gr-plasma: A New GNU Radio-based Tool for Software-defined Radar

Shane Flandermeyer SHANE.FLANDERMEYER@OU.EDU
School of Electrical and Computer Engineering, Advanced Radar Research Center, University of Oklahoma

Rylee Mattingly RMATTINGLY@OU.EDU
School of Electrical and Computer Engineering, Advanced Radar Research Center, University of Oklahoma

Justin Metcalf JMETCALF@OU.EDU
School of Electrical and Computer Engineering, Advanced Radar Research Center, University of Oklahoma

Abstract

A low-cost experimental setup is an invaluable tool for rapid prototyping of radar signal processing algorithms. Here, the `gr-plasma` out-of-tree module is presented as a convenient way to collect radar data using UHD-compatible software-defined radio (SDR) systems. The module operates entirely over the GNU Radio message passing interface, packetizing each pulse received by the radar into a protocol data unit (PDU) to simplify downstream processing and minimize latency. It includes blocks for waveform generation, monostatic transmission and reception, and data storage, and supports both CPU and GPU backend processing. All signal processing functionality in `gr-plasma` is an implementation of `plasma_dsp`, a separate companion library that can also be used for projects outside the GNU Radio ecosystem. In this paper, the utility and structure of each block is discussed, and the results of range-Doppler processing are shown for data collected with gr-plasma and an Ettus X310 in an open-air test.

1. Introduction

Building experimental radar systems has traditionally required expensive custom hardware. Such designs are often difficult to program or modify, preventing easy experimentation with new radar signal processing algorithms. While SDR hardware is not inherently designed to operate as a radar, there have been numerous documented examples of software-defined radar implementations on commercial-off-the-shelf systems, such as Christiansen & Smith (2019) and Wunsch (2014). This work aims to combine the best

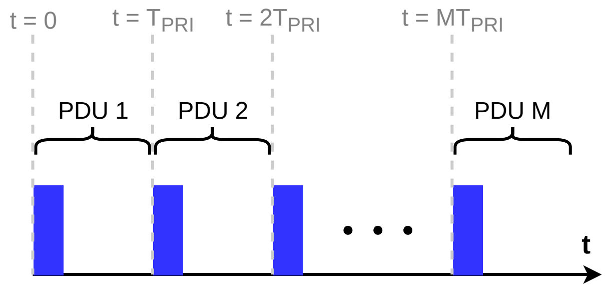


Figure 1. Pulsed radar data structure

features from previous projects to facilitate experimentation by both academics and hobbyists while making it easy for developers to extend the module to other use cases.

There are two key challenges with software-defined radar that differ from most communications protocols: transmit/receive synchronization and ultra-reliable high-speed duplex operation. Radar systems operate by transmitting electromagnetic waves into the environment and measuring their reflections off of objects. To achieve localization accuracy, the transmit and receive processes must be synchronized down to the sample level. The pulsed radar developed for this work maintains precise knowledge of the beginning and end of each pulse along with the beginning of the receive interval. Since the range resolution of a radar sensor is inversely proportional to the bandwidth of the emitted waveform, the radar module must also be capable of streaming data at high sample rates in order to maximize performance.

This work will demonstrate the utility of GNU Radio's message passing interface in achieving these goals. Messages are well-suited for bursty data, and they make it easy to determine the bounds of each radar pulse compared to traditional stream buffers that are not large enough to process entire pulses of data unless the pulse duration is short or the sample rate is sufficiently small. Each pulse repetition interval (PRI) of received radar data can then be

Proceedings of the 1st GNU Radio Conference, Copyright 2016 by the author(s).

packaged into a protocol data unit (PDUs) as shown in Fig. 1, which simplifies processing by providing a standardized data format for downstream signal processing blocks. Message passing also increases the maximum sample rate that can be achieved for USRP devices (compared to the in-tree UHD source and sink blocks) because timing and data throughput limitations can be managed with no reliance on the scheduler.

The remainder of this paper is outlined as follows: Section 2 outlines the signal processing blocks that are implemented in `gr-plasma`, along with select features in `plasma_dsp` that are useful for development. Section 3 describes an open-air testing setup used for collecting radar data, including system specifications and required software. Section 4 shows the results of this experiment along with range-doppler maps that were generated during data collection. Finally, Section 5 outlines the conclusions and presents future plans for the `gr-plasma` module.

