Automatic Stream Delay Calibration in GNU Radio

Ian Spika

IPv6 & IoT 2025 FEI STU

May 2025

github.com/spikovich/gnuradio-delay-calibrator

Abstract:

This work details the design and implementation of a system for automatic time delay calibration between two identical data streams within the GNU Radio framework. The objective was to create a feedback mechanism capable of determining and compensating for an unknown delay in one of the streams. The solution employs an Embedded Python Block (EPB) that implements a cross-correlation algorithm to calculate the time offset between the streams. User control over the calibration process is provided via a graphical user interface (GUI) featuring 'Start Calibration' and 'Reset Calibration' functionalities. The system's performance demonstrates successful stream synchronization, validated by the minimization of the difference signal between them to a near-zero level

Keywords:

GNU Radio, Embedded Python Block, signal processing, stream synchronization, delay calibration, cross-correlation, automatic control.

Introduction:

In modern digital signal processing systems, particularly when dealing with multiple data sources or distributed sensor networks, precise time synchronization of data streams is of critical importance. Discrepancies in signal arrival times can lead to errors in their joint processing, such as in phase difference calculations, coherent summing, or comparative analysis. Manual adjustment of delays can be laborious and inefficient, especially if the delay is variable or unknown beforehand.

The primary goal of this project is to develop an automatic delay calibration system within the GNU Radio environment. The task, as outlined in the assignment, involves creating a flowgraph with an automatic feedback loop that, upon execution, can identify the correct delay value to compensate for the misalignment of identical sample streams.

To implement this, an Embedded Python Block (EPB) approach within GNU Radio was chosen. This block encapsulates the logic for analyzing input streams, computing their cross-correlation to determine the time offset (lag), and subsequently updating the delay parameter of a compensating delay block within the GNU Radio flowgraph. User interaction for controlling the process is facilitated through "Start Calibration" and "Reset Calibration" buttons in a QT GUI.

This report will present the architecture of the developed solution, detail the implementation of the calibration algorithm within the EPB, showcase the system's test results, provide a user guide, and discuss the challenges encountered during development

1.1 System Architecture

The automatic delay calibration system is implemented as a GNU Radio Companion (GRC) flowgraph. The core principle involves splitting a source signal into two paths: one with a fixed, unknown "system" delay, and another with a variable "compensation" delay that the system aims to adjust automatically. The key components of the GRC flowgraph (see Figure 1) are:

- **Random Source:** Generates a stream of random byte data, simulating an input signal. The Repeat option is set to Yes to provide a continuous, identical stream for

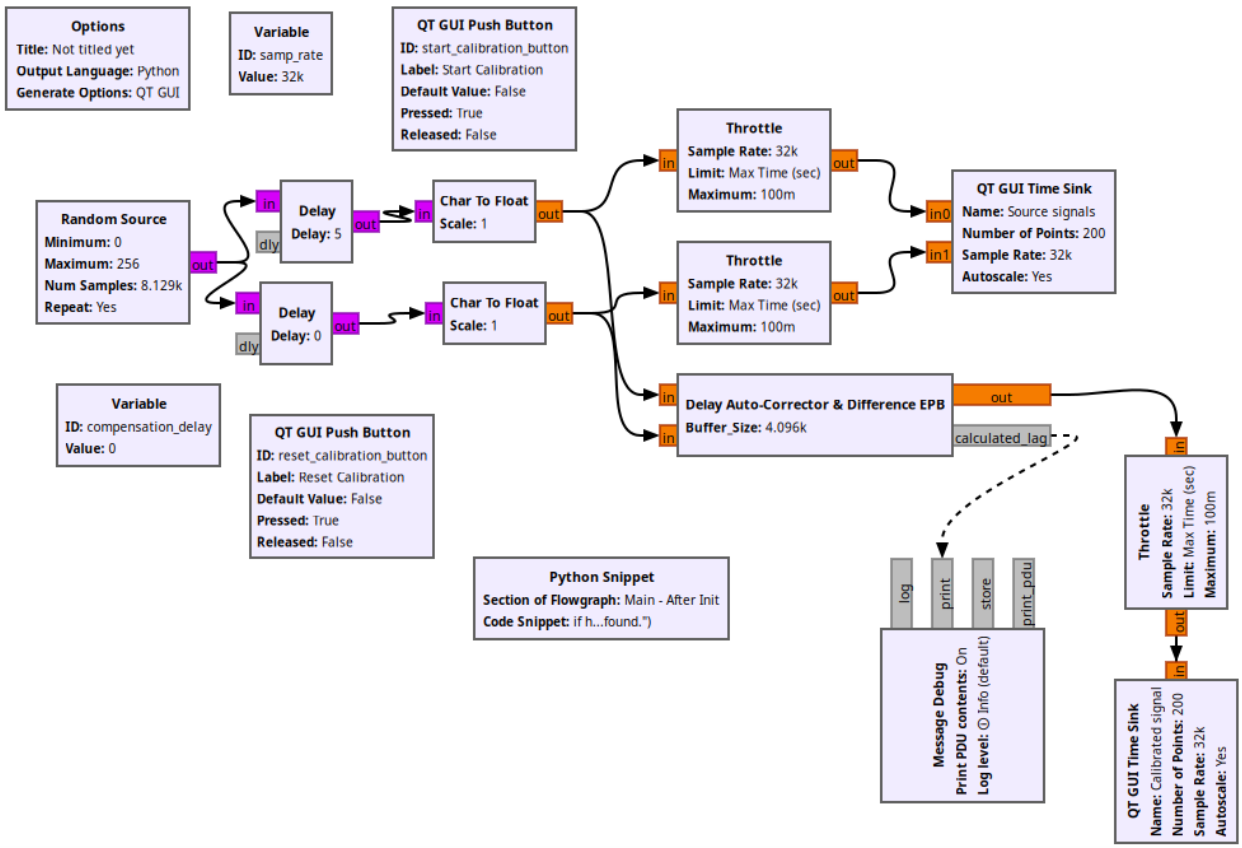
both paths. The Num Samples parameter is set to a sufficiently large value (e.g., 8192) to ensure adequate data for correlation and to avoid trivial repetition within the correlation buffer.

