Digital Communication using GNU Radio

Prof. Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

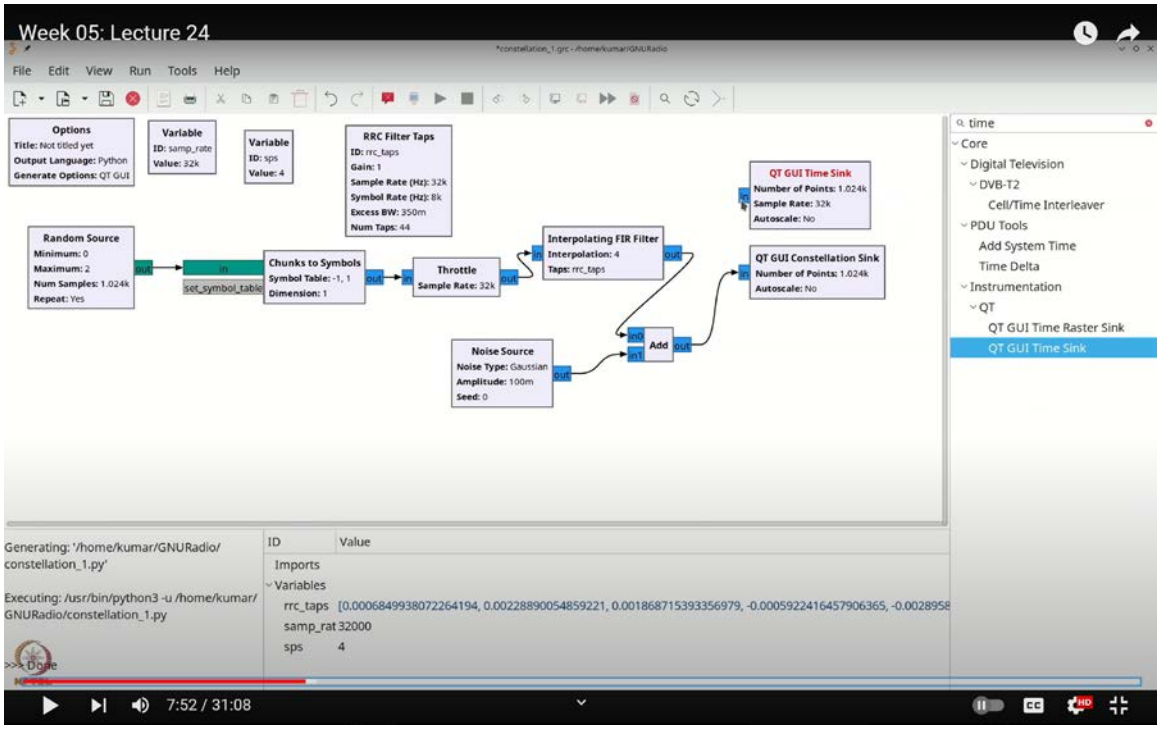
Week-05

Lecture-24

Constructing and Visualising Constellations using GNU Radio

Welcome to this lecture on Digital Communication Using GNU Radio. My name is Kumar Appiah, and I am from the Department of Electrical Engineering at IIT Bombay. In the upcoming lectures, we will put into practice our understanding of constellations and noise. Specifically, this lecture will focus on using GNU Radio to construct and visualize constellations.

(Refer Slide Time: 07:52)



We will explore various methods for generating random symbols and converting them into complex symbols that represent constellations. We'll also examine how to visualize these

constellations in GNU Radio effectively. Initially, we will manually create constellations using the "Chunks to Symbols" block. Following this, we will introduce some of the built-in features of GNU Radio for constellation creation.

Let's get started. Our first step is to add a throttle to our flow graph. You can find the throttle block by pressing Control-F (or Command-F) and searching for "throttle." Once located, place it in the flow graph.

Next, we will introduce the "Chunks to Symbols" block. Again, use Control-F (or Command-F) and search for "chunks to symbols." Place this block in the flow graph as well. Additionally, we need a random source to generate symbols. Search for "random source" using Control-F (or Command-F), and add it to the flow graph.

With these components in place, we can proceed with constructing and visualizing our constellations.

We will connect the random source to the "Chunks to Symbols" block, and then connect the "Chunks to Symbols" block to the throttle. Initially, we will configure the random source to output only two values. However, later, we will adjust it to generate 1024 samples. The symbol table will be one-dimensional, and we'll keep it as complex for convenience. We'll define the symbols as [-1, 1]. This setup means that when the random source produces a zero, the "Chunks to Symbols" block outputs -1, and when it produces a one, it outputs 1.

Next, we will introduce the constellation viewing block, which is extremely useful for debugging. Search for "qt-gui constellation sink" using Control-F (or Command-F), and drag it into the flow graph. Connect this block accordingly to complete the setup. When you run the system, you should see the constellation plot displaying +1 and -1.

To observe the impact of variations, we will add some noise. Use Control-F (or Command-F) to search for "noise" and select the noise source. Initially, set the amplitude to 0.1 and choose complex Gaussian noise. Disconnect the "qt-gui constellation sink" from its current position, add an adder block by searching for "add" using Control-F (or Command-F), and connect the noise source to the adder.

This setup introduces symbol-level noise. When you run the constellation view now, you will notice that the noise causes a spread or "blob" effect in the constellation plot. Essentially, this noise alters the 1024 samples and causes variability in the display, showing the effect of noise on the constellation.

