Project 15

Developing Two-Transmit Channel SDR with Beamforming Capabilities

Peyton D. Adkins, Gabriel Faro, Alexandra Calvo, Joel Brigida, Jason Kurth

November 18th, 2023

• • • Algorithmic Research for DOA and Beamforming • • •

References

1. [[PySDR: DOA and Beamforming](https://pysdr.org/content/doa.html)]{https://pysdr.org/content/doa.html}
2. [[Jon Kraft: Building a Beamformer](https://www.youtube.com/watch?v=2QXKuEYR4Bw)] {https://www.youtube.com/watch?v=2QXKuEYR4Bw}
3. [[Jon Kraft: Enable Dual Rx Dual Tx](https://www.youtube.com/watch?v=ph0Kv4SgSuI)] {https://www.youtube.com/watch?v=ph0Kv4SgSuI}
4. [[Jon Kraft: Testing Dual Rx Dual Tx](https://github.com/jonkraft/PlutoSDR_Labs/blob/master/Pluto_revC_rev2.py)] {https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py}
5. [[Analog Devices pyadi-iio](https://analogdevicesinc.github.io/pyadi-iio/)] {https://analogdevicesinc.github.io/pyadi-iio/}
6. [[Analog Devices libiio](https://analogdevicesinc.github.io/libiio/v0.23/python/index.html)] {https://analogdevicesinc.github.io/libiio/v0.23/python/index.html}
7. [[NumPy FFT](https://numpy.org/doc/stable/reference/routines.fft.html)] {https://numpy.org/doc/stable/reference/routines.fft.html}
8. [[dBFS Calculation](https://support.xilinx.com/s/question/0D52E00006hpWJuSAM/dbfs-calculation?language=en_US)] {https://support.xilinx.com/s/question/0D52E00006hpWJuSAM/dbfs-calculation?language=en\_US}
9. [[NumPy Correlate](https://numpy.org/doc/stable/reference/generated/numpy.correlate.html)] {https://numpy.org/doc/stable/reference/generated/numpy.correlate.html}
10. [[Theta Phase Delay Calculation](https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/rsn2.12352)] {https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/rsn2.12352}
11. [[Jon Kraft: Monopulse Tracking](https://www.youtube.com/watch?v=XP8OWMDHfOQ)] {https://www.youtube.com/watch?v=XP8OWMDHfOQ}
12. [[Quadrature](https://www.tek.com/en/blog/quadrature-iq-signals-explained)] {https://www.tek.com/en/blog/quadrature-iq-signals-explained}

Jon Kraft: Build Your Own Phased Array Beamformer

**SECT. I**

**Overview of Beamforming Basics**

A screen shot of a diagram

Description automatically generated

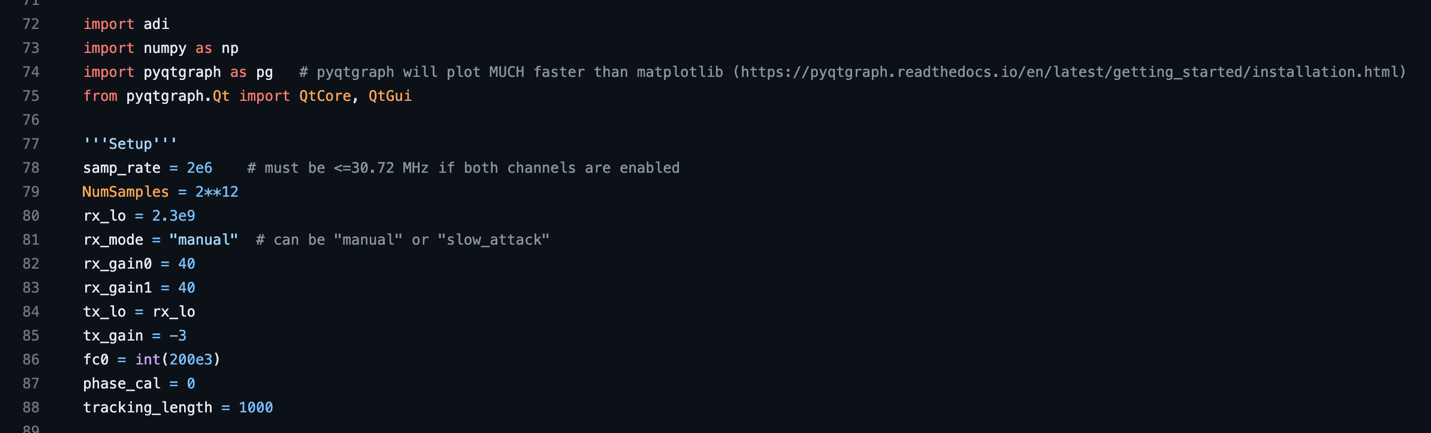
The delay between each element may be presented in one of three ways:

1. An incremental distance to travel
   1. This is to describe the delay as an incremental distance that the wavefront must travel to strike the adjacent element, the incremental distance being referred to as “L”
2. A time delay between elements
   1. This is to describe the delay as a time delay between elements, which is the distance “L” divided by the speed of light.
3. A phase shift between elements
   1. This describes the delay as a phase shift between each element; it is the time delay at a single frequency. These are implemented more often as they are perceived as “easier.”

**SECT. II**

**Configuring connectivity to the Pluto SDR using pyadi-iio library**

The following code contributes to the enabling of dual receive and dual transmit on Pluto SDR by Jon Kraft (*Kraft 4*). Physical modification of the Pluto is required (*Kraft 3*).



https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_MonopulseTracking\_youtube.py – Python lines 72-88

*Line 72*

The pyadi-iio library is “a python abstraction module for ADI hardware with IIO drivers to make them easier to use” (*Analog 5*). We configure the Pluto SDR to see itself as an ad9361, and using the library, create an object called “sdr” set to the Pluto’s URL.

*Lines 78-79*

The sample rate must be less than or equal to 30.72 MHz when both channels are enabled because the Pluto still uses a cmos interface. The sample size dictates the size of each buffer.