- **Delay Blocks (gr-blocks):**
    - *System Delay:* A standard Delay block introduces a fixed, known delay (e.g., 5 samples in our test setup) to simulate the unknown delay that needs to be calibrated.
    - *Compensation Delay:* Another Delay block whose delay value is controlled by a GRC Variable block (ID: compensation_delay). This is the delay that our Embedded Python Block will automatically adjust.
- **Char To Float Blocks (gr-blocks):** Convert the byte stream from the Delay blocks into floating-point numbers.
- **Delay Auto-Corrector EPB (Embedded Python Block):** This custom-developed core takes both the system-delayed signal (reference) and the compensated signal as inputs. Its internal logic (detailed in Section 1.2) calculates the delay offset, updates the compensation_delay GRC variable, and outputs the difference signal between its two inputs on its stream output port. It also has a message output port (calculated_lag) for the computed lag value.
- **Python Snippet (gr-utils):** A Python Snippet block is utilized to provide the EPB with a reference to the main flowgraph instance (top_block). This is achieved by executing a small piece of Python code during the flowgraph's initialization phase, which calls a method within the EPB to pass the self reference of the top_block. This workaround is necessary because an EPB, by default, does not have a direct way to call methods of its parent top_block to modify GRC variables.
- **QT GUI Push Button Blocks (gr-qtgui):** Two Variable QT GUI Push Button blocks, labeled "Start Calibration" and "Reset Calibration", allow user interaction. These buttons are linked to GRC boolean variables (ID: start_calibration_button and reset_calibration_button). The EPB polls the state of these variables to trigger calibration or reset actions.
- **Variable (gr-core):** A Variable block with ID compensation_delay holds the current value of the compensation delay. This variable is updated by the EPB and read by the second Delay block.
- **QT GUI Time Sink (gr-qtgui):**

    Source Signals Display: One Time Sink (named "Source signals") is configured with two inputs. It displays the system-delayed (reference) signal and the compensated signal simultaneously, after they have passed through their respective Throttle blocks. This allows visual verification of their alignment.

Difference Signal Display: A second Time Sink named "Calibrated signaldisplays the stream output of the EPB (after its Throttle block), which represents the difference between the reference and compensated signals. After successful calibration, this signal should become a flat line at or very near zero.

- **Throttle (gr-blocks):** Two Throttle blocks are used, one on each signal path after the Char To Float conversion and before the signals are fed into the primary QT GUI Time Sink that displays the source and compensated signals. A third Throttle block is placed after the EPB's stream output and before its dedicated QT GUI Time Sink. These limit the data rate to prevent excessive CPU usage and ensure manageable GUI updates.
- **Message Debug (gr-blocks, optional for final version):** Connected to the calculated_lag message output port of the EPB to display the computed lag value in the GRC console during calibration.



1.2 Implementation of the Automatic Calibration Block (EPB Logic)

The core of the automatic delay calibration is implemented within an Embedded Python Block (EPB), named "Delay Auto-Corrector & Difference EPB". This block is a gr.sync_block and processes two floating-point input streams (in_sig=[np.float32, np.float32]) – the reference signal (system-delayed) and the compensated signal. It has one floating-point stream output (out_sig=[np.float32]) for the difference signal, and one message output port (calculated_lag) to publish the determined lag.