This is a basic introduction to the qt-gui constellation sink. Now, let’s proceed with a brief exercise to perform pulse shaping and retrieve the original constellation points from our flow graph. To achieve this, we need to incorporate a root-raised cosine (RRC) pulse, which will shape the pulse of our original data, and an RRC filter to act as a matched filter. Here’s how we’ll go about it:

First, let’s generate the RRC pulse shaping taps. Press Control-F (or Command-F) and type “RRC” to find the RRC filter taps block. Place it in the flow graph and double-click it to rename it as "RRC taps." Set the gain to 1 for now, and we can adjust this later as needed.

We need to set the symbol rate to 8000, as we used 8000 symbols per second in our previous experiments. To define this, we’ll use a variable called **`samples per second`** (SPS). The symbol rate should be the sample rate divided by SPS. Remember that the double slash `//` represents integer division, ensuring that the symbol rate remains an integer. We’ll add a variable called **`SPS`** by searching for “variable” using Control-F (or Command-F) and setting its value to 4.

Once this is done, the RRC filter taps block should turn black, indicating that it is ready for use. Before adding the noise and other components, disconnect the current connections from the flow graph.

Next, we’ll perform pulse shaping using an interpolating FIR filter with the RRC taps. Search for “interpolating” using Control-F (or Command-F) to find the interpolating FIR filter. Connect the output from the throttle to this filter. We will also need to add noise later, and we'll incorporate a decimating filter as well, but we'll address that part shortly.

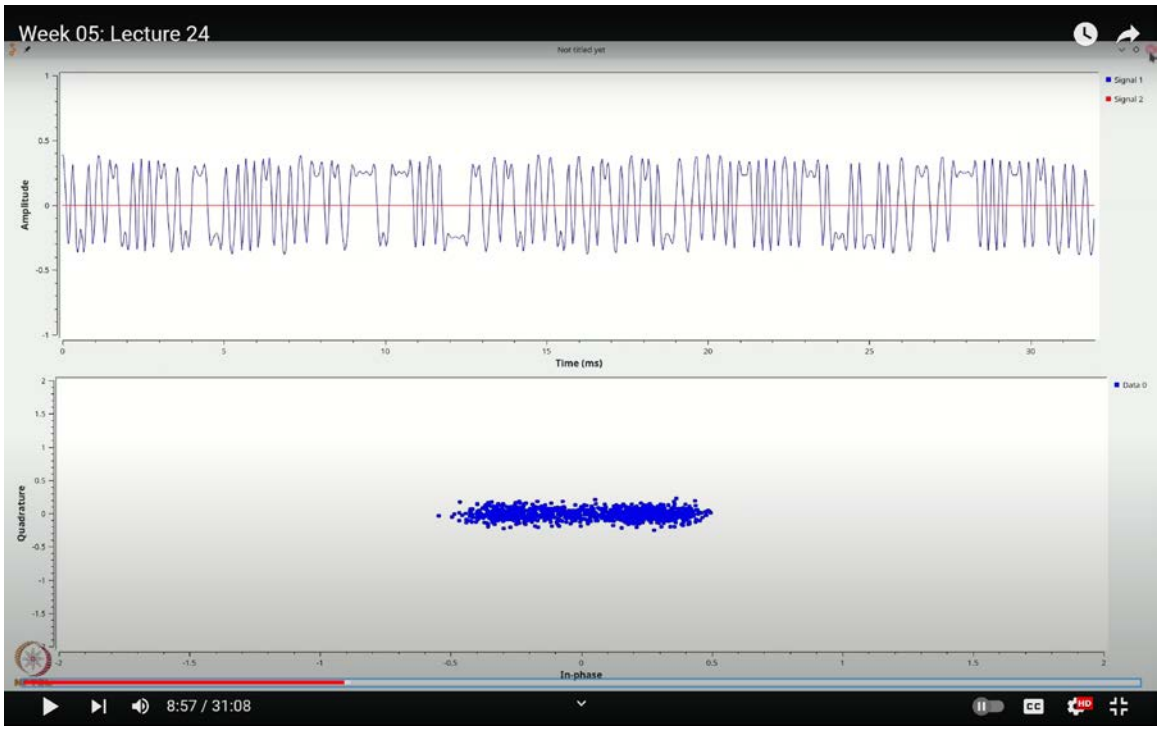
For the interpolating FIR filter, we will use the same taps as those from the RRC taps block. The interpolation factor does not need to be hardcoded; instead, we’ll use the **`SPS`** value we defined earlier. Here, **`SPS`** represents the samples per symbol.

With our interpolated signal now prepared, we need to validate the results. First, let's view it using the constellation sink and also add a time sink for a more detailed view. Press Control-F (or Command-F) and search for "time sink" to add it to your flow graph.

When you view the output, you'll see a pulse-shaped waveform. However, this visualization alone might not be particularly meaningful because it shows the raw values as plotted. What you really need are the values after match filtering, which, as we covered in previous classes, involves integrating and then multiplying by the RRC filter. In other words, to retrieve the original constellation points, we need to apply match filtering by filtering with the flipped version of the RRC filter and sampling appropriately.

To achieve this, we will use a decimating FIR filter. First, disconnect the existing components and rearrange your setup. Press Control-F (or Command-F) and type “dec” to find and add the decimating FIR filter. This filter complements the interpolating FIR filter and should use the same **`SPS`** value to ensure consistency.