*Lines 80-84*

The “rx\_lo” variable is carrier frequency of the Rx path, which is set to 1GHz, later used to also set the “tx\_lo” variable. For “rx\_mode,” this can either be set to “manual” or “slow\_attack.” In “slow\_attack,” the variables “rx\_gain0” and “rx\_gain1” are ignored, whereas they would be used in “manual” mode.

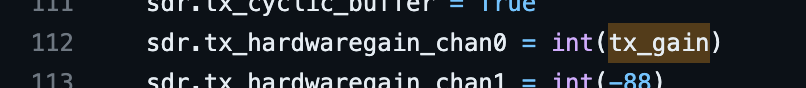
*Line 85*

The “tx\_gain” variable is later assigned to the sdr. tx\_hardwaregain\_chan0 attribute. This attribute is only applicable when the gain\_control\_mode attribute is set to “manual” as is remarked as the gain attenuation:

A black background with white text

Description automatically generated

https://analogdevicesinc.github.io/pyadi-iio/devices/adi.ad936x.html#adi.ad936x.ad9361



https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_MonopulseTracking\_youtube.py – Python Line 112

*Line 86*

The sdr.rx\_rf\_bandwidth and sdr.tx\_rf\_bandwidth property is imposed onto the DAC with a metric of 200KHz. This is assigned initially to the variable “fc0” for later use:

A black background with white text

Description automatically generated

**A black and white text

Description automatically generated**

https://analogdevicesinc.github.io/pyadi-iio/devices/adi.ad936x.html#adi.ad936x.ad9361

*Line 87*

The “phase\_cal” variable is the manually configured phase shift calibration adjustment set to later subtract out the phase errors between Rx1 and Rx2. This calibration may be adjusted and accounts for cable differences and other discrepancies between elements.

*Line 88*

The “tracking\_length” is a metric for the range of the monopulse tracking capabilities, used when defining the Y-axis of the plot window later in line 211:

A computer screen with text

Description automatically generated

https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_MonopulseTracking\_youtube.py – Python Line 211

The following code configures the distance between the Rx antennas, assuming the physical distance between them is 65mm (about 2.56 in), which is half wavelength spacing for a 2.3GHz frequency.

A screenshot of a computer code

Description automatically generated

https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_MonopulseTracking\_youtube.py – Python lines 90-94

*Lines 91-93*

As mentioned above, the current configuration for the monopulse tracker is established such that *d* the distance between each element is a fraction of the wavelength at 2.3GHz, which can be written as:

The variable “d” in line 93 is set to this calculation.

The following code takes the variables established in all prior lines to establish connectivity and configuration with the PlutoSDR device.

A computer screen shot of a program

Description automatically generatedhttps://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_MonopulseTracking\_youtube.py – Python Lines 96-114

*Line 97*

The “sdr” variable is set to an object from the pyadi-iio library under the ad9361 attribute. Recall that the Pluto SDR has had a firmware reconfiguration to have it see itself as this device, allowing use to utilize this library.

*Lines 100-114 (note for line 108)*

Each of the appropriate “sdr” object attributes is set. Each of these attributes can be found in the pyadi-iio library documentation [here](https://analogdevicesinc.github.io/pyadi-iio/devices/adi.ad936x.html#adi.ad936x.ad9361). A point of note lies in line 108 where we access the libiio python API directly (the library pyadi-iio is built on top of). We access the set\_kernel\_buffers\_count attribute and set this to 1. The default configuration of buffer kernels is 4 but is set to 1 to avoid staling the data used in the Pluto (*Analog 6*).

The following code creates and transmits an iQ vector.

**A computer screen with text and numbers

Description automatically generated**

https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_MonopulseTracking\_youtube.py – Python Lines 116-124

*Lines 117-120*

The “N” constant is set to 65536, which is the number of samples according to the internal Pluto ADC. It is what allows the definition of the number of samples per period in the wave.

The “ts” variable is set to a rate of the number of samples with respect to time (per seconds). It serves to define the step of the continuous wave, which is stored in line 120.

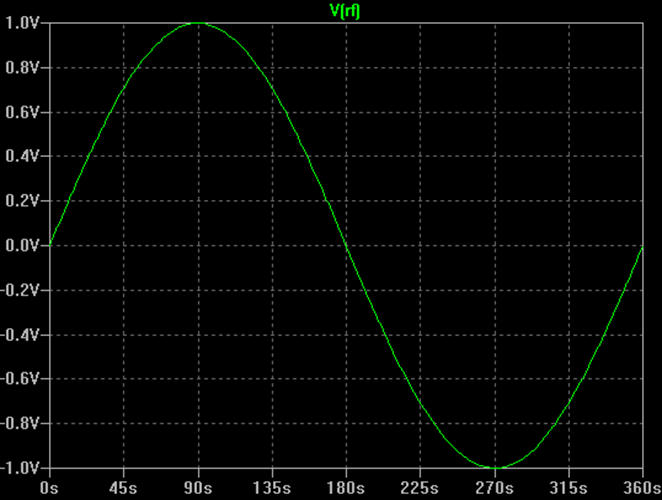
*Lines 121-123*

Using numpy.cos() and numpy.sin(), arrays of trigonometric sines/cosines for the NumPy array “t” are created. These lines perform the modulation for the wave done via quadrature. The arguments must be converted to radians for the computation to be sound. Using the complex number written as “1j”, the cosine and sine arrays are joined to create the iQ array.

A white text with black text

Description automatically generated with medium confidence

https://www.tek.com/en/blog/quadrature-iq-signals-explained



https://www.tek.com/en/blog/quadrature-iq-signals-explained

*Line 124*

The iQ array is sent as a vector.

The following code utilizes the NumPy fft library for frequency domain manipulation and shifting. These frequency bins are then assigned and given focus by defining the start and end of the signal through the step up and step down of the bandwidths set by “fc0”.

