CUDRDC Documentation

Release 0.0.7

Jason Gao

Jun 27, 2019

CONTENTS

CHAPTER

ONE

ABOUT PROJECT

This project is funded by the Defense Research and Development Canada (DRDC) and they are working alongside Carleton University to develop algorithms for radar and lidar sensors. The radars used in this project are Novelda X4, Thor WFS30-K1, Ouster lidar and Advacam Minipix TPX3.

Contents:

1.1 Radar Information

1.1.1 About X4 radar

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.

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

1.1.2 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.

1.1.3 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

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

1.2 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

1.2.1 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 to home directory*

1.3 X4 Radar

1.3.1 Parser for iq data

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

Takes binary data file and iterates through in-phase (real) and quadrature (imaginary) values. Data from range bins is taken and in-phase values are matched with quadrature values to be stored in a user defined .csv file.

Parameter

filename: str The .dat binary file name.

csvname: str User defined .csv file name

Example

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

Returns

In-phase and quadrature pairs stored together in a .csv file.

1.3.2 Parser for raw data

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

Takes raw data file and iterates through in-phase (real) and quadrature (imaginary) values. Data from range bins is taken, and in-phase value are put apart from quadrature in a user defined .csv file.

:parameter

filename: str The .dat binary file name.

csvname: str User defined .csv file name

Example

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

Returns

In-phase and quadrature stored separately in a .csv file.

1.3.3 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 com4 -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.

Parameter:

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 with 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 over time.

1.4 TI parser code

`TI_parser.`**`readTIdata`**`(`*filename*, *csvname*`)`

Takes a .bin binary file and outputs the iq 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

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

Returns

A csv file with the iq data taken from the binary file.

1.5 Ouster OS1 Lidar

1.5.1 Specs

All below specs are for OS1-16 lidar that was used in this project

- Works on channel 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.

1.5.2 Setting up 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.

1.5.3 Ouster Github

The following Github page was provides information on how to view raw data, visualized lidar data and use a ROS to save recorded data in .bag files and later parse to a .csv file. The ROS can also be used for replaying data.

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.

Ouster client

The Ouster client allows users to see the raw data 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>*

<os1_hostname> is hostname/ip address of lidar.

<udp_data_dest_ip> is the destination ip address the lidar sends data to. e.g. ip address from running *ifconfig*.

Ouster visualization

The Ouster visualization is used for building a basic visualizer frame of collect lidar data. Instructions on building visualizer and it's dependencies can be found here Building visualizer.

Running visualizer

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

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

<os1_hostname> is the ip address of lidar.

<udp_data_ip_dest> is the ip address lidar sends data to.

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 Buidling ROS Kinetic.

For Ubuntu 16.04 users and lower: Installation of ROS Kinetic

For Ubuntu 18.04 users: Installation of ROS Melodic

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 before setting the variables.

<os1_hostname> is the ip address of the lidar

<os1_udp_dest> is the ip address the lidar sends data to

<lidar_mode> is the size of the lidar visualization frame

2. *rosbag record /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 nodes 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.

The recorded data will be stored as a .bag file and can be used for playback.

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.txt*

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

1.6 Test file

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

> **`test_TI`**`()`
>
> Method to test if .bin binary file was converted successfully to .csv file with iq data put together.
>
> **Returns**
>
> converted
>
> **`test_iq`**`()`
>
> Method to test if .dat binary file was converted successfully to .csv file with in-phase and quadrature components together.
>
> **Returns**
>
> converted
>
> **`test_raw`**`()`
>
> Method to test if .dat binary file was converted successfully to .csv file with in-phase and quadrature component separated.
>
> **Returns**
>
> converted

```
Convert X4 binary .dat file to csv
```

```
X4 data collection code
```

```
Convert TI binary .bin file to csv
```

PYTHON MODULE INDEX