(Refer Slide Time: 08:57)



For the decimating FIR filter, we will use the RRC taps, but in this case, we should use the

flipped version of the RRC filter. Although flipping the taps can be done by specifying **`[::-1]`** to reverse the sample order, it's often not necessary because the RRC filter is symmetric about its center.

The flipped version of the RRC filter is ideally the same as the unflipped version due to its symmetric properties. So, we'll proceed with this understanding. Connect the output of the noise-added signal to the constellation sink, and inspect the results. When you run the flow graph, you should see the output as expected, with some variation due to the added noise.

It’s important to note that obtaining the constellation with decimation in this setup is somewhat fortuitous. Let’s explore what happens if there's a slight delay in sampling. To illustrate this, let's introduce a delay into the system.

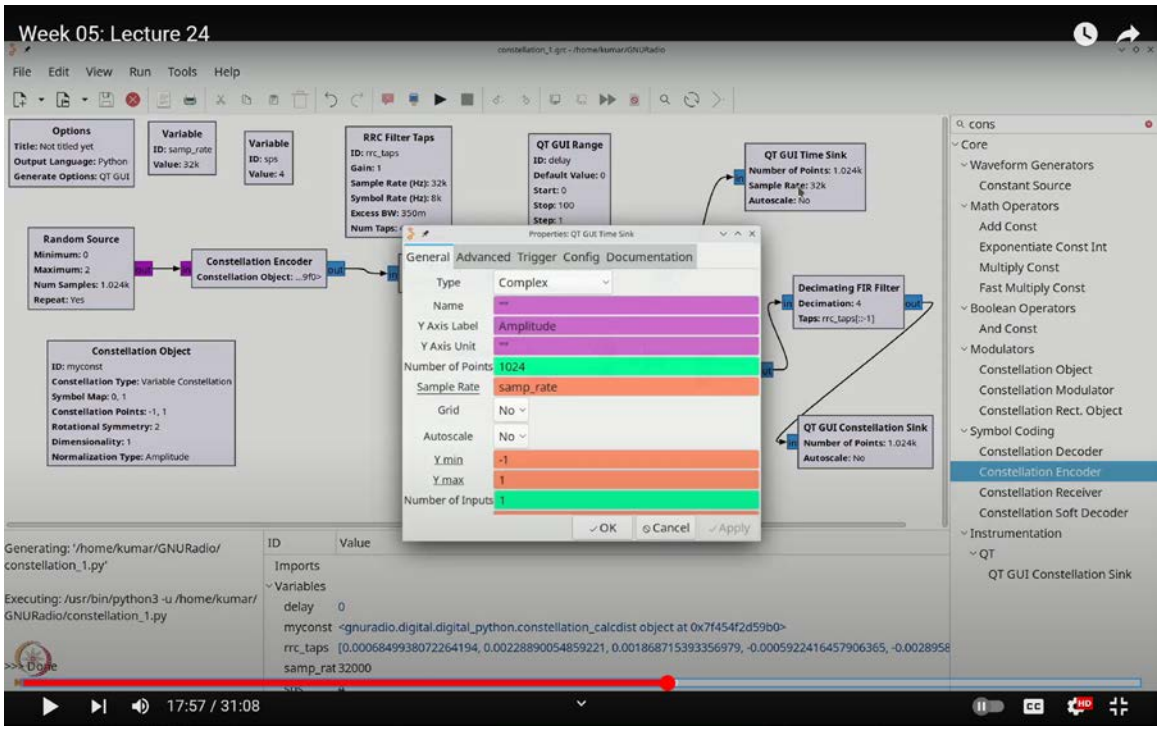
First, double-click on the FIR filter block to adjust the number of taps if needed, but for now, let's keep it as is. We'll add a delay block to the flow graph. Press Control-F (or Command-F), type "delay," and drag the delay block into your flow graph. Connect it appropriately and set the delay to a variable named **`delay`**.

Next, add a Qt GUI Range block to control the delay. Press Control-F (or Command-F), type "range," and place the range block on your flow graph. Name it **`delay`**, set the default value to 0, and allow the range to go up to 10. Ensure it is an integer type. Click OK to apply these changes.

Now, run the flow graph and introduce some delay. You will observe that the constellation becomes distorted. This distortion occurs because the sampling point is not correctly aligned. In other words, although you have applied match filtering, the sampling does not occur at the correct point. To further illustrate, let’s adjust the samples per second to 5 and rerun the flow graph.

Now, you may notice that the constellation plot doesn’t look very clear or pleasing initially. However, if you adjust the settings, you will obtain the correct constellation. In fact, if you set the noise level to zero and execute the flow graph, you will see that the constellation appears blurred. Adding a delay of 1 sample will restore the original constellation.

(Refer Slide Time: 17:57)



Why does this happen? The reason lies in the nature of the interpolating FIR filter: it is a causal filter, not an instantaneous one. Consequently, it introduces a certain amount of delay. Specifically, you can estimate this delay based on the number of taps in the filter. For instance, with 55 taps, this filter introduces approximately 27-28 samples of delay. Additionally, any further filtering will introduce another 27-28 samples of delay.

If you calculate the total delay introduced by the filter, it could be around 55 samples. Since the decimating filter must sample at the exact right point, you need to account for this delay to ensure proper alignment. For example, changing the samples per second to 6 might work due to matching multiples, and adding a delay of 6 samples aligns the system correctly.

