Digital Communication using GNU Radio

Prof. Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

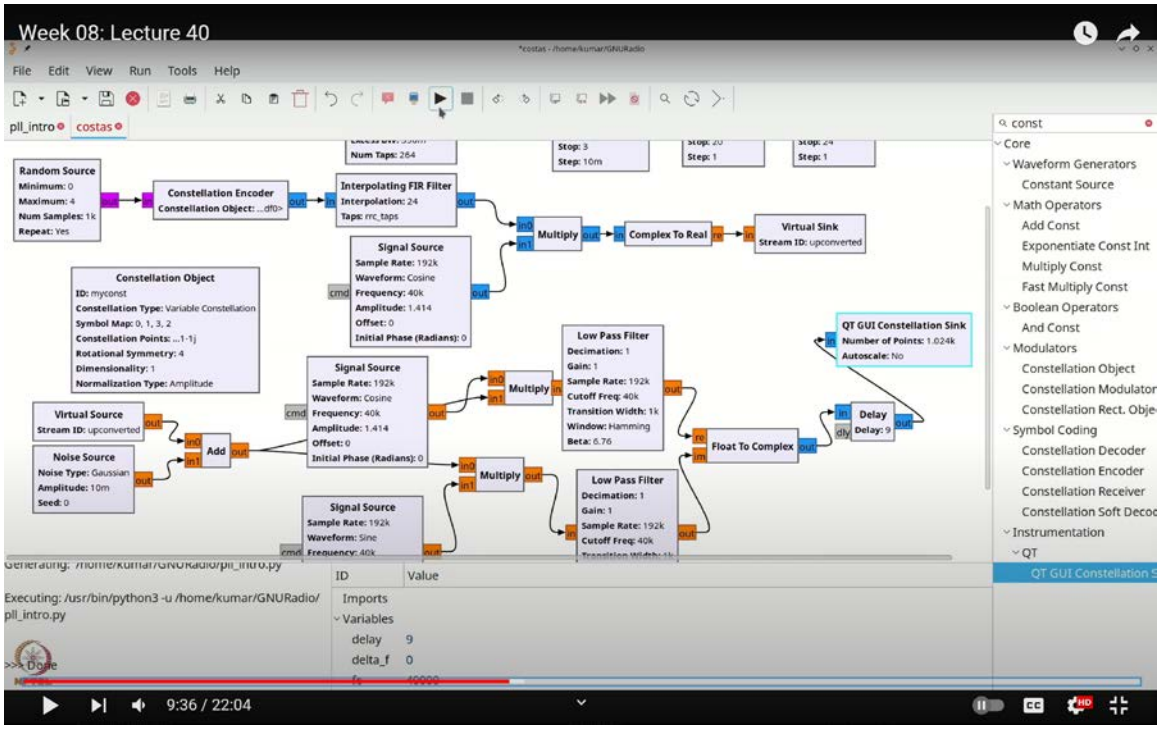
Week-08

Lecture-40

Costas Loop and Differential PSK in GNU Radio

Welcome to this lecture on Digital Communication using GNU Radio. My name is Kumar Appiah, and in this session, we will delve into techniques that allow us to communicate effectively without having to worry about frequency or phase offsets.

(Refer Slide Time: 09:36)



We will focus on two specific approaches: the Costas loop and Differential Phase Shift Keying (DPSK). The Costas loop, as you may recall, is a powerful tool for tracking frequency offsets in PSK symbols. For instance, with PSK symbols like QPSK, you can leverage higher-order powers of the PSK symbols, such as raising to the power of 4, to

estimate and subsequently correct for frequency offsets.

The second approach we'll explore is Differential Phase Shift Keying. In DPSK, information is encoded not in the PSK symbols themselves, but in the transitions between successive PSK symbols. This method minimizes the impact of frequency offsets by using the difference between the current and previous symbols.

To get started with these techniques in GNU Radio, let's first add a random source. Use **`Ctrl F`** or **`Cmd F`** to search for and select the random source block. We will increase the sampling rate to 192,000 and configure the random source to output bytes ranging from 0 through 4.