2. Blocks and Features

2.1. Waveform Generators

`gr-plasma` currently implements two types of radar waveforms. The first is a linear frequency-modulated waveform (LFM), which is the most common waveform used in pulsed radar systems due to its simplicity in implementation and desirable pulse-Doppler ambiguity properties. As the name implies, an LFM "chirp" sweeps linearly across its frequency range over a finite time duration. In `gr-plasma`, the Linear FM block generates an LFM at complex baseband (i.e., centered around 0 Hz) that sweeps upwards from the lower frequency to the higher frequency, or

$$x(t) = \exp\left[j\pi\left(-Bt + \frac{B}{T_p}t^2\right) + \theta_0\right] \qquad (1)$$

where B is the bandwidth of the waveform, T_p is the duration of the frequency sweep, and θ_0 is an arbitrary starting phase. The time-domain output of the LFM block is shown in Fig. 2 for a waveform with $B = 40\,\text{MHz}$ and $T_p = 20\,\mu\text{s}$.

`gr-plasma` can also generate polyphase-coded FM (PCFM) waveforms, which were first introduced in (Blunt et al., 2014a) and (Blunt et al., 2014b). PCFM waveforms are based on the continuous phase modulation (CPM) scheme from the communications literature, which maps a sequence of data symbols into a nearly continuous constant-amplitude waveform. Rather than modulating symbols in a data stream, PCFM modulates discrete sequences of phase values known as phase codes (Levanon & Mozeson, 2004). A waveform can be formed from a length-N_c code by transmitting each code value for some amount of time T_c, producing a waveform of length T_p (Fig. 3).

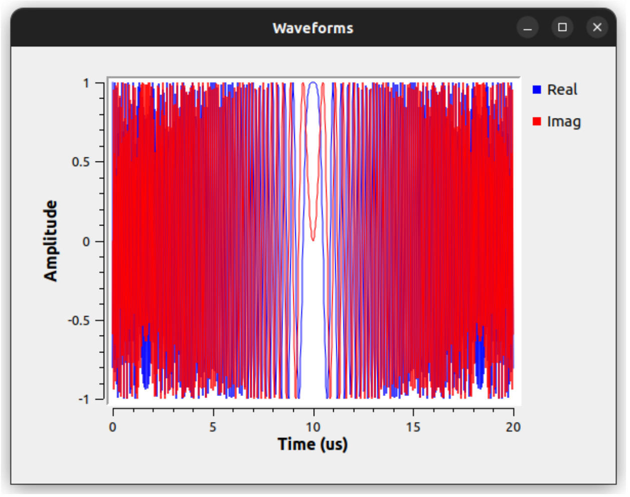


Figure 2. LFM waveform in the time domain

Unlike traditional waveforms (such as the LFM) which are a function of only a few parameters, phase-coded waveforms are parameterized by the individual phase values in the code, giving them many degrees of freedom that can be used for optimization. However, due to the instantaneous transitions between phase values, the resulting waveform will be distorted when it is passed through a power amplifier operating in the saturation region. PCFM addresses this shortcoming by applying a smoothing filter to the waveform's phase trajectory, resulting in an emission that is (approximately) both continuous and constant modulus and thus amenable to transmission through a high-power amplifier (Blunt et al., 2014a).

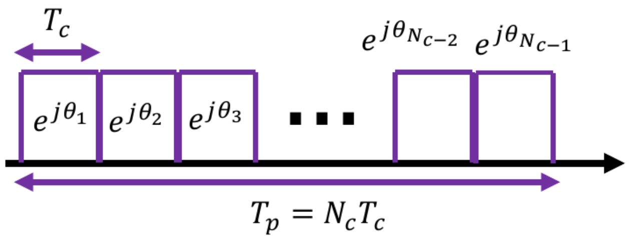


Figure 3. Forming a phase-coded waveform from individual phase values

The PCFM Source block in `gr-plasma` implements first-order PCFM (Tan et al., 2015), which linearly interpolates to K intermediate values between each phase code value on the unit circle. Mathematically, the phase function at complex baseband resulting from phase code $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_{N_c}]$ can be described as

$$\psi(t;\boldsymbol{\theta}) = \int_0^t w_c(\lambda) * \sum_{n=0}^{N_c-1} \delta(t - nT_c)\alpha_n d\lambda + \theta_0 \qquad (2)$$

where $w_c(t)$ is a unit-energy shaping filter and α_i is instantaneous frequency, which is the difference between successive phase code values (wrapped into the range $[-\pi, \pi]$) with $\alpha_0 = 0$. In the PCFM source block, w_c is rectangular with length K and Barker, Frank, and P4 codes are currently supported (Fig. 4).