If you try other delays, such as 12 samples, the system works, but delays like 11 or 10 samples may not align properly. This indicates that to achieve the exact sampling point, precise adjustments using delay may be necessary. This technique will be useful in your future experiments. To determine the exact mechanism, you can calculate the group delay of the interpolating FIR filter and the group delay introduced by the decimating FIR filter,

then take the modulo with respect to the symbols per second. This will help you find the correct sampling point.

I’m setting the samples per second to 4 and adding back the noise at a level of 0.1. This will restore the original constellation.

Using the "Chunks to Symbols" block for setting up constellations can be cumbersome. Fortunately, GNU Radio offers a more straightforward and systematic method for generating constellations. So, let’s remove the "Chunks to Symbols" block and introduce the "Constellation Encoder" block instead.

To do this, press Ctrl+F (or Cmd+F) and search for "constellation object." Double-click to choose from a variety of pre-built constellations. For instance, let’s select BPSK and click OK. Next, we need to add a "Constellation Encoder."

Again, press Ctrl+F (or Cmd+F) and search for "CONST" to find the "Constellation Encoder" block. This block takes byte inputs, so let’s modify our random source by double-clicking it and selecting "byte." Then, connect the random source to the constellation encoder.

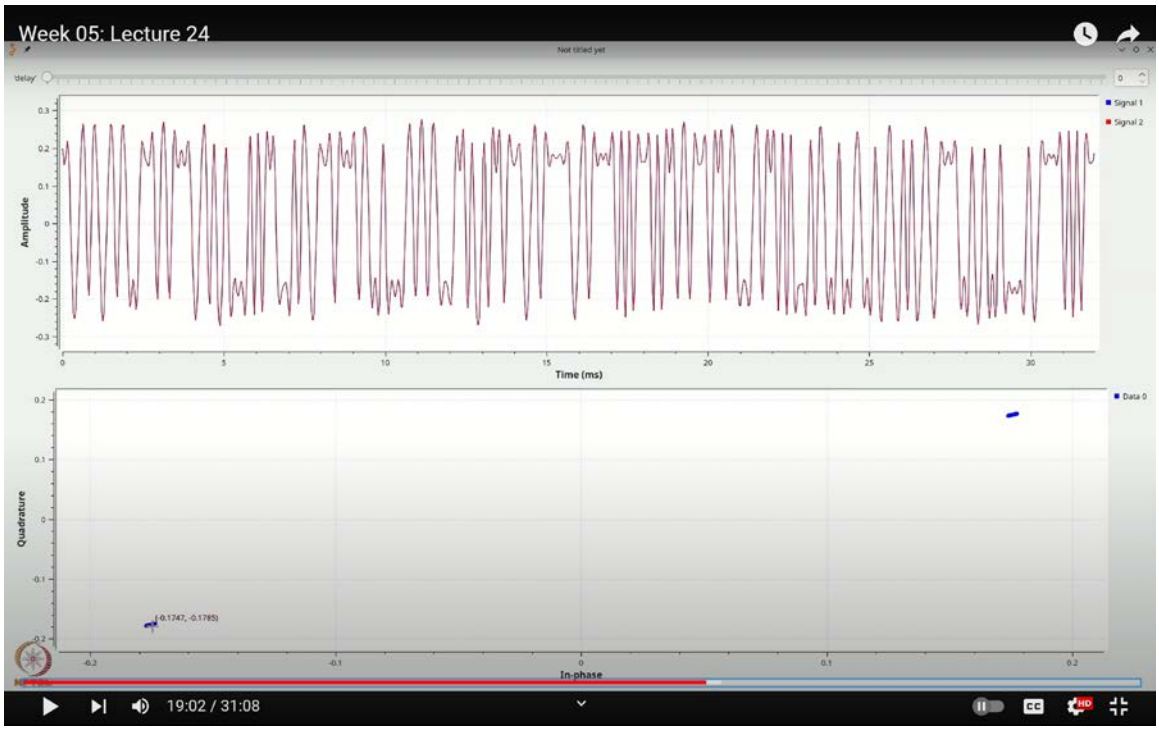
The "Constellation Object" is essentially a variable, so let’s name it **`myCONST`**. The constellation encoder will use this object as its input. Consequently, any adjustments to the constellation object will automatically update the encoder.

Let’s see the results. Running this flow graph should give us the expected output. To further explore, you can make changes to the constellation object. For example, if you switch to QPSK and adjust the random source to output up to 4 values, you’ll start seeing QPSK samples, and the waveform will adapt accordingly.

Let’s return to the BPSK configuration. I’ll set this to 2 and modify the constellation to be variable. You can now specify an arbitrary constellation by defining a list of integers. These integers represent the symbols output by the constellation encoder. For instance, if the random source outputs a 3, the third symbol in your list will be selected. Similarly, if it produces a 2, the second symbol in the list will be output, and so on. This flexibility will

be particularly useful when dealing with PITS (Pulse Integral Time-Space).

(Refer Slide Time: 19:02)



Let’s simplify this. We’ll set our constellation to include 0 and 1, with the actual symbols being -1 and 1. For this, select "OK" and set the rotational symmetry to 2. Running this configuration should yield the expected results.

Before proceeding, let’s enhance our visualization. Double-click the time sink to add a grid and enable auto-scaling. Do the same for the constellation sink.

Next, let’s adjust the constellation object. Change it to include symbols such as -1 - 1j and 1 + 1j. This adjustment will place the symbols at the bottom-left and top-right corners of the plot. Notice that there’s an option for normalization, which performs amplitude-based normalization.