A screen shot of a computer code

Description automatically generatedhttps://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_MonopulseTracking\_youtube.py – Python Lines 126-130

**SECT. III**

**Exploring Jon Kraft’s Monopulse Tracker**

The following code is derived from Jon Kraft’s GitHub repository under the Pluto\_beamformer\_MonopulseTracking\_youtube.py file (Kraft 2). This is a continuation of the previous research, looking into the calculations behind monopulse tracking.

A screen shot of a computer program

Description automatically generatedhttps://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 207-223

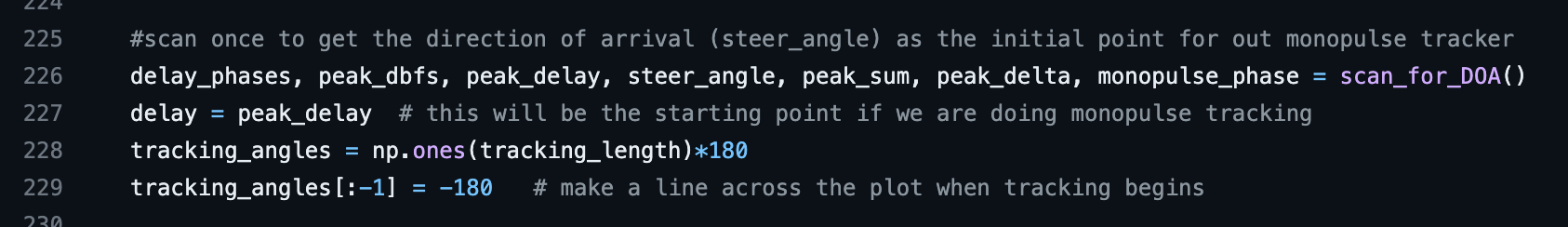
*Lines 208-218*

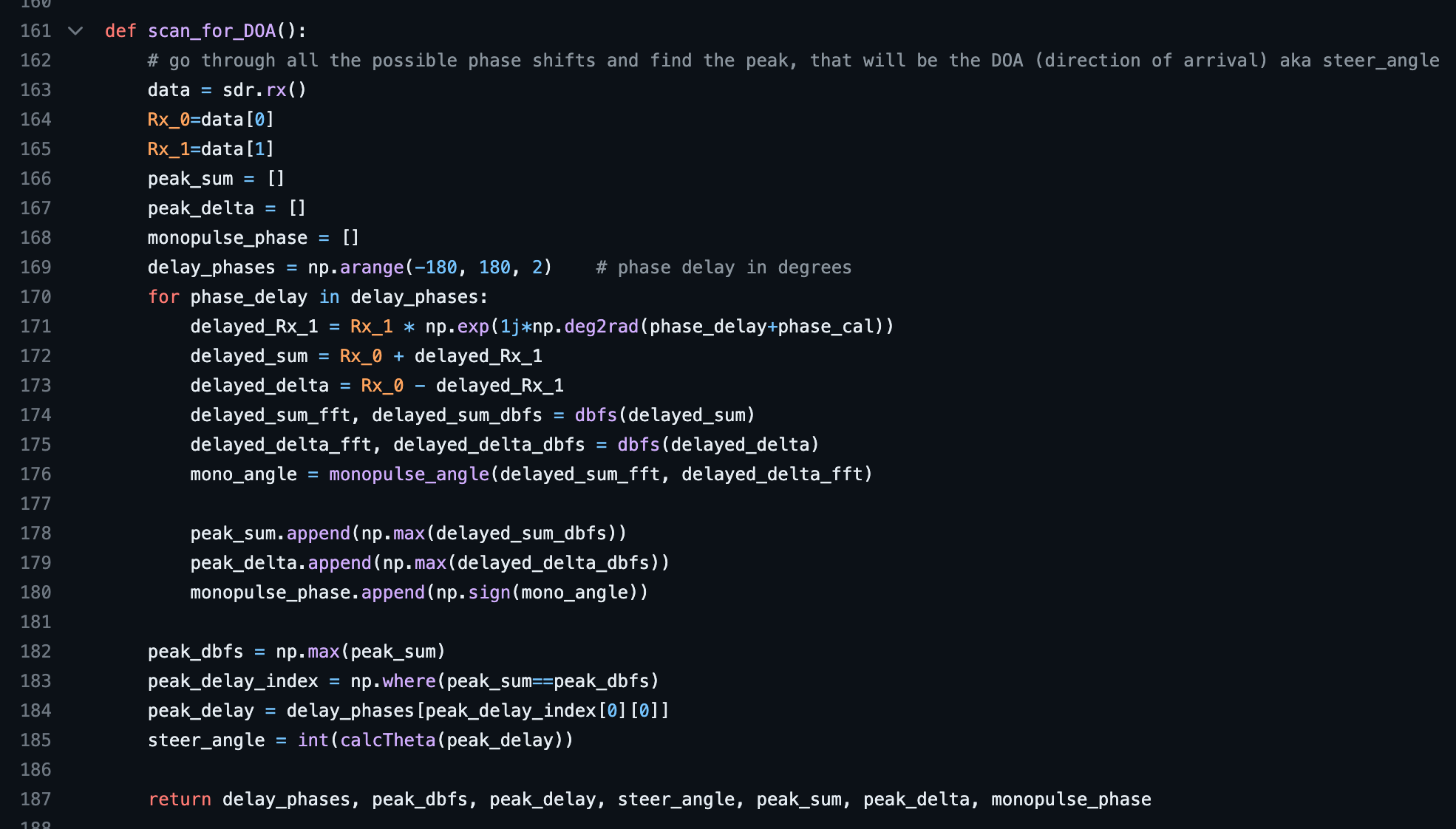
The pyqtgraph library is used to create a graphical interface for running various metrics used for testing. This is noted to perform much quicker than matplotlib.

*Lines 208-218*

These lines allow the Pluto time to work through calibrations.

The following code snippet serves as the main computation for getting the steering angle (direction of arrival DOA). This serves as the initial point for the monopulse tracker. The variables in line 226 are derived from the function scan\_for\_DOA(), which will be covered in detail next:

https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 226-229

https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 161-187

*Lines 163-169*

Data is recollected by the sdr.rx() function and stored in the “data” array variable, which is then split into the first and second receiver in lines 164 and 165, respectively. The variables in lines 166 through 168 are arrays set to be assigned with the maximum respected values of each iteration of possible phase delays in degrees, which is assigned in line 169.