2.2. USRP Interfacing and Control

Unlike in communications systems, a radar's transmitter and receiver are often co-located and managed by the same host system (known as a monostatic configuration). Therefore, `gr-plasma` uses a single custom block to handle both the transmission and reception of radar pulses for UHD-compatible SDR devices. Unlike the USRP Source and Sink in the main tree, the USRP Radar block operates entirely in the message domain. The block operates as follows: first, a waveform must be passed to the input port as a PDU. The metadata dictionary in this PDU must contain an item with a key called `prf` whose value defines the pulse repetition frequency (PRF) at which the waveform is to be transmitted. The PRF tag can be added to any PDU using the Waveform Controller block. Although the Waveform Controller block is currently only used to set the PRF, it will eventually be extended to facilitate cognitive operation in which it uses the results of downstream processing to update the transmitted waveform.

When the PDU has been processed, the radar block will repeatedly transmit and receive the waveform until a new PDU is received. This method minimizes the number of messages that must be processed by the block and makes it possible to operate at higher sample rates since no overhead is incurred from the scheduler. It also reduces latency during processing compared to a stream-based approach since messages can be processed immediately when they are received, making it possible to update the waveform on a sub-millisecond timescale. Moreover, unlike in tagged stream blocks, the size of the waveform is not limited by the size of each block's buffer. Since radar returns are often processed on a pulse-by-pulse basis, the received data is processed into pulses and passed to the output as a new PDU. Metadata from upstream blocks is only propagated on the first received pulse for which it applies in order to further minimize the amount of data to be processed by downstream blocks. The complete USRP processing chain is summarized in Fig. 5.

One hardware quirk that had to be addressed before the USRP radar could be used for open-air experiments was the delay between transmit (Tx) and receive (Rx) ports due to the internal DSP of each device, which manifests as a range offset for all targets during processing. This delay is constant for a given device, sample rate, and master clock rate, and can thus be accounted for in a pre-processing step. The delay is computed using a streamlined version of the Estimator Sync Pulse block from `gr-radar`: with the Tx port connected directly to the Rx port (with a 30 dB attenuator in the loop), a known reference pulse is transmitted and then cross-correlated with the received data, and the first peak is taken to be the delay. The calibration can be performed for any number of sample rates. For example, a B210 may be calibrated for 10 MS/s and 20 MS/s with the following command

```
plasma_calibrate_delay --rates 10e6
20e6 --filename $HOME/delay.json
```

```
"B210": [
  {
    "delay": 46,
    "master_clock_rate": 40000000.0,
    "samp_rate": 10000000.0
  },
  {
    "delay": 165,
    "master_clock_rate": 20000000.0,
    "samp_rate": 20000000.0
  },
]
```

Listing 1. Delay calibration output file format

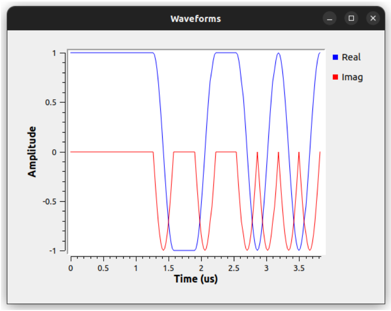
The output of the calibration script is then saved to a JSON file that stores a list of delays, master clock rates, and sample rates for each device configuration that has been calibrated. For the example above, the JSON in Listing 1 would be produced. The USRP radar block takes this file as an input, and skips the first N_{delay} received samples if the configuration is present in the file.