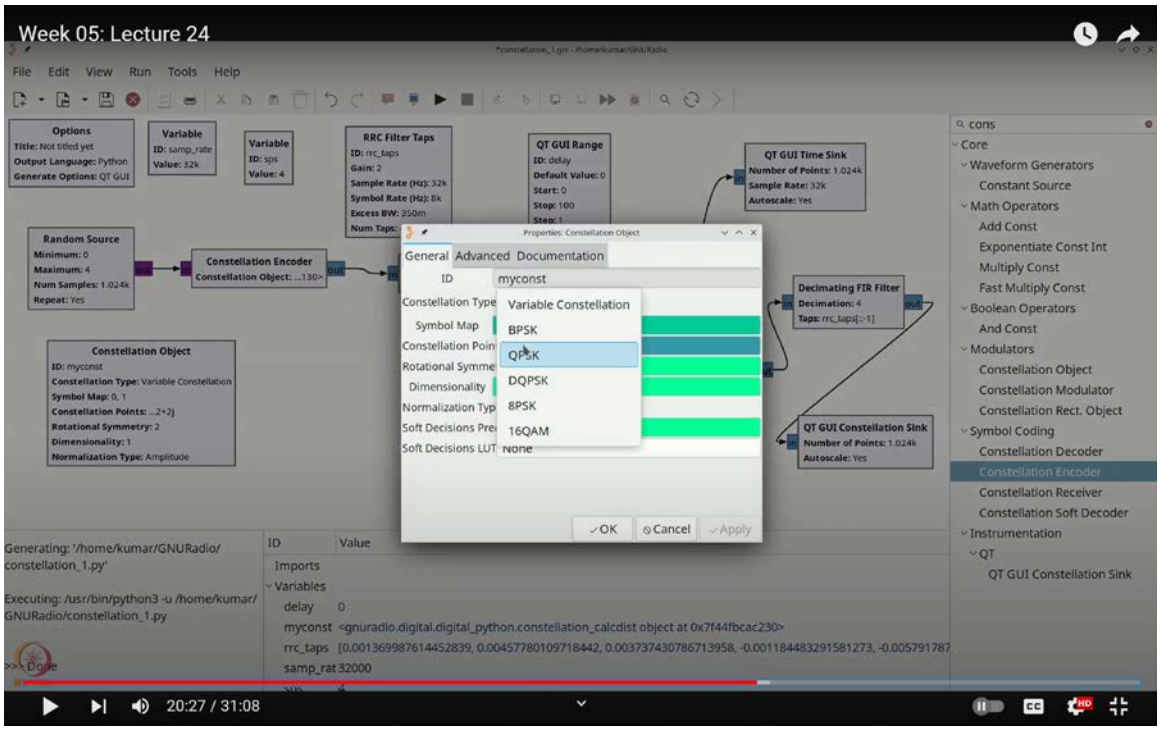
Execute the flow graph with these changes. You should observe that the constellation appears constrained between approximately -0.2 and 0.2. This result is due to the filtering process, which causes a reduction in gain.

For example, even if the noise is set to 0, executing this flow graph will result in constellation points around -0.17, -0.17, and 0.17, 0.17. To correct this, you need to adjust the gain in the RRC filter taps. The gain should be precisely $\sqrt{SPS}$, where SPS represents the samples per symbol. You can input this as **`SPS ** 0.5`** in your configuration. By doing this, your constellation points will be positioned around -0.7, -0.7, 0.7, and 0.7, which is very close to $\pm\frac{1}{\sqrt{2}}$. This adjustment ensures that the points are equidistant from the origin, which is crucial for accurate constellation representation.

This technique proves useful because even if you set your constellation to something like -2, -2 + 2j, the normalization process will adjust and you'll still obtain the set of values -0.7, -0.7, 0.7, and 0.7. This makes the constellation much easier to manage.

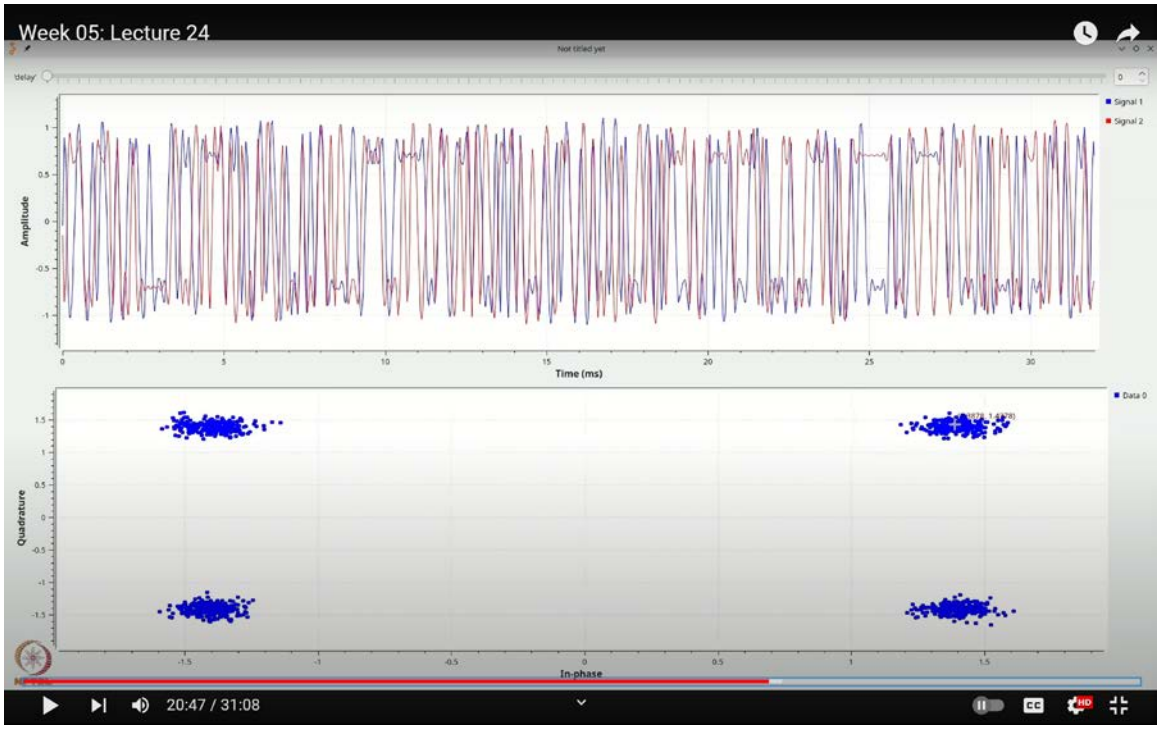
Now, let's reintroduce the noise, setting it to 0.1. You'll observe that the constellation becomes slightly unstable, but its center remains around $-\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}$ and $\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}$. This method is an effective way to generate and work with different constellations.