Next, we need a constellation encoder. Search for **`CONSTE`** using **`Ctrl F`** or **`Cmd F`** to find and add the constellation encoder block. Connect this encoder to a constellation object. We'll use a standard QPSK constellation, which we will name **`myCONST`**.

To model the impact of noise, we'll add an additional modulation step. Let's start by performing upconversion. Search for **`interpolating FIR filter`** with **`Ctrl F`** or **`Cmd F`** and add an RRC (Root Raised Cosine) filter. This will allow us to manage the modulation as we did in previous examples. Now, let's connect everything and observe how these techniques function in the presence of noise and frequency offsets.

We will designate this as **`RRC_taps`** and set the gain to 4.9. The sampling rate will be **`samp_rate`**, with a symbol rate of 8000 symbols per second. To manage the symbols per second (SPS), create a variable by pressing **`Ctrl F`** or **`Cmd F`** and typing **`variable`**. Name this variable **`SPS`** and set it to the integer value of **`samp_rate`** divided by 8000, ensuring that we have 8000 symbols per second.

Now, set the **`RRC_taps`** for interpolation to match **`SPS`**. With our upconverted signal established, we will next add a carrier. First, refine the carrier frequency by searching **`Ctrl F`** or **`Cmd F`** for **`Fc`** and setting it to 40,000 Hz. Then, add a signal source by searching for **`signal source`**. Configure this source with a frequency of **`Fc`**, an amplitude of 1.414 (which approximates $\sqrt{2}$), and connect it accordingly.

To finalize, we need to extract the real part of the signal. Search for **`complex to real`** using **`Ctrl F`** or **`Cmd F`**, and use this block to obtain the real component of our signal. This completes the setup for our upconverted transmit signal.

Next, add a virtual sink to continue the process. Use **`Ctrl F`** or **`Cmd F`** to search for **`VIRT`** and select a virtual sink. Connect this virtual sink to the output and name it **`up_converted`**. Add a virtual source and call it **`up_converted`** to retrieve the same sequence.

To analyze the impact of noise, add a Gaussian noise source. Start by creating a range to control the noise variance. Search for **`range`** using **`Ctrl F`** or **`Cmd F`**, name it **`noise_std`**, and set the default value to 0.01. Allow it to vary from 0 to 3 with a step of 0.01 to establish the noise standard deviation.

Next, add a noise source by searching for **`noise source`** with **`Ctrl F`** or **`Cmd F`**. Choose a real noise source, set it to float, and configure the amplitude to **`noise_std`**. Add this noise to the signal by searching for **`add`** using **`Ctrl F`** or **`Cmd F`**, select the add operator, and set it to float. This results in our upconverted noisy signal.

We will now repeat the previous steps, but this time we will introduce a frequency offset. We need to mix the signal with both cosine and sine components, apply a low pass filter, and incorporate an additional frequency offset. Create a range for the frequency offset by searching **`range`** using **`Ctrl F`** or **`Cmd F`**, name it **`delta_F`**, and set it to vary from -20 Hz to 20 Hz.

To implement this, we need two signal sources for mixing. Search for **`signal source`** using **`Ctrl F`** or **`Cmd F`**. Configure the first source as a cosine with a frequency of **`Fc + delta_F`**, no initial phase, and an amplitude of 1.414. Duplicate this source (using **`Ctrl C`** and **`Ctrl V`**), configure the duplicate as a sine source, and adjust its phase to negative to ensure correct mixing.