2.3. Signal Processing Blocks

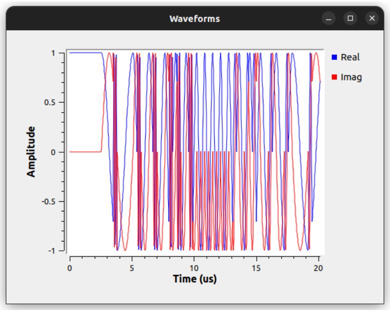
Matched filtering, also known as pulse compression in radar and sonar applications, is a method for detecting a signal in the presence of comparatively high-power noise. The goal of matched filtering is to maximize the signal-to-noise ratio (SNR) at the filter output for a given transmitted waveform. It can be shown that for a waveform $x_{tx}(t)$ with duration τ_p in additive white Gaussian noise (AWGN), the matched filter's impulse response is given as (Richards, 2013)

$$h(t) = \alpha x^*_{tx}(T_m - t) \tag{3}$$

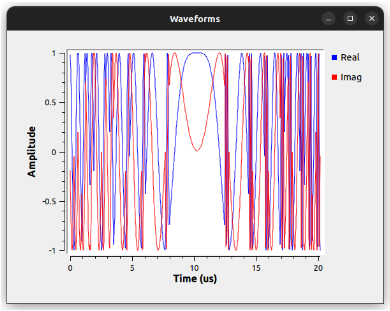
where scaling constant α has no impact on SNR and is often set to unity, and T_m is the time at which the SNR is maximized. T_m can also be any value, but for causality it is required that $T_m \geq \tau_p$ (in the matched filter GNU Radio block, it is assumed that $T_m = \tau_p$). Since $h(t)$ is a



(a) Barker Code ($N_c = 13$, $K = 16$)



(b) Frank Code ($N_c = 64$, $K = 16$)



(c) P4 Code ($N_c = 64$, $K = 16$)

Figure 4. PCFM waveforms generated from common phase codes

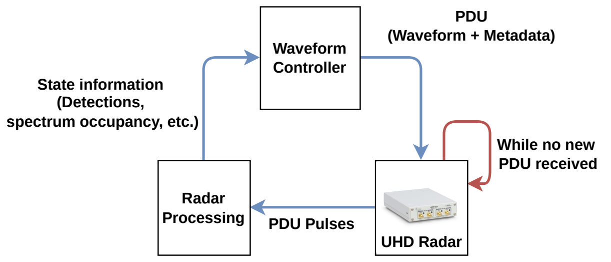
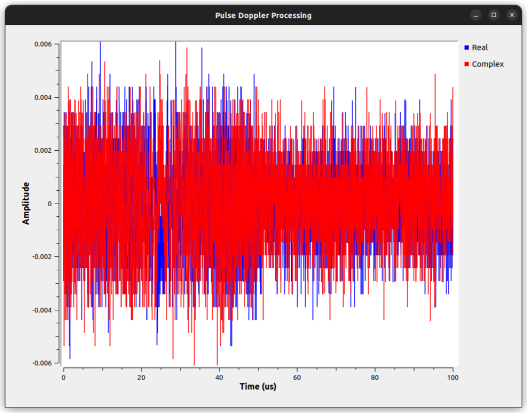


Figure 5. USRP radar operational logic

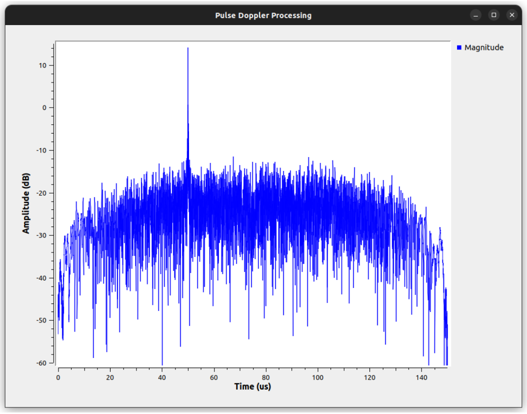
time-reversed, complex conjugated version of the $x_{tx}(t)$, it is said to be "matched" to $x_{tx}(t)$. Performing a convolution of the matched filter with some received signal $x_{rx}(t)$ gives

$$\begin{aligned}(x_{tx} * x_{rx})(\tau) &= \int_{-\infty}^{\infty} h(\tau - t)x_{rx}(t)dt \\ &= \int_{-\infty}^{\infty} x_{tx}^*(t-\tau)x_{rx}(t)dt \\ &= (x_{tx} \star x_{rx})(\tau)\end{aligned} \tag{4}$$