(Refer Slide Time: 20:27)



You can easily switch back to QPSK by adjusting the settings to range from 0 to 4 and selecting QPSK as your constellation type. This will allow you to see the QPSK constellation in action. Feel free to experiment with various constellations and observe the corresponding waveforms.

(Refer Slide Time: 20:47)



An interesting point to note is that, due to the amplitude-based scaling, the values are close to $\sqrt{2}, \sqrt{2}$. If you double-click the constellation object, you can set it to a variable constellation and define your own QPSK constellation, which will provide different results. Experimenting with these settings will help you understand their effects. GNU Radio offers a convenient wrapper for using constellations, particularly because combining constellations with pulse shaping is a common practice.

There is also a component known as the constellation modulator. To explore its functionality, press **`Ctrl-F`** or **`Cmd-F`**, type "constellation," and select the constellation modulator. This tool can be quite useful for your simulations, so let’s delve into its operation.

Before we start, let’s add a throttle to manage the flow rate. Press **`Ctrl-F`** or **`Cmd-F`**, select the throttle, and place it onto the flow graph.

Next, let’s set up some variables similar to our previous flow graph. Press **`Ctrl-F`** or **`Cmd-F`**, type "variable," and place a variable block. We’ll create a variable named **`SPS`** and set its value to 4.

Now, add a random source by pressing **`Ctrl-F`** or **`Cmd-F`**, typing "random source," and placing it onto the graph. Double-click the random source to configure it to output values from 0 to 2, and ensure it's set to byte format, as required by the constellation modulator.

Create a constellation object by pressing **`Ctrl-F`** or **`Cmd-F`**, typing "constellation object," and placing it on the graph. Double-click this object and name it **`myconst`**. You may keep the constellation settings as is for now, but change the random source to output up to 4 values.

Next, add the constellation modulator by pressing **`Ctrl-F`** or **`Cmd-F`**, typing "constellation modulator," and placing it on the graph. Connect the output of the random source to the constellation modulator and link the modulator’s output to the throttle.

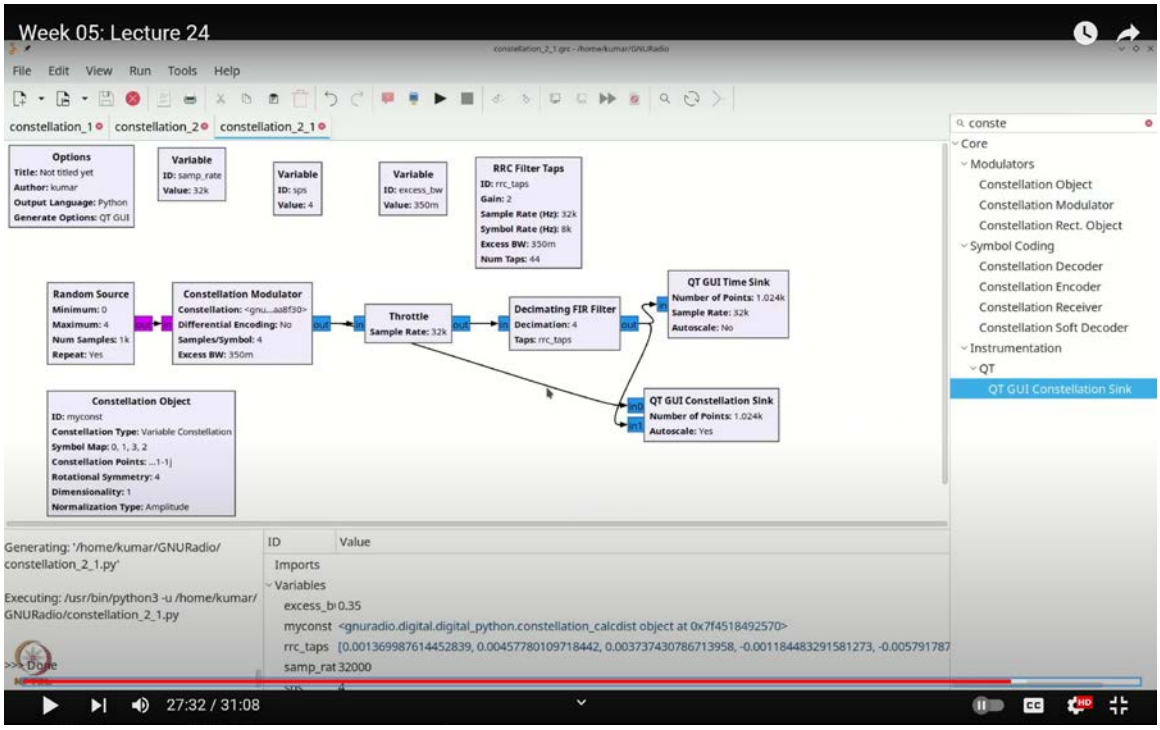
Double-click the constellation modulator to configure it. Set the constellation parameter to **`myconst`**, disable differential encoding by selecting "NO" (we will discuss differential encoding in detail later), and set the samples per symbol to **`SPS`**.