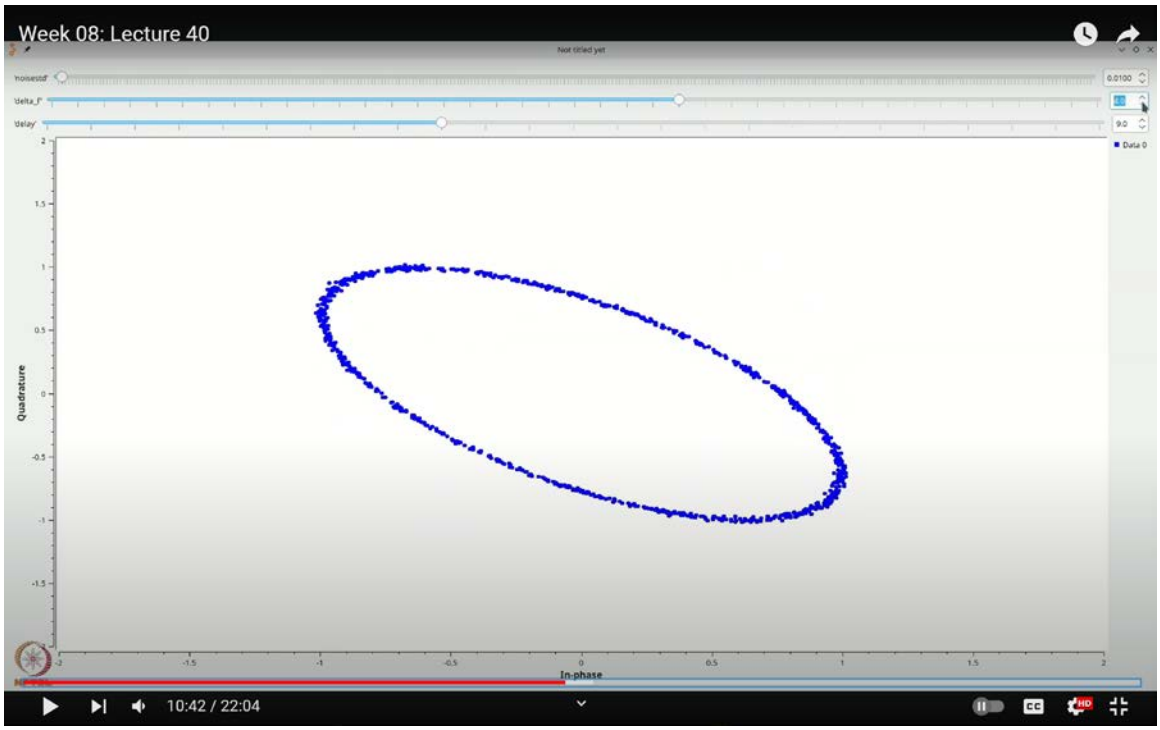
We will now add multiply blocks. To do this, copy the block using **`Ctrl C`** and **`Ctrl V`**, and configure it to float. Connect the signal source to this block, then duplicate it once more (**`Ctrl C`**, **`Ctrl V`**) and make the necessary connections.

Next, we need a pair of low pass filters. Search for **`LOW`** using **`Ctrl F`** or **`Cmd F`**, and select the low pass filter block to remove the two **`fc`** components. Double-click the filter block and set the cutoff frequency to **`fc`**, with a transition width of 1000. Ensure the filter is set to float-to-float. Copy and paste this filter block (**`Ctrl C`**, **`Ctrl V`**) and connect it accordingly.

We will then need to connect this to a complex block. Search for **`complex`** using **`Ctrl F`** and select the float-to-complex block. We also need to add a delay and a decimating filter.

Start by adding a delay block. Search for **`range`** using **`Ctrl F`** or **`Cmd F`**, and call this range **`delay`**. Set the integer delay to range from 0 to **`SPS`**. From previous setup, we know the delay was 9, so set the default value to 9. Search for **`delay`** using **`Ctrl F`** or **`Cmd F`**, select the delay block, and set the delay to this value.

(Refer Slide Time: 10:42)



Next, add a constellation sync block. Search for **`CONST`** using **`Ctrl F`** or **`Cmd F`**, select the constellation sync block, and connect it appropriately.

Before adding the constellation sync, we need to incorporate a decimating FIR filter. Search for **`decimating FIR filter`** using **`Ctrl F`** or **`Cmd F`**. Rotate the block if necessary, and set the decimation to **`SPS`** with **`RRC_taps`** as the filter taps. Connect the output of this filter to the constellation sync.

Execute the flow graph to see the constellation diagram. If you observe a rotation when introducing any frequency offset **`delta_F`**, this indicates that the phase rotation is problematic, affecting the recovery of symbols. To address this, we will add a Costas loop.