where $a(t) \star b(t)$ denotes the cross-correlation between signals $a(t)$ and $b(t)$. Therefore, the matched filter is mathematically equivalent to the cross-correlation between the transmitted and received signals. When a copy of the transmitted pulse is present in the received signal, there will be a peak in the matched filter response at the corresponding delay. Noise and interference, on the other hand, will be filtered as long as they are not highly correlated to the transmitted pulse. An example matched filter output is shown in Fig. 6 for an LFM transmitted by an NI USRP-2901 in loopback mode so that $\tau = 0$. The waveform is sampled at $50\,\mathrm{MS/s}$ with bandwidth $B = 25\,\mathrm{MHz}$ and pulse width $\tau_p = 50\,\mu\mathrm{s}$. Before applying the matched filter, the signal is hardly visible above the noise (Fig. 6a). However, applying the matched filter to the received pulse produces a distinctive peak more than $30\,\mathrm{dB}$ above the noise floor (Fig. 6b). Although there is no delay, the peak exhibits a time shift equal to the duration of the waveform. The width of the peak, known as the range resolution, is primarily a function of the waveform bandwidth and determines the ability of the radar to distinguish targets that are similar in range.



(a) Before matched filtering



(b) After Matched filtering

Figure 6. Time-domain data from an LFM pulse in loopback

Once the matched filter has been used to determine the range of a target, the target's Doppler shift can be extracted using a technique known as pulse-Doppler processing. In `gr-plasma`, this algorithm is implemented in the "Doppler Processing" block and works as follows. Consider a radar that transmits M pulses that are reflected from an object moving with constant radial velocity such that the received signal is Doppler shifted by F_D Hz. If the matched filter output for a single pulse is $y_p(t)$, the matched filter output for all M pulses is given by

$$y(t) = \alpha \sum_{m=0}^{M-1} \exp(j2\pi F_D t) y_p(t - mT_{PRI} - \tau_m) \quad (5)$$

where τ_m is the target delay at each pulse, and α is a complex scale factor that encapsulates global amplitude and phase factors that do not vary significantly over the processing interval. Next, the signal is sampled in both fast-time (analog-to-digital converter (ADC) samples) and slow-time (pulses). Defining T_s as the sampling interval of the ADC and k as the sample index, Eq. 5 can be re-written as

$$y[k,m] = \alpha \exp(j2\pi F_D m T_{PRI}) y_p(kT_s - \tau_m) \quad (6)$$

which is a matrix where each column is the matched filter output for each pulse. For a given Doppler shift, a peak can be produced at the target range by coherently integrating the signal in slow-time so that the pulse-to-pulse phase rotation is eliminated. In practice, the Doppler shift is unknown a priori and the coherent integration must be performed for a Doppler filter bank, where each filter applies a phase correction for a different Doppler shift. Thus, for each possible Doppler $\hat{F_D}$, the Doppler processing block computes

$$Y[k, \hat{F}_D] = \sum_{m=0}^{M-1} y[k,m] \exp(-j2\pi \hat{F}_D m T_{PRI}) \quad (7)$$

which is equivalent to the discrete Fourier transform (DFT) of each row in y. Each element in the resulting matrix is a range-Doppler parameter pair, and a statistical hypothesis test can be applied to each element to determine if an object is present. Fig. 7 shows the range-Doppler response for an NI USRP-2901 in loopback, which produces a "target" at zero delay and Doppler. `gr-plasma` also includes a Range Doppler Sink block for displaying these maps in real time.

The signal processing and waveform generation blocks discussed above are implemented using ArrayFire (Yalamanchili et al., 2015), and the user can choose to perform the computations using CUDA, OpenCL, or CPU backends.

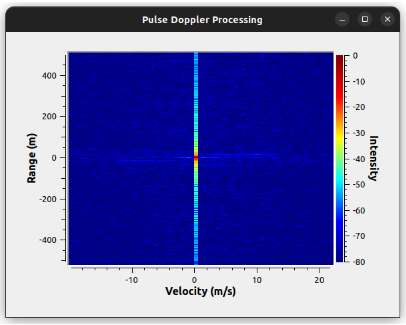


Figure 7. Range-Doppler map for a loopback configuration

On a host system with an AMD Ryzen 5800x CPU and Nvidia RTX 3070 GPU, range-Doppler processing could be performed in real time at sample rates up to $100\,\mathrm{MS/s}$ on the CUDA and OpenCL backends. If the message queue of a signal processing block has overflowed above a user-specified size (i.e., the host PC cannot achieve the required throughput for a given rate), messages are dropped until there is again space in the queue. Since data is processed on a CPI-by-CPI basis, this ensures that the processed data remains coherent even when samples must be dropped.

