-04, ...,
    6.64671904e-05, 2.17249015e-04, 1.93614790e-04]])
```

Returns:

A numpy array of the data.

X4_threshold.**distance_finder**(*filename*, *estimate_threshold*)

Converts the positive range bin/bins to a given distance in centimetres for a target. Formula used for this is (bin)*5.25-18. Each range bin is 5.25 cm and starting offset is 18 cm.

Parameters:

filename: str Csv file name.

estimated_threshold: float Threshold to block out background noise.

Example:

```
>>> distance_finder("Heli150040.csv",0.02)
>>> [29.25]
```

Returns:

List of ranges in centimetres where all the possible targets above the threshold are located.

X4_threshold.**noise_power_estimate**(*filename*, *estimated_threshold*)
Finds the noise power estimate.

Given the filename, the file is converted to an array. The average for the the array is taken excluding the points above the threshold and their respective guard cells. The noise power estimate is then found by subtracting the positive and guard cells from the overall sum. When the noise power estimate is multiplied by the threshold factor, the threshold can be found.

Parameters:

filename: str Csv file name.

estimated_threshold: float Threshold to block out background noise.

Example:

```
>>> noise_power_estimate("Heli150040.csv",0.02)
>>> 0.00045988029076158947
```

Returns:

A number representing the noise power estimate.

X4_threshold.**plot_data**(*filename*)
Plots the data array for the 5th sample set.

Parameters:

filename: str Csv file name.

Example:

```
>>> plot_data("Heli150040.csv")
```

Returns:

Graph showing the range bin with respect to their corresponding strength of signal.

X4_threshold.**range_finder**(*filename*, *estimated_threshold*)
Finds the range bin/bins from the radar data.

Given the filename, the file is converted to an array. The points above the given threshold are returned.

Parameters:

filename: str Csv file name

estimated_threshold: float Threshold to block out background noise.

Example:

```
>>> range_finder("Heli150040.csv",0.02)
>>> [9]
```

Returns:

A list of range bins that has signal strength values above the threshold.

1.3 TSW parser code

TSW_IWR.**readTSWdata**(*filename*, *csvname*)

Reads in a binary file and outputs the iq complex data to a csv file specified by csvname.

Parameter:

filename: str file name of binary file.

csvname: str csv file name that stores the iq data from binary file.

Example:

```
>>> readTSWdata('TIdata.bin','TIdata')
>>> 'converted'
```

Return:

Readable csv file containing complex data.

1.4 Library GUI

This GUI allows user to run parsers that are included in this library such as the ones for the Novelda X4 radars, TSW1642 radars and Ouster OS1-16 lidar. This GUI allows users to open their desired file to read and with the use of buttons, read the desired data. The GUI even has a instruction page on how to use it.

1.4.1 Functions

Below, are the functions that are used and run in the GUI.

library_gui.**command**(*entry*)

Takes in a number of rows to be read from file and maps to list of ints.

Paramters:

entry: int The row numbers that will be output.

Return:

mapped list of data

library_gui.**convert_TSW**()

Converts the TSW binary file to a readable csv file.

library_gui.**convert_x4**()

GUI window with button to allow conversion of X4 binary file to csv file.

Return:

Readable X4 csv file.

library_gui.**imu_multiple_row**()

GUI window for reading IMU data of multiple rows

Return:

A textbox of the parameter data user wanted to read.

library_gui.**imu_row_section**()
GUI window for reading IMU data of the row section

Return:

A textbox of the parameter data user wanted to read.

library_gui.**imu_single_row**()
GUI window for reading IMU data of the single row

Return:

A textbox of the parameter data user wanted to read.

library_gui.**instruction**()
A set of instructions on how to use the program

library_gui.**lidar_multiple_row**()
GUI window for reading lidar data of multiple rows

Return:

A textbox of the parameter data user wanted to read.

library_gui.**lidar_row_section**()
GUI window for reading lidar data of the row section

Return:

A textbox of the parameter data user wanted to read.

library_gui.**lidar_single_row**()
GUI window for reading lidar data in a single row

Return:

A textbox of the parameter data user wanted to read.

library_gui.**open_TSW_bin**()
Opens the TSW binary file.

library_gui.**open_x4_bin**()
Opens the Novelda X4 binary file for reading.

library_gui.**open_x4_csv**()
Opens the Novelda X4 csv file for use in threshold functions.

library_gui.**openfile**()
Opens the csv file for reading lidar and IMU packet parameters.

library_gui.**print_list**(*lst*)
Takes in an array and prints it to textbox.

Parameters:

lst: list A list object of the data that will be read to user.

library_gui.**x4_threshold**()
GUI window to run X4 threshold functions

Returns:

Radar threshold, noise and range estimates

library_gui.**xyz_calc**()
GUI window for reading the xyz coordinates gotten from lines of the lidar csv file.

Return:

A textbox of the xyz coordinates from rows of the lidar csv file.

1.5 Linux setup

To install Linux with Ubuntu v18.04 on a Windows PC, users must install the following files:

VirutalBox Software

Ubuntu 18.04 download

Quick tutorial on installing Ubuntu

1.5.1 Quick tips for linux beginners

- Set the network setting to use bridged adapter so Linux doesn't share the same ip address as your Windows PC.
- Insert guest addition CD image found in the *Devices* tab for auto-adjusted screen resolutions.
- Use *~/ to cd from home directory*

1.6 Ouster OS1 Lidar

1.6.1 Specs

All below specs are for the OS1-16 lidar that was used in this project - Works on channels 16, 64, 128. - Maximum range of 120 meters. - Field of view of 33.2 degree vertically and 360 degree horizontally. - Sampling rate of 327,680 points/second.

1.6.2 Setup lidar

1. Connect lidar interface box to router that supports Gigabit connection.
2. Connect lidar to lidar interface box via cable.
3. Determine the ip address your router gave the lidar when it connected to the network and jot it down.
4. Determine your linux ip adress by running *ifconfig* in terminal and jot it down.

1.6.3 Ouster Github

The following Github page provides information on how to view raw data streams, visualize data and use a robot operating system (ROS) to save recorded data in a .bag file. ROS commands can also replay data in .bag files and convert .bag files to .csv files.

Note: Some version of Linux running Ubuntu must be used. It is recommended to run Ubuntu 18.04 for best results. Follow instructions in *Ubuntu* page for more details on installing Linux with Ubuntu.

Variable reference

<os1_hostname> is the hostname/ip address of OS1-16 lidar.

<udp_data_dest_ip> is the destination ip address the lidar sends data

<frame_size> is the size of the visualization frame and can ONLY be the following: 512x10, 512x20, 1024x10, 1024x20, 2048x10.

Ouster client

The Ouster client allows users to see the raw data stream that the lidar is collecting and sending to the specified ip address. Instruction on building the client in Linux can be found here Building client.

Running client

1. cd /path/to/ouster_client_example
2. type *./ouster_client_example <os1_hostname> <udp_dest_ip>*

Ouster visualization

The Ouster visualization is used for building a basic visualizer frame of collected lidar data. Instructions on building visualizer and it's dependencies can be found here Building visualizer. The visualizer can be run in real time or with recorded data.

Running visualizer

1. cd /path/to/ouster_viz/build
2. *./viz -m <frame_size> <os1_hostname> <udp_data_dest_ip>*

Ouster ROS

Note: For Ubuntu 18.04 users it is best to use **ROS Melodic** as **ROS Kinetic** (The ROS provided on the GitHub page) is only compatible with Ubuntu 16.04 and lower.

Building the ROS Node can be found here Building ROS Kinetic.

For Ubuntu 16.04 users and lower: Installation of ROS Kinetic

For Ubuntu 18.04 users: Installation of ROS Melodic

For new users to using ROS: ROS Tutorials

Running ROS Node

Note: Before typing any commands make sure to always source the setup.bash file in your created ROS workspace otherwise it will return a error. The file can be sourced with the command *source /path/to/myworkspace/devel/setup.bash*.

For recording lidar data:

1. *roslaunch ouster_ros os1.launch os1_hostname:=<os1_hostname> os1_udp_dest:=<os1_udp_dest> lidar_mode<:=<lidar_mode>*. The option to visualize live data can be turned on by adding *viz:=true* to the roslaunch command.
2. *rosbag record -O <recorded__bag_filename> /os1_node/imu_packets /os1_node/lidar_packets* in a new terminal

/os1_node/imu_packets and */os1_node/lidar_packets* are your topic names that the lidar sends messages to via the node you built. These topic names can be changed to user preference.

Note: DO NOT close the terminal with the roslaunch command open otherwise rosbag will crash.

For replaying lidar data:

1. *roslaunch ouster_ros os1.launch replay:=true os1_hostname:=<os1_hostname>*
2. In a **new** terminal run *rosbag play <bag_filename>*

Note: DO NOT close the terminal with the roslaunch command open otherwise rosbag will crash.

Converting data to csv file: Run *rostopic echo "topic name" -b "bag_filename" -p > filename.csv*

Note: To find topic names run the command *rosbag info <bag_filename>*

1.7 Test file

class test.**TestParser**(*methodName='runTest'*)

X4_Threshold_bin_to_distance()
Method to test if range bin to distance converted correctly

X4_Threshold_noise_estimate()
Method to test if noise estimate was calculated properly

X4_Threshold_range_finder()
Method to test if correct range bin was gotten from running function on csv file.

test_TI()
Method to test if .bin binary file was converted successfully to .csv file with iq data put together.

test_iq()
Method to test if .dat binary file was converted successfully to .csv file with in-phase and quadrature components together.

test_raw()
Method to test if .dat binary file was converted successfully to .csv file with in-phase and quadrature component separated.

PYTHON MODULE INDEX

l

t

x