The constellation modulator is designed to handle root-raised cosine filtering for you. This means that the excess bandwidth term is accounted for within the modulator, so you can expect its output to include the root-raised cosine pulse shaping directly.

To verify the root-raised cosine filtering, let’s add a time sink. Press **`Ctrl-F`** or **`Cmd-F`**, type "qtgui time sink," and place it onto the flow graph. When you connect it, you'll see that the pulses are shaped according to the root-raised cosine filter. If you verify this, although I won't go through the details here, you should find that the pulses are spaced exactly $\frac{1}{8000}$ seconds apart. This spacing results from a sampling rate of 32,000 samples per second and upsampling by a factor of 4.

Next, let’s retrieve the constellation. For this, we could use a constellation receiver to reverse the effects of the constellation modulator. However, for now, we’ll manually retrieve the signal as we did earlier, since the constellation receiver offers additional features that we’ll cover later.

(Refer Slide Time: 27:32)



First, let’s remove the time sink by selecting it and hitting delete. We’ll then replicate our previous steps: generate the root-raised cosine (RRC) filter taps used by the constellation modulator and perform root-raised cosine match filtering.

Press **`Ctrl-F`** or **`Cmd-F`**, type "decimating FIR filter," and add this block to the graph. Set the excess bandwidth to 0.35 by creating a variable named **`Xs_Bw`**. Configure the decimating FIR filter with **`Xs_Bw`** for the excess bandwidth and connect the taps to the **`RRC_taps`**. Although you can flip the taps using **`:: -1`**, we won’t do that this time.

Now, to create the RRC filter taps, press **`Ctrl-F`** or **`Cmd-F`**, type "RRC," and add the RRC filter taps block. Name it **`RRC_taps`**. Set the gain to $\text{SPS}^{0.5}$ for normalization. While this step isn’t mandatory, it simplifies things. Set the symbol rate to $\frac{\text{sample rate}}{\text{SPS}}$, which

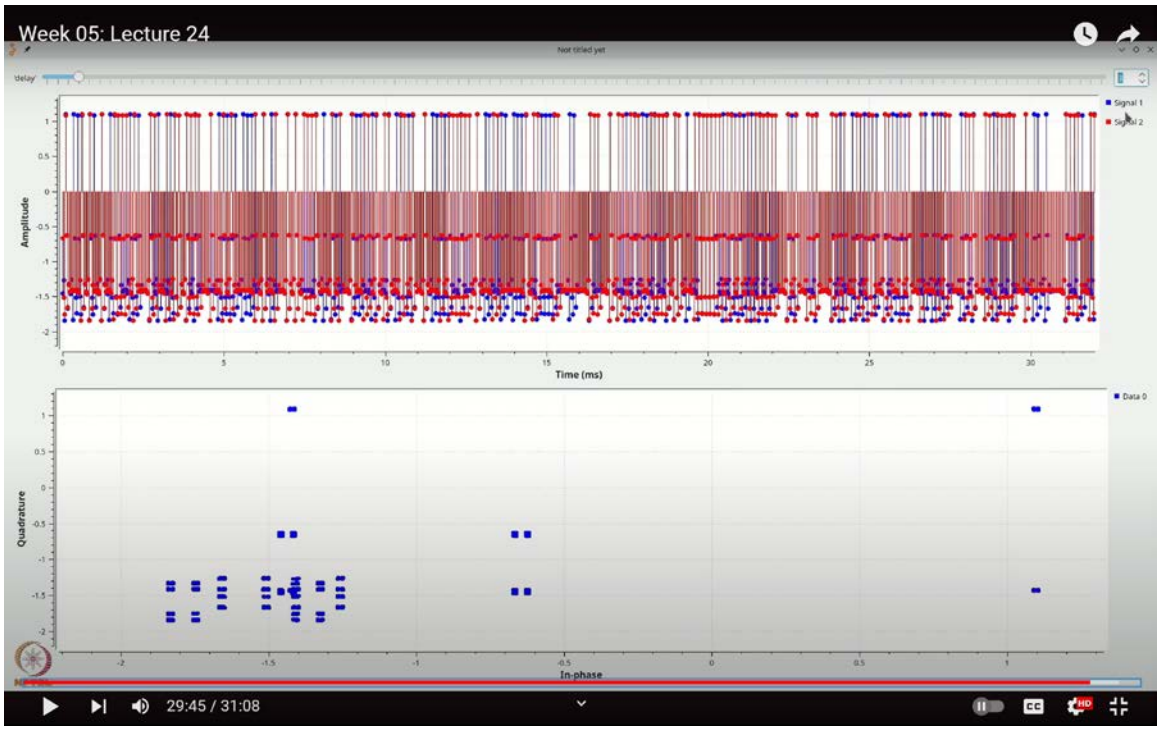
should give you 8,000, and ensure the excess bandwidth is set to **`Xs_Bw`**.