2.4. Miscellaneous Blocks

In addition to the waveform generation and signal processing blocks described above, `gr-plasma` also contains blocks for managing PDUs. As mentioned in Section 2.2, the USRP radar block outputs one PDU per pulse. However, many radar signal processing tasks operate on a number of pulses combined to form a coherent processing interval (CPI). For example, the DFT operation in Doppler processing assumes coherence over many pulses. Therefore, the "Pulses to CPI" block can be used to consolidate N_{CPI} PDUs containing individual pulses into one large PDU. This block also propagates any metadata in the input directly to the output for use by downstream processing. A custom PDU file sink block was also designed to facilitate saving the collected data for future use without first converting back to a traditional stream. The new file sink block can also (optionally) save metadata to a separate file by converting PDU metadata dictionaries to JSON. This makes it simple to save metadata that complies to the SigMF standard (Hilburn et al., 2018), and it is flexible enough to easily extend to other formats. A sample of the JSON metadata output that is generated automatically from the blocks is shown in Listing 2. In future work, this non-standard format will be replaced with a formal SigMF extension for processing radar data.

```
{
  ”annotations”: [
    {
      ”core:label”: ”pcfm”,
      ”core:sample_start”: 0,
      ”radar:doppler_fft_size”: 1024,
      ”radar:duration”: 5.12e−05,
      ”radar:num_phase_code_chips”: 128,
      ”radar:num_pulse_cpi”: 1024,
      ”radar:phase_code_class”: ”p4”,
      ”radar:prf”: 10000.0
    }
  ],
  ”captures”: [
    {
      ”core:frequency”: 5000000000.0,
      ”core:sample_start”: 0
    }
  ],
  ”global”: {
    ”core:datatype”: ”cf64_le”,
    ”core:sample_rate”: 20000000.0,
    ”core:version”: ”1.0.0”
  }
}
```

Listing 2. PDU file sink metadata output

3. Experiment Design

To verify the functionality of the module, an open-air experimental test bed was created using only commercial-off-the-shelf (COTS) components. In the experiment, an NI USRP-2901 was connected to 25 dBi parabolic dish antennas in a simultaneous transmit and recieve (STAR) configuration (Fig. 8) using only the internal amplification in the radio. The host PC was a Dell XPS 13 9310 laptop with an Intel i7-1165G7 CPU, 32 GB of RAM, and integrated graphics. This laptop was used for the experiment to demonstrate the utility of `gr-plasma` for performing radar tasks with low-cost, general-purpose systems. Since it does not have a dedicated GPU, it is unable to perform pulse-Doppler processing in real time. Instead, it simply performs range-Doppler processing as quickly as it can and drops any excess PDUs it receives while data is being processed.

The signal processing chain and the parameter specifications used in the experiment are as shown in the flowgraph in Fig. 9. Here, an LFM waveform is transmitted and received at the constant PRF. When a full CPI has been collected, matched filtering and Doppler processing are performed and the resulting range-Doppler map is plotted at the fastest rate that the host PC can sustain. At the same time, raw I/Q data from each pulse is saved to a file (along with its associated metadata) for further offline processing. This process continues until the flowgraph is manually stopped. Relevant radar processing parameters are summarized in Table 1. Note that the USRP source and sink blocks in the main tree are unable to support this sample rate for B210 devices due to overhead from the scheduler, resulting in fatal underflows and late commands.

Fig. 10 outlines the geography of the area used for the test. The target is a vehicle that accelerates in the direction shown until it reaches the speed limit of 25 mph. When it reaches the end of the street, it turns around and accelerates

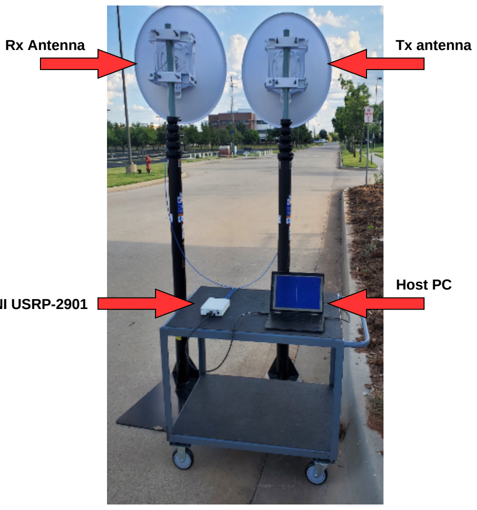
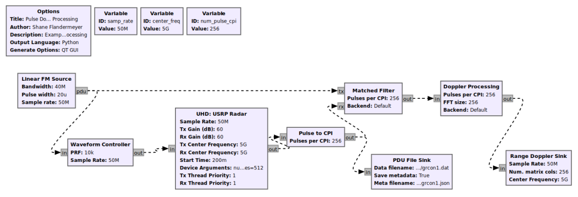