Search for **`COSTA`** using **`Ctrl F`** or **`Cmd F`**, and select the Costas loop block. This loop requires three parameters. Connect the Costas loop, and set the loop bandwidth to $\frac{2\pi}{100}$, which is 6.28 divided by 100. For QPSK, set the order to 4.

Execute the flow graph. Even with a frequency offset, the Costas loop should correctly track the frequency and phase, compensating for any rotation. If you increase the frequency offset, you will observe that the Costas loop can still effectively track the changes.

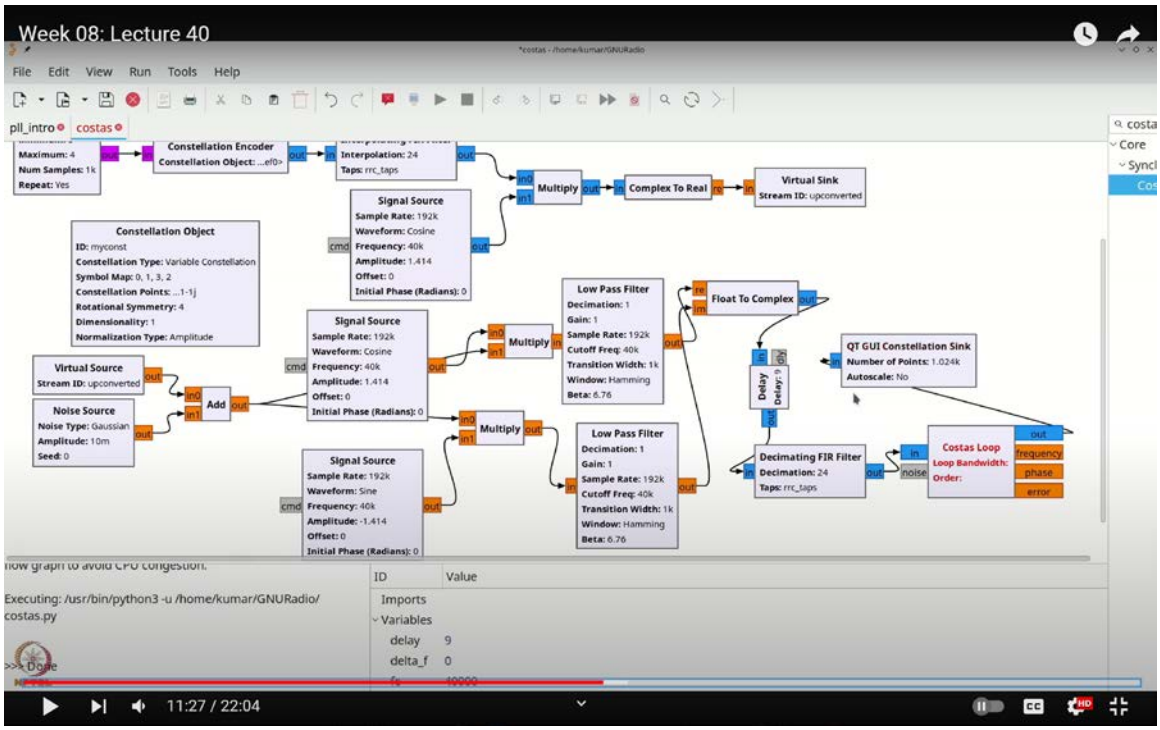
To verify, view both the rotated and de-rotated constellations simultaneously. Double-click on the constellation sync block, increase it to two inputs, and connect the original constellation to the first input and the rotated constellation to the second.

As you test with different frequency offsets, the blue constellation will rotate due to the offset, while the red constellation, processed by the Costas loop, will remain aligned. This demonstrates the loop's ability to track frequency and phase offsets effectively. However, as discussed, if the frequency offset exceeds certain limits, the performance of the Costas loop will diminish due to additional phase shifts introduced by $e^{j2\pi\Delta ft}$.

As you increase the frequency offset further, you will notice that the constellation points begin to spread out. This spreading results in a decrease in your signal-to-noise ratio (SNR), eventually leading to poorer performance compared to the original carrier recovery algorithm. If the frequency offset becomes too large, the system may lose lock, and additional noise will exacerbate performance degradation. While the Costas loop significantly simplifies implementation, it does come with its own limitations. For

example, if the frequency offset is excessive, the Costas loop may struggle to maintain lock and ultimately fail.

