Sensor Documentation

Release 0.0.5

Jason Gao

Jun 21, 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 X4M300, Novelda X4M200, Thor WFS30-K1 and Advacam Minipix TPX3. Use the table of contents below to find out more about the radar specs, library parser and radar configuration.

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 X4 Parser code

1.2.1 Parser for in-phase and quadrature data combined

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.

1.2.2 Parser for in-phase and quadrature data seperated

`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 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.

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

PYTHON MODULE INDEX