*Lines 170-180*

This loop iterates through each possible phase delay through degrees -180º through 180º. Using the first Rx for calculation, each point in the “Rx\_1” array is multiplied by a complex number, sourced from taking Euler’s number to the power of the current phase delay (adjusted by the established phase calibration) multiplied by the complex number “1j”. This effectively shifts each point in the “Rx\_1” array by the current phase delay (in degrees) set by the loop.

The newly shifted data stored in the variable “delayed\_Rx\_1” is then summated and subtracted to the “Rx\_0 data” to find the delayed sum and delayed delta, respectively. These two new variables “delayed\_sum” and “delayed\_delta” in lines 172 and 173 are used to find the converted fast Fourier transform (FFT) scaled in decibels relative to full scale (dBFS). More information about this process may be found in the breakdown of the dBFS function.

In line 176, the monopulse angle of the iterated phase delay is calculated by the monopulse\_angle() function, discussed in more detail later in the paper.

*NOTE:*

The purpose of extracting the sum and delta curves is to be able to find the *phase difference* which is denoted as the variable “mono\_angle.”

* The difference between the SUM and DELTA will tell if the beam is off target.
* The phase difference (in radians) will tell which way to redirect the beam.
  + If difference is positive, decrease the phase shift on Rx1
  + If difference is negative, increase the phase shift on Rx1

A diagram of a graph

Description automatically generated

https://www.youtube.com/watch?v=XP8OWMDHfOQ

*Lines 182-185*

Using NumPy, the max value of the delayed sums (in the “peak\_sum” array) and its index are found. The delay phase for which this value was found in the before loop is found by indexing the “delay\_phases” variable containing the phase delays by where the peak sum was found. The steer angle is then calculated using the calcTheta() function, discussed later in the paper.

*NOTE:*

This method for finding the DOA is one of a few possible methods. The current method relies on finding the maximum of the SUM curve, though this could also be found using the minimum of the DELTA curve. Or alternatively, finding where the phase difference shifts between negative and positive can also lead to the same result.

The dbfs() function is used in the algorithm to convert the iQ samples from former calculations into fast Fourier transform (FFT) scaled in decibels relative to full scale (dBFS). The purpose of this is to take the raw data which is in the time domain and fit it into the frequency domain. This function returns both a shifted array and the dBFS.

https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 140-148

*Lines 142-144*

Utilizing the Hamming Window, the raw data used as the parameter in the function is adjusted to reduce the rippling, giving a better structure as to the original signal’s frequency spectrum.

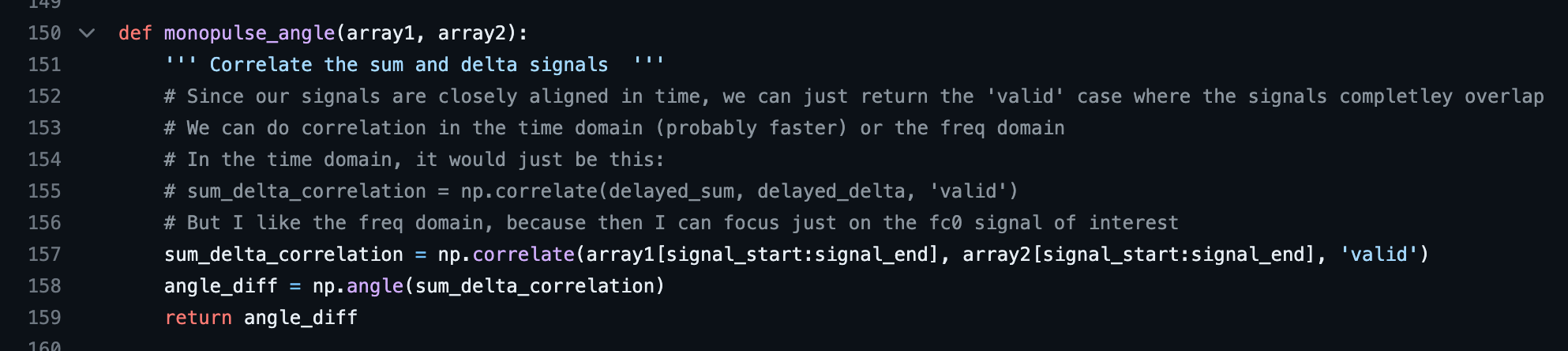
*Lines 145-146*

The adjusted data “y” is used to create an FFT shift distributed across the adjusted data (hence the division by the np.sum(win)). This is then centered by means of NumPy’s FFT functions to the “s\_ssft” variable in line 146 (*NumPy 7*).

*Line 147*

A conversion is necessary to scale the shifted data into dBFS. This can be achieved through the dBFS formula of 20\*log10(abs(A)/B) where A is the sample data (in our case the shifted data) and B in our case is 2^11 to convert to dBFS, as Pluto is a signed 12-bit ADC (*dBFS 8*).

The monopulse\_angle() function is used in the algorithm to correlate the sum and delta signals. Jon Kraft has the algorithm work to the frequency domain but notes how the algorithm may be adjusted to work in the time domain.

https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 150-159

*Line 157*

Set to work in the frequency domain, the algorithm indexes the parameter arrays set by the “signal\_start” and “signal\_end” variables in NumPy’s correlate function (*NumPy 9*). This result is returned to the “sum\_delta\_correlation” variable, which is a discrete cross-correlation of both indexed arrays. The mode is set to ‘valid’ to ensure that only the value of the correlation for the overlapping data is returned.

A diagram of a graph

Description automatically generated with medium confidence

https://en.wikipedia.org/wiki/Cross-correlation

*Line 158*

The angle difference is found by NumPy’s angle function, which is set to radians by default. This is the returned value. The value is later re-extracted and used in a sign operation to find the polarity in line 201. Again, the purpose of this phase difference polarity variable is to understand whether the phase shift should be increased or decreased.