Figure 8. Experimental test bed setup



```
Options
Title: Pulse Do... Processing
Author: Shane Flandermeyer
Description: Examp...ocessing
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 50M

Variable
ID: center_freq
Value: 5G

Variable
ID: num_pulse_cpi
Value: 256

Linear FM Source
Bandwidth: 40M
Pulse width: 20u
Sample rate: 50M

Waveform Controller
PRF: 10k
Sample Rate: 50M

UHD: USRP Radar
Sample Rate: 50M
Tx Gain (dB): 60
Rx Gain (dB): 60
Tx Center Frequency: 5G
Rx Center Frequency: 5G
Start Time: 200m
Device Arguments: nu...es=512
Tx Thread Priority: 1
Rx Thread Priority: 1

Pulse to CPI
Pulses per CPI: 256

Matched Filter
Pulses per CPI: 256
Backend: Default

Doppler Processing
Pulses per CPI: 256
FFT size: 256
Backend: Default

PDU File Sink
Data filename: .../grcon1.dat
Save metadata: True
Meta filename: ...grcon1.json

Range Doppler Sink
Sample Rate: 50M
Num. matrix cols: 256
Center Frequency: 5G
```

Figure 9. Pulse-Doppler processing experiment

Parameter	Value
Sample rate	50 MS/s
Center frequency	5 GHz
Bandwidth	40 MHz
Pulse width	20 μs
PRF	10 kHz
Range resolution	3.75 m
Pulses per CPI	256

Table 1. Radar Operational Parameters

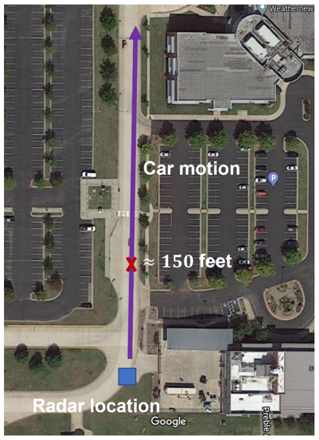


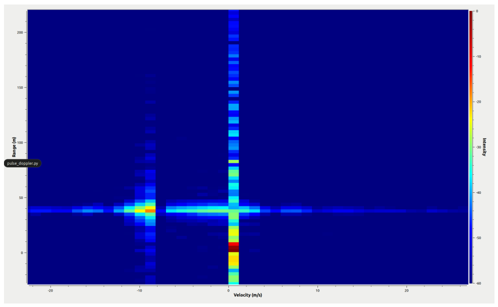
Figure 10. Data collection geometry

again toward the radar before decelerating to a stop. This test setup is desirable for a number of reasons. First, a short collection range is needed since the output power is limited by the internal amplification in the SDR. Moreover, a short collection range can be used since the radar has no blind range in the STAR configuration. Although the vehicle is not instrumented with GPS, having a known speed limit also makes it easy to validate the range-Doppler output as the car accelerates and decelerates.

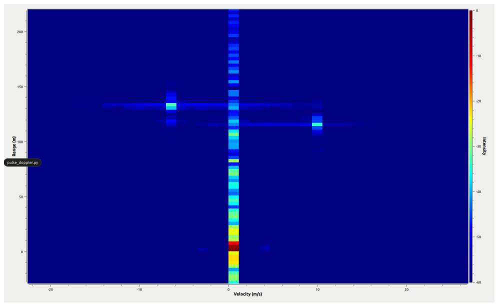
4. Experiment Results

Fig. 11 shows the range-Doppler maps that were produced from the Range Doppler Sink block during the experiment, zoomed in to the range and Doppler values that were relevant for the given geometry. No windowing was performed in range or Doppler to maximize SNR and resolution at the expense of higher sidelobes. In Fig. 11a, the vehicle is accelerating away from the radar. There is a pronounced clutter ridge near zero Doppler from stationary scatterers in the scene (e.g., buildings, trees, and the ground), and the vehicle can be clearly seen at a range of $50\,\mathrm{m}$ as it moves away from the radar at roughly $10\,\mathrm{m/s}$ (with a negative Doppler shift).