(Refer Slide Time: 11:27)



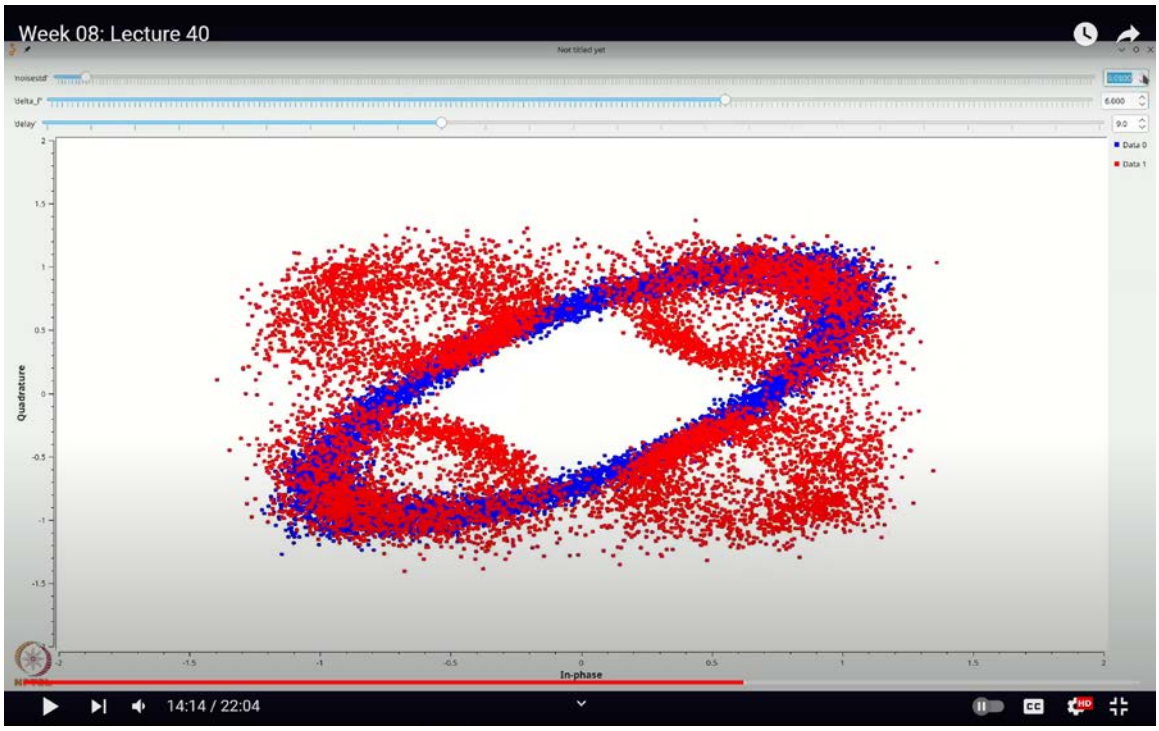
To illustrate, let’s explore the performance of Differential Phase Shift Keying (DPSK) with a simplified example. We'll start by setting the sampling rate to 192,000. Save this configuration with the name **`DPSK_4`**.

First, add a random source by searching for "random source" with **`Ctrl F`** or **`Cmd F`**. Configure the random source to generate bytes ranging from 0 to 4. Next, add a constellation object and a constellation encoder. Place the constellation object below the constellation encoder. Double-click the constellation object, rename it to **`my_const`**, and configure it to use that constellation.

Assume the output is differentially encoded for this example, even though it may not be the case. Search for "throttle" using **`Ctrl F`** or **`Cmd F`** to add a throttle block. Then, add noise by searching for "range" with **`Ctrl F`** or **`Cmd F`**, and configure it to vary from 0 to 3 with a step of 0.1. Also, add a frequency offset by creating another range block for

`Delta_F`, which will vary from -10 to 10 with a step of 0.1.