* + If difference is positive, decrease the phase shift on Rx1
  + If difference is negative, increase the phase shift on Rx1

A diagram of a graph

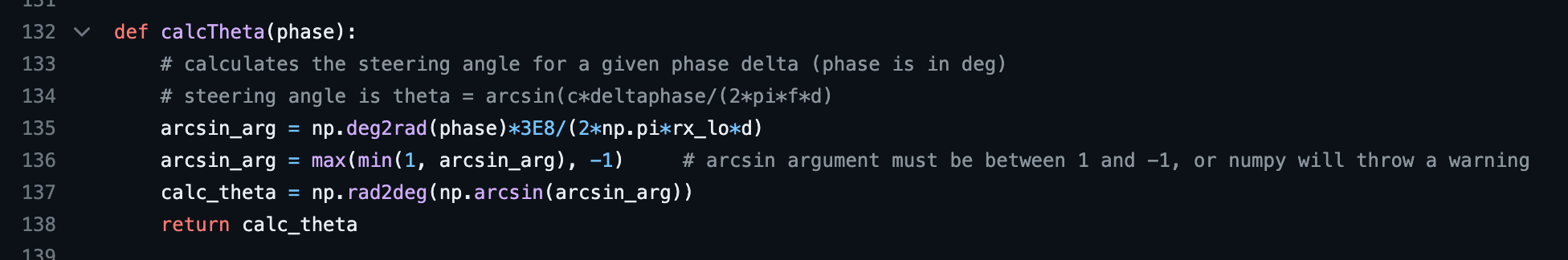
Description automatically generated

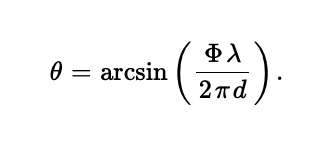
https://www.youtube.com/watch?v=XP8OWMDHfOQ

A computer screen shot of a program

Description automatically generatedhttps://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 199-205

The calcTheta() function is used in the algorithm to take a given phase delta (in degrees) and calculate the steering angle. The formula for Theta in this case is theta = arcsin(c\*deltaphase/(2\*pi\*f\*d). The phase shift is denoted as “c” or Φ (Greek letter Phi) in reference (*Theta 10*).

https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 132-138



https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/rsn2.12352

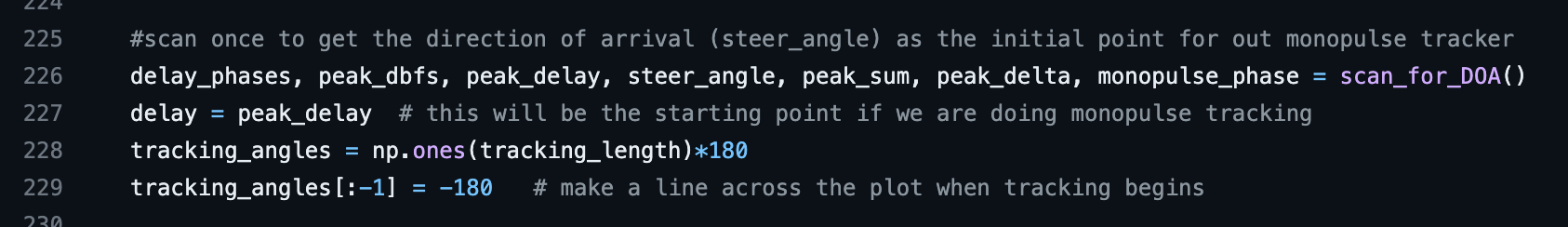
*Line 135*

With the phase delta parameter assumed to be given in degrees, the formula set for finding theta is found by first calculating the arcsine argument.

*Lines 136-137*

To properly allow the NumPy function “arcsine” to accept a parameter, the arcsine argument must be between the values -1 and 1. The result is calculated using the NumPy arcsine function, which is also converted back into degrees.

Recall the previous lines of code that calls each prior function discussed in the paper:

https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 226-229

*Line 226*

The function scan\_for\_DOA() is called, returning delay phases, the peak dBFS, peak delay, peak summation, peak delta, steering angle, and the phase difference (mono\_angle). This line and the rest of the snippet is outside the main timeout loop and serves to find the DOA as the initial point for the monopulse tracker.

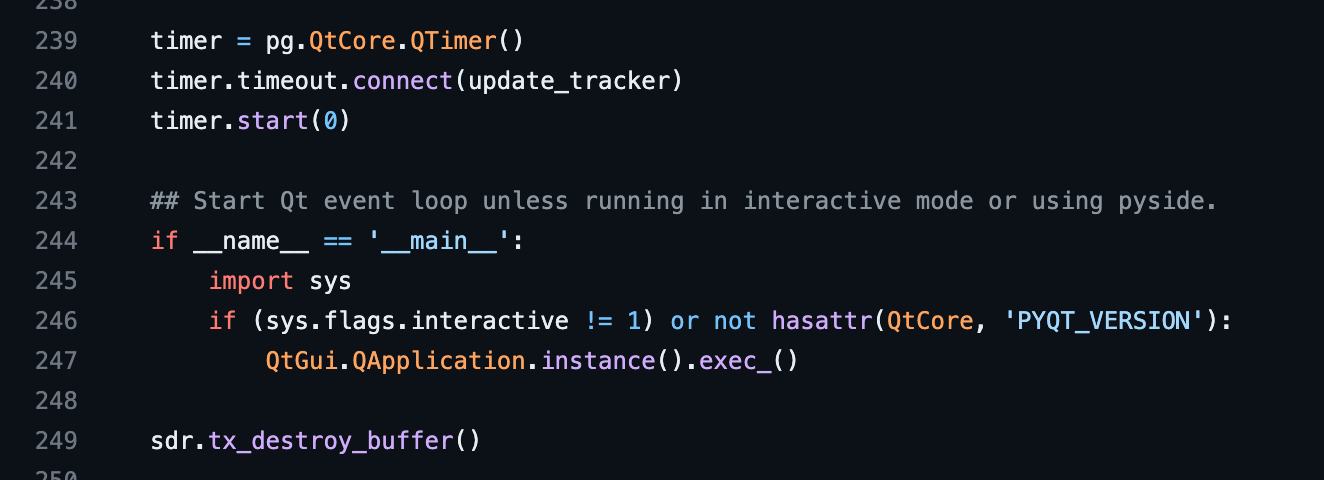
NOTE:

In the scan\_for\_DOA() function, each phase delay is iterated through, and the SUM and DELTA curves are found. The peak\_delay is another variable of interest as it is what serves to start the tracker. The re-iteration of the loop does not occur again, only once for the initial discovery of the DOA.