In the range-Doppler map in Fig. 11b, multiple vehicles are traveling along the road. In this figure, the car from Fig. 11a is moving towards the radar and can be seen with range and velocity $(R, v) = (115\,\mathrm{m}, 10\,\mathrm{m/s})$. At the same time, a large truck and a small car can be seen driving away from the radar (on the left). The large truck is located at $(135\,\mathrm{m}, -6\,\mathrm{m/s})$, and is the brightest scatterer in the scene besides the clutter. The small car is following the truck at the same speed, but at $120\,\mathrm{m}$ from the radar. The car's range-Doppler response has a much lower amplitude than the truck's since the truck has a much larger radar cross section (RCS) than the car.



(a) Vehicle moving away from radar



(b) Vehicle moving toward radar

Figure 11. Range-Doppler maps

5. Conclusion and Future Work

In this paper, the `gr-plasma` module has been presented as a new tool for generating and processing radar data from within GNU Radio. Every block in the module uses the message passing architecture to process data as PDUs, which better captures the bursty nature of radar processing while improving performance by avoiding overhead from the scheduler. The module currently implements blocks that generate LFM and PCFM waveforms, collect data from USRP SDRs, and perform pulse-Doppler processing on both the CPU and GPU. In future work, `gr-plasma` will be extended to include a more diverse collection of waveform generators and more signal processing blocks (e.g., for constant false alarm rate (CFAR) detection). Adaptive capabilities that utilize the results of downstream processing are also planned for a future release.

References

Blunt, Shannon D., Cook, Matthew, Jakabosky, John, De Graaf, Jean, and Perrins, Erik. Polyphase-coded fm (pcfm) radar waveforms, part i: implementation. *IEEE Transactions on Aerospace and Electronic Systems*, 50(3):2218–2229, 2014a. doi: 10.1109/TAES.2014.130361.

Blunt, Shannon D., Jakabosky, John, Cook, Matthew, Stiles, James, Seguin, Sarah, and Mokole, E. L.

Polyphase-coded fm (pcfm) radar waveforms, part ii: optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 50(3):2230–2241, 2014b. doi: 10.1109/TAES.2014.130362.

Christiansen, Jonas Myhre and Smith, Graeme E. Development and calibration of a low-cost radar testbed based on the universal software radio peripheral. *IEEE Aerospace and Electronic Systems Magazine*, 34(12):50–60, 2019. doi: 10.1109/MAES.2019.2953803.

Hilburn, Ben, West, Nathan, O'Shea, Tim, and Roy, Tamoghna. Sigmf: The signal metadata format. *Proceedings of the GNU Radio Conference*, 3(1), 2018. URL `https://pubs.gnuradio.org/index.php/grcon/article/view/52`.

Levanon, Nadav and Mozeson, Eli. *Radar Signals*. John Wiley Sons, Ltd, 2004. ISBN 9780471663089. doi: https://doi.org/10.1002/0471663085.ch6. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/0471663085.ch6`.

Richards, M.A. *Fundamentals of Radar Signal Processing, Second Edition*. McGraw-Hill Education, 2013. ISBN 9780071798334. URL `https://books.google.com/books?id=zQlVAgAAQBAJ`.

Tan, Peng Seng, Jakabosky, John, Stiles, James M., and Blunt, Shannon D. On higher-order representations of polyphase-coded fm radar waveforms. In *2015 IEEE Radar Conference (RadarCon)*, pp. 0467–0472, 2015. doi: 10.1109/RADAR.2015.7131044.

Wunsch, Stefan. gr-radar. `https://github.com/kit-cel/gr-radar`, 2014.

Yalamanchili, Pavan, Arshad, Umar, Mohammed, Zakiuddin, Garigipati, Pradeep, Entschev, Peter, Kloppenborg, Brian, Malcolm, James, and Melonakos, John. ArrayFire - A high performance software library for parallel computing with an easy-to-use API, 2015. URL `https://github.com/arrayfire/arrayfire`.