(Refer Slide Time: 14:14)



To add noise and rotation, search for "add" using **`Ctrl F`** or **`Cmd F`** and connect this block to a noise source. Search for "noise source" and configure it appropriately. Use a multiplier to introduce frequency offset rotation by adding a signal source. Set the frequency of this signal source to **`Delta_F`**.

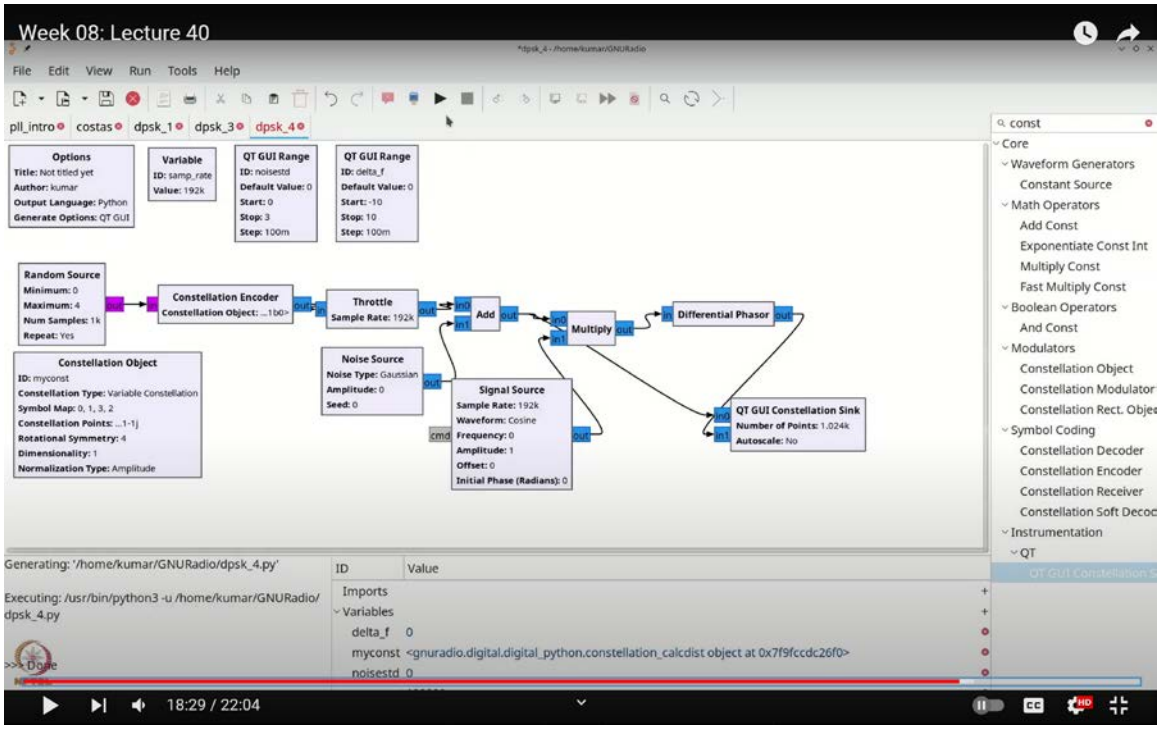
For differential decoding, search for "differential" using **`Ctrl F`** or **`Cmd F`** to add a differential phase decoder. This decoder will take successive samples, conjugate them, and perform the differential decoding. Connect the output to a constellation sync block, which will allow you to view both the original and the differentially decoded constellations.

Double-click on the constellation sync block, set it to two inputs, and connect the original constellation with rotation to the first input, and the differentially decoded signal to the second input.

Now, execute the flow graph. Even with some noise, you should see the constellation points

clearly. The differential decoder should correctly handle phase differences corresponding to $\pi/2$, π, $3\pi/2$, and 2π. This demonstrates that differential encoding is robust against small frequency offsets.