*Lines 228-229*

This array is created to be used in the plot graph. It generates a line across the plot whenever tracking begins.

The base loop for the tracking exists within this snippet as well as a systems-check to see if the program is running in “interactive” mode or whether Pyside is currently in use.

https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 239-249

*Lines 239-241*

Using the pyqtgraph.Qt library, a timer is instantiated, connected to the function update\_tracker(). This timer starts in line 241 and serves as the primary way the tracker continuously updates.

*Lines 244-247*

The system is checked for whether “interactive” mode is on or if Pyside is in use.

*Line 249*

The buffer for the sdr object is destroyed once all calculations are finished and the program is executed.

Attached to the timer object discussed prior, the update\_tracker() function services to be the main algorithm for the program. At the very end of the function, it also updates the curve of the GUI to account for any changes in position.

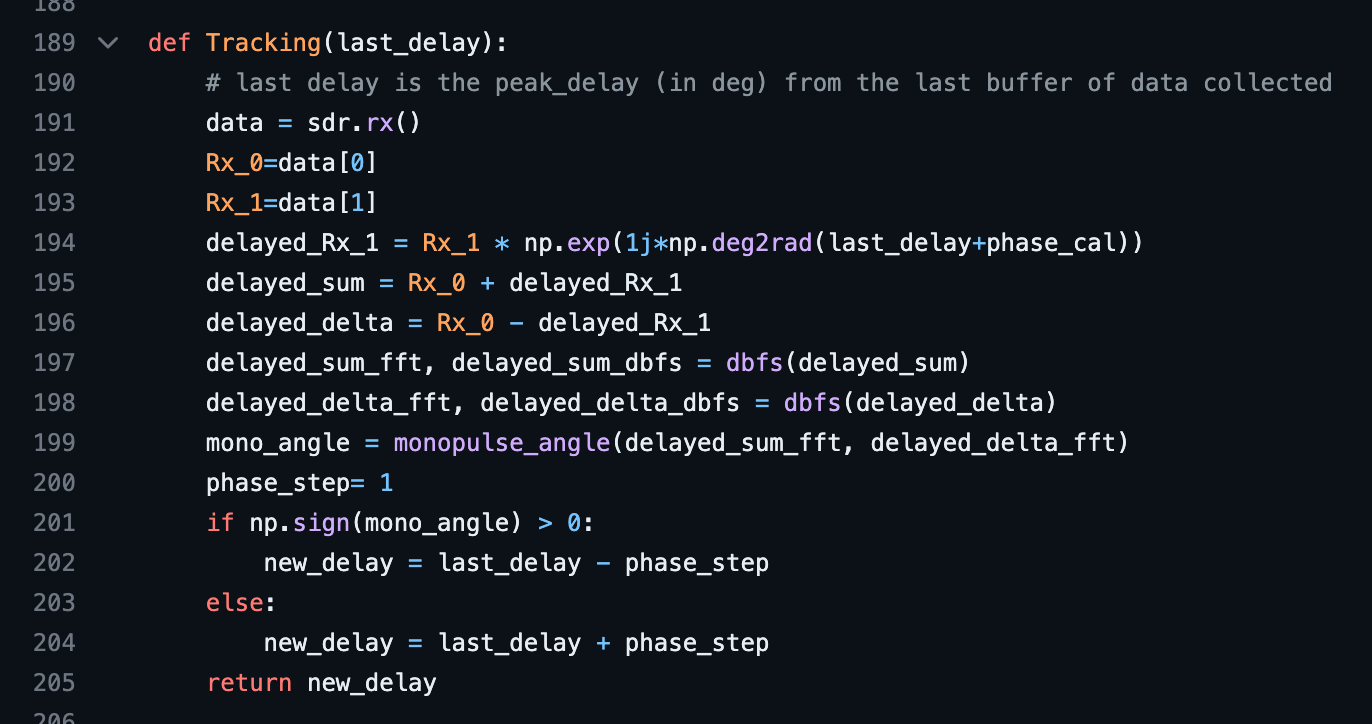
A screen shot of a computer code

Description automatically generatedhttps://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 232-237

*Lines 233-236*

The new delay is found through the Tracking() function which is then appended onto an array of tracking angles after being calculated for the Theta value through the calcTheta() function. These values are what are used to create the curve in the GUI.

The Tracking() function is like the scan\_for\_DOA() function, only this time we are given the last delay found by this function earlier in the algorithm. This function updates the new delay and new steering angle according to the continuous algorithmic loop.

https://github.com/jonkraft/PlutoSDR\_Labs/blob/master/Pluto\_revC\_rev2.py – Python lines 189-205

*Lines 191-200*

The code is conducted similarly as the scan\_for\_DOA() function. The “phase\_step” variable is used to control the rate for which the delay is adjusted according to the existence of a new delay.

*Lines 201-204*

The “mono\_angle” is converted into a variable of 1, 0, or -1 through NumPy’s sign function. Whether the mono angle is greater than or less than / equal to zero determines if the new delay needs to be adjusted by an increase or decrease in “phase\_step”.