Let's proceed by adding a time sink and a constellation sink to visualize the decimated output. Press **`Ctrl-F`** or **`Cmd-F`**, type "time sink," and place it on the flow graph. Similarly, add a constellation sink by pressing **`Ctrl-F`** or **`Cmd-F`**, and typing "constellation sink." For the constellation sink, let's enhance the display by adding a grid and enabling auto-scaling.

Next, connect the output of your decimated signal to the constellation sink. Initially, avoid connecting the original constellation, as it is pulse-shaped and not suitable for this direct comparison.

Now, execute the flow graph. What you'll see is that even after match filtering, some pulse shaping effects persist. This is evident as the constellation plot shows variations, particularly noticeable at the lower end, close to the -1.5 mark. The constellation variations reflect changes between approximately -2 and -1.2, which appear in the time sink display as well.

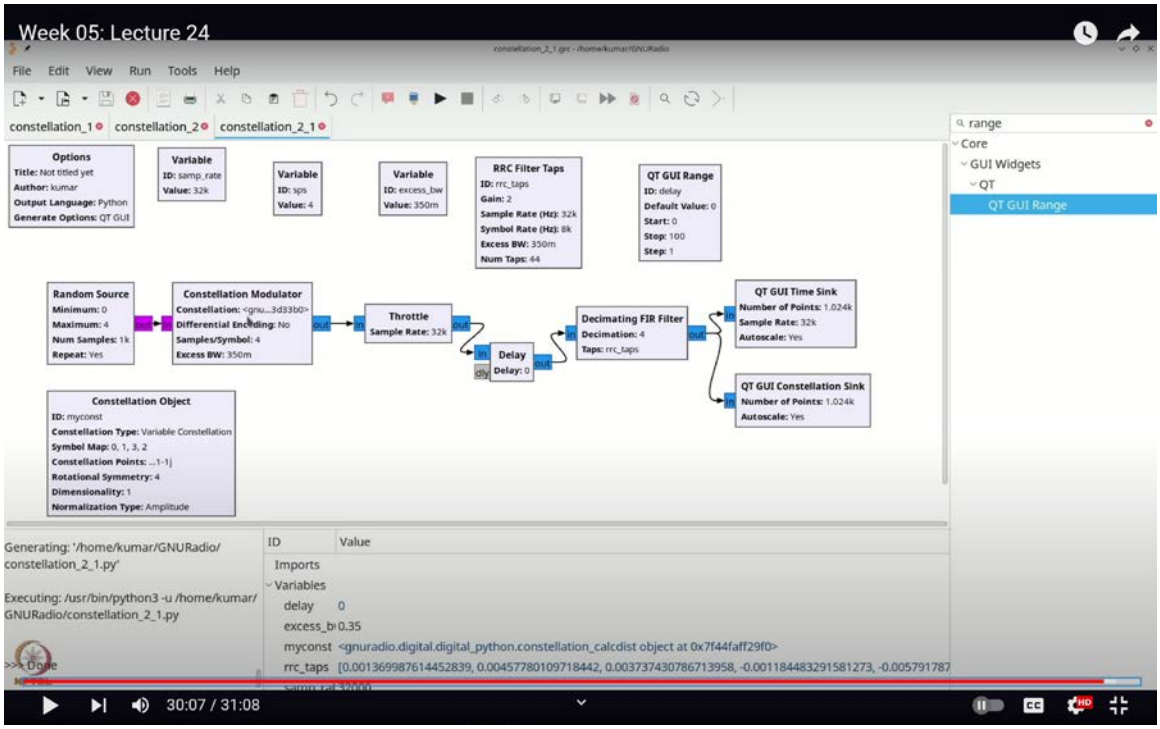
(Refer Slide Time: 29:45)



This indicates that while match filtering is occurring, the sampling point is not correctly aligned. This misalignment happens because we may not fully account for the delay introduced by the constellation modulator. To correct this, we'll follow the same approach we used earlier: remove the decimating filter and introduce a delay to realign the sampling point.

To incorporate a delay, press **`Ctrl-F`** or **`Cmd-F`** and type "delay." Add a delay block to the flow graph and connect it as needed. Double-click on the delay block and name it "delay." Next, press **`Ctrl-F`** or **`Cmd-F`** again, type "range," and add a range block. Name this range block "delay" and set its default value to 0, allowing it to run from 0 to 100, even though the effective range should be from 1 to samples per second.

(Refer Slide Time: 30:07)



Execute the flow graph and set a one-sample delay. Observe that the constellation changes accordingly, showing variations. Increasing the delay by one sample further sharpens the constellation. If you examine the stem plot, you'll see that the values remain consistently between positive and negative root 2 levels. Adjusting the delay more, you will notice that

a two-sample delay is sufficient to correctly align the constellation.

This exercise highlights the importance of properly adjusting the delay when using the built-in constellation modulator object, which includes pulse shaping. Throughout this lecture, we've explored various methods for generating constellation points from random data symbols using GNU Radio. In future lectures, we will build upon this foundation to generate QPSK, QAM, or BPSK symbols and assess their performance under conditions like additive white Gaussian noise and other impairments. Thank you.