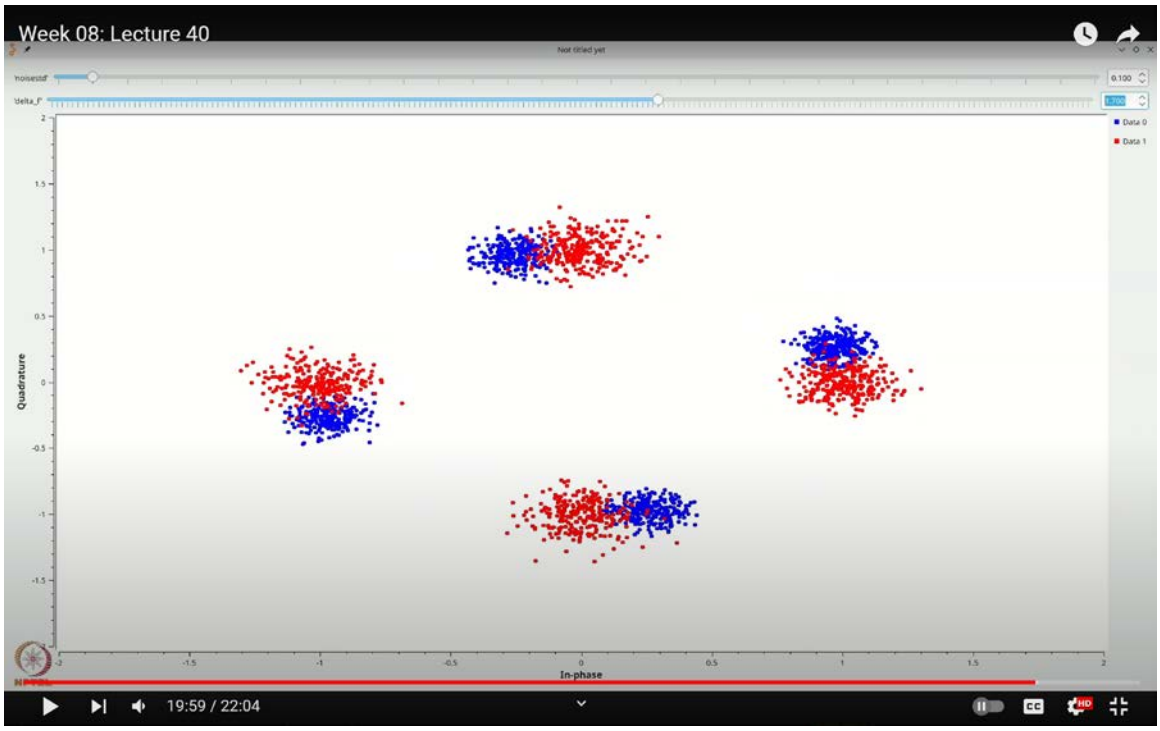
(Refer Slide Time: 18:29)



To verify, increase the frequency offset and add noise. You will observe that while the constellation rotates due to the frequency offset, the red constellation, processed through differential decoding, remains stable and correctly aligned. This indicates that the differential phase decoding effectively compensates for the frequency offset, maintaining the integrity of the signal.

Let’s increase the number of points by a factor of 10. When you execute this with noise, you’ll notice that the constellation rotates, while the differentially decoded constellation remains stable. For slow rotations, differential decoding performs quite well. However, as the rotation speed increases, the system becomes more susceptible to noise, which can pose a problem. This demonstrates that differential detection, particularly the red constellation, is more robust and less prone to rotation, especially for small frequency offsets.

(Refer Slide Time: 19:59)



In this lecture, we've explored techniques that allow us to operate without relying heavily on a Phase-Locked Loop (PLL) for frequency tracking. Instead, we used the Costas loop to track the frequency. However, it's important to note that the Costas loop has its limitations. It is effective only in specific scenarios and primarily for PSK constellations.

Differential Phase Shift Keying (DPSK) leverages the fact that information is encoded in the transitions between successive symbol phases, which helps mitigate the effects of frequency and phase offsets to some extent. Yet, DPSK is less robust than coherent detection strategies that employ a PLL for frequency locking and tracking. This is because frequency offsets can shift your decision regions. For example, in differential phase shift keying with four symbols, slight deviations can make the system less resilient to noise, as it relies on the assumption that offsets won't significantly disrupt the signal. These concepts have been covered in this series of lectures. In the upcoming lectures, we will shift our focus to another impairment: how the medium or channel affects performance. We'll explore how distortions introduced by the medium can be identified and corrected. Thank you.