* + If difference is positive, decrease the phase shift on Rx1
  + If difference is negative, increase the phase shift on Rx1

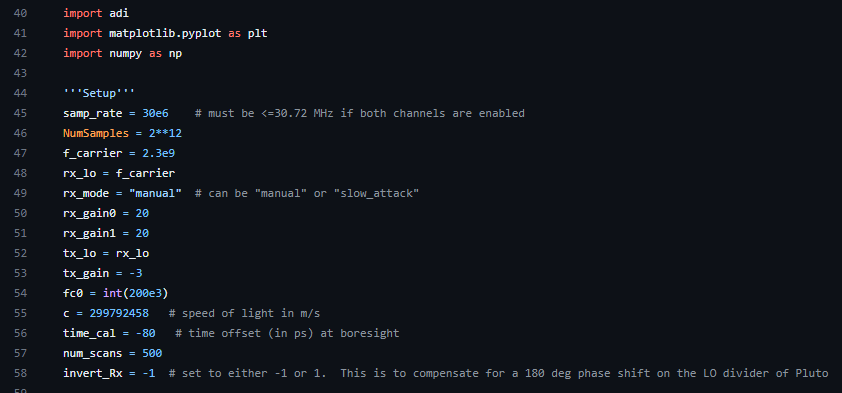
A computer screen shot of a computer screen

Description automatically generatedhttps://www.youtube.com/watch?v=XP8OWMDHfOQ

**SECT. IV**

**Exploring Jon Kraft’s Time Delay Option Code: Set Up**

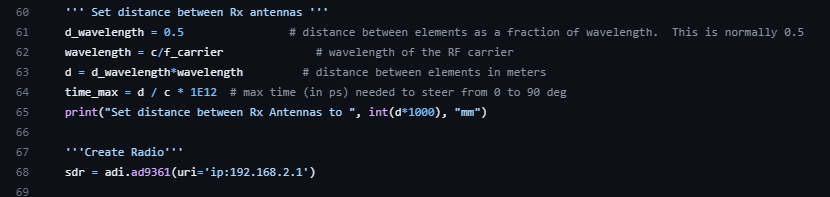
The figure below now begins to explore a new set of code that steers the phased array using a time delay trick as opposed to a phase shift. This gives the group more options when troubleshooting the project and allows us to have more freedom in choosing the code that works best for the group. This time delay is also set up for a 2.3GHz signal.

https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_TimeDelay\_2p3GHz\_youtube.py

*Lines 55-58*

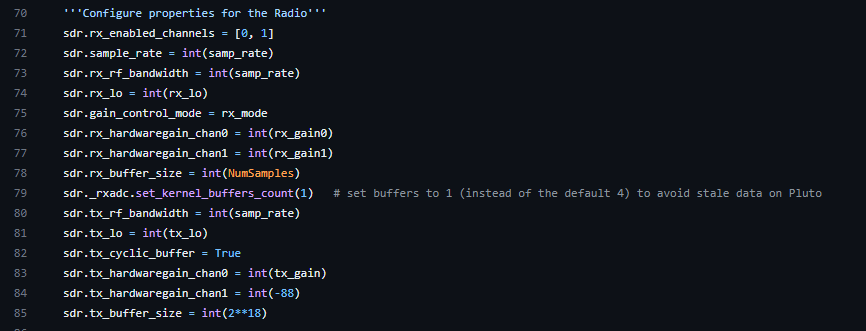
The set up is much of the same as from the monopulse tracking python code, with the biggest difference starting at lines 55 through 58. Here a new set of variables is created for the calculations that will be needed later on, including speed of light, a time offset, and a number of scans to be done.

Now the distance between the receiver antennas and the radio needs to be set up. The figure below shows the lines of code that were used to set up this distance and to create the radio.

https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_TimeDelay\_2p3GHz\_youtube.py

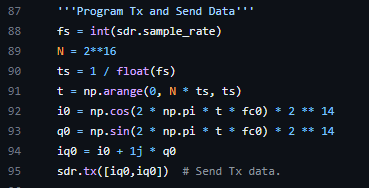
*Lines 61-68*

The lines above show a very similar set up to the phase shift monopulse tracking python code. Here there are some slight differences with the nomenclature and variable values, like with the wavelength using c and f\_carrier, which is just like the monopulse code, except that c uses a more specific number for the speed of light and the carrier signal is renamed to the f\_carrier variable. The major difference between this time-delay set up and the phase-delay set up is in the time\_max variable that sets the maximum time in picoseconds needed to steer from an angle of 0 to 90 degrees, using the distance between the antennas divided by the speed of light. Since the same SDR (Pluto SDR) is being utilized throughout this project, as well as in this reference code, it is the same ip address.

The next part of the code consists of configuring the properties for the Pluto SDR radio. The figure below shows all the configurations set up for the Pluto SDR. https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_TimeDelay\_2p3GHz\_youtube.py

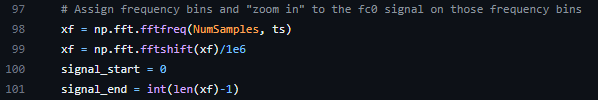
This is nearly identical to the configurations done in the monopulse tracking code document. It also uses the same sdr variable explained above. The only difference is that the bandwidth is set to the sample rate as opposed to fc0\*3 in the monopulse tracking code.

The next part, which involves programming the transmission signal Tx and the send data, is done exactly the same as the monopulse tracking code for lines 87-95, the figure below shows the reference code.



https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_TimeDelay\_2p3GHz\_youtube.py

However for the following code for assigning the frequency bins and "zooming in” to the fc0 signal, the code does have some changes when comparing to the monopulse tracking code.

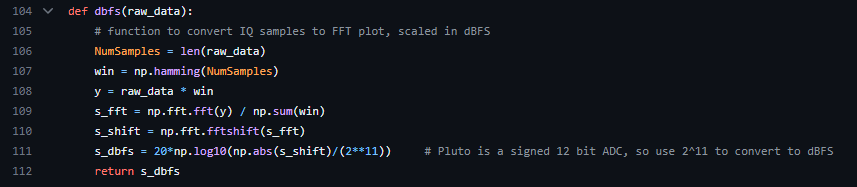


https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_TimeDelay\_2p3GHz\_youtube.py

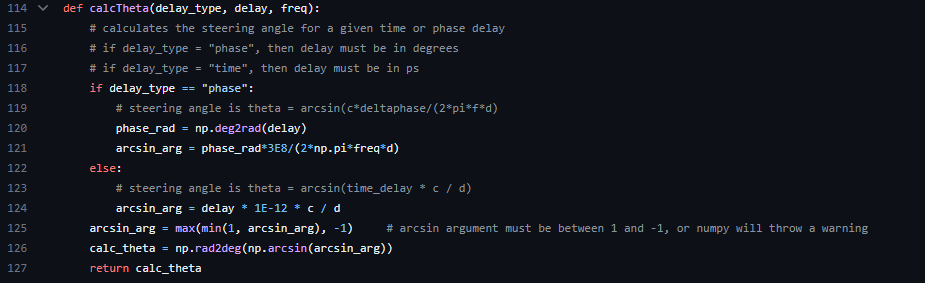
*Lines 97-101*

Since time is what is being used instead shift, the signal\_start is set to zero instead of the sample rate, and the signal\_end is set to the end of the length of the xf signal is.

The following figure shows the code for a definition that will take the raw\_data and convert it to FFT plot, which is scaled in decibels relative to full scale, thus returning a value that is in dBFS.

https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_TimeDelay\_2p3GHz\_youtube.py

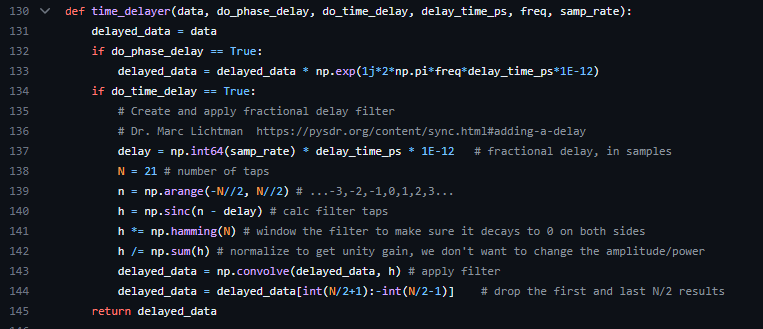
The next function is the calcTheta function, which once again is very similar to the explanation in the monopulse tracking code. Here, the calculation for the steering angle is done for a given time or phase delay. The figure below shows the difference.

https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_TimeDelay\_2p3GHz\_youtube.py

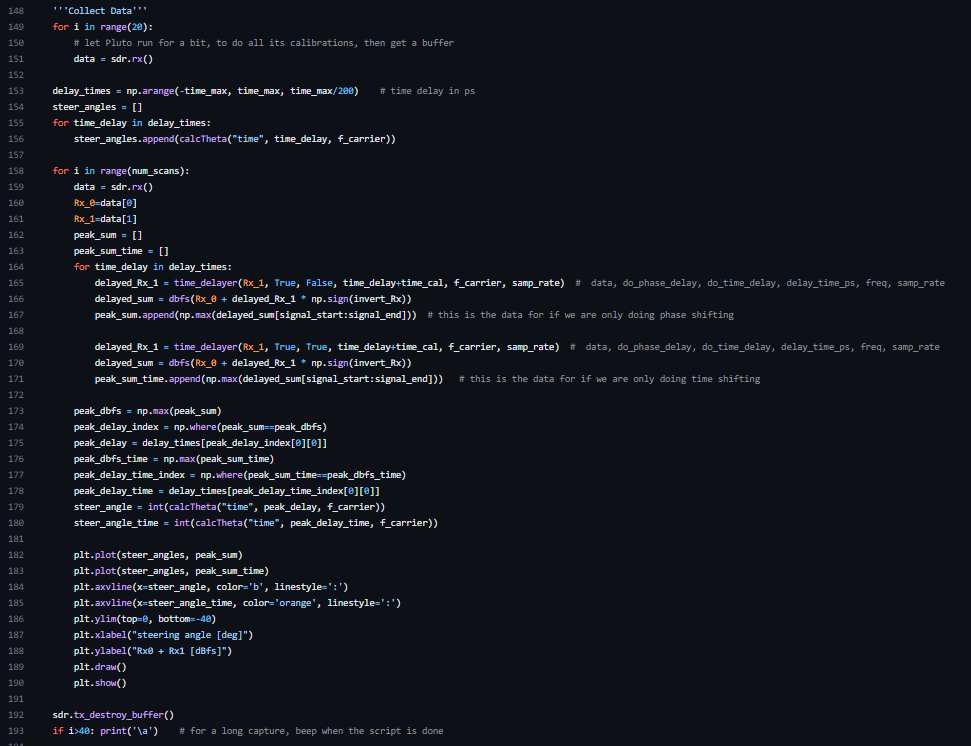
*Line 118*

This line shows the delay\_type variable addition to this version of the calcTheta function, where Jon Kraft gives the option of using either "phase” or “time”, where the main difference is that if time delay is chosen, then the delay must be in picoseconds. The calculations are still done in the phase domain, however it is set up so that one who desires to shift to a time delay option can do so.

The time\_delayer function will take the data collected from the receiver antennas and apply a delay filter. The figure below shows how the adjustment for the time delay of incoming data is done.

https://github.com/jonkraft/Pluto\_Beamformer/blob/main/Pluto\_beamformer\_TimeDelay\_2p3GHz\_youtube.py

This delay filter is cited by Jon Kraft as work done by Dr. Marc Lichtman <https://pysdr.org/content/sync.html#adding-a-delay>. The function then returns the delated\_data variable, which is what is utilized in the receive antennas to properly adjust and steer itself to obtain the highest gain from the incoming signal. For reference, the set of code responsible for collecting the data in the pluto is done as follows by Jon Kraft.

The collect data section of the code is where all the functions are mostly called and applied. Plots of the steering angle, and the receiver antenna results are also done here. The very last part of this code is to destroy the buffer of the Pluto SDR, making a beam when the script is done.

<https://www.digikey.ca/htmldatasheets/production/2073700/0/0/1/adalm-pluto-product-highlight.html>