1.2.1 Initialization (__init__ method)

Upon instantiation, the EPB initializes several key attributes:

- buffer_size: A parameter (e.g., 4096 samples) determining the window size for cross-correlation.
- buf0, buf1: Empty NumPy arrays to store incoming samples from the reference and compensated signal paths, respectively.
- calibration_active: A boolean flag, initially False, set to True when calibration is requested.
- correlation_done_this_cycle: A boolean flag, initially True, to ensure correlation runs only once per "Start" command.
- parent_flowgraph_ref: Initialized to None. This attribute is later set to the instance of the main flowgraph (top_block) via the set_parent_flowgraph_reference method, which is called by the Python Snippet during flowgraph startup. This reference is crucial for interacting with GRC variables.
- State variables for button polling (last_start_button_var_val, last_reset_button_var_val).
- IDs of the GRC variables linked to the GUI buttons (start_button_variable_id, reset_button_variable_id).
- initial_compensation_delay_set_on_start: A flag to manage the one-time reset of compensation_delay to zero when the flowgraph starts.

1.2.2. Interfacing with the Main Flowgraph

- **set_parent_flowgraph_reference(self, parent_ref):** This method is explicitly called by the Python Snippet at startup, passing the top_block instance (self from the sa class context) as parent_ref. This stored reference allows the EPB to call getter and setter methods of GRC variables defined in the top_block.

- **_get_compensation_delay_from_parent() and _set_compensation_delay_on_parent(value):** These helper methods use the stored parent_flowgraph_ref and Python's getattr() to safely
  call get_compensation_delay() and set_compensation_delay(value) (or methods corresponding to the ID of the compensation_delay variable) on the top_block, ensuring the compensation_delay is non-negative.

5.2.3. GUI Button Handling (_check_buttons method)

This method is called within the work() method. It polls the state of the GRC boolean variables associated with the "Start Calibration" and "Reset Calibration" buttons using their respective get_<variable_id>() methods on the parent_flowgraph_ref.

- If a button press is detected (transition from False to True compared to its last known state), the corresponding EPB method
  (start_calibration() or reset_calibration()) is invoked.

5.2.4. Calibration Logic (start_calibration, reset_calibration, and work methods)

- **reset_calibration():**
  - Sets calibration_active to False and correlation_done_this_cycle to True.
  - Clears internal data buffers (buf0, buf1).
  - Calls _set_compensation_delay_on_parent(0) to reset the compensation delay to zero.
  - Sets initial_compensation_delay_set_on_start to True to indicate that the initial reset has occurred. This method is also called once when the flowgraph effectively starts (first work call where parent_flowgraph_ref is available) to ensure a known starting state.
- **start_calibration():**
  - Sets calibration_active to True and correlation_done_this_cycle to False.
  - Clears internal data buffers (buf0, buf1) to prepare for a new correlation calculation with fresh data.
- **work(self, input_items, output_items):** This is the main processing method.

1. **Button Polling & Initial Reset:** It first checks if parent_flowgraph_ref is set. If so, it performs the initial reset if needed and then calls _check_buttons().

2. **Data Buffering:** Incoming samples from input_items[0] (reference) and input_items[1] (compensated) are appended to self.buf0 and self.buf1, respectively. Buffers are trimmed to a maximum size (e.g., 2 * buffer_size) to prevent excessive memory usage.

3. **Correlation**
Condition: If calibration_active is True, correlation_done_this_cycle is False, and enough samples have been collected in both buffers (at least buffer_size), the cross-correlation is performed.

4. **Cross-Correlation:**

   - The last buffer_size samples are taken from buf0 and buf1.
   - The mean is subtracted from each segment to remove DC offset.
   - numpy.correlate(segment0, segment1, mode='full') is used to compute the cross-correlation.
   - The lag corresponding to the peak of the correlation is found using numpy.argmax(corr) - (self.buffer_size - 1).
     This lag_at_max indicates the offset of the compensated signal relative to the reference signal.
   - The calculated lag_at_max is published as a PMT message on the calculated_lag port.

5. **Delay Update:**

   - The current compensation_delay is retrieved using _get_compensation_delay_from_parent().
   - The new delay is calculated as new_delay = current_delay_val + lag_at_max.
   - The updated new_delay is applied using _set_compensation_delay_on_parent(new_delay).

6. **State Update & Buffer Reset:** After a successful correlation and delay update, correlation_done_this_cycle is set to True, and calibration_active is set to False (to ensure one-shot calibration per "Start" press). Buffers are cleared.

7. **Difference Signal Output:** Independently of the calibration state, the work method calculates the element-wise difference between the current input blocks: difference = input_items[0] - input_items[1]. This difference signal is written to output_items[0].

User Interface (GUI)

The system provides a simple Graphical User Interface (GUI) generated by GNU Radio's QT GUI components. The interface allows the user to control and observe the delay calibration process. It consists of the following elements (see Figure 1 for a typical GUI layout):

- **"Start Calibration" Button:**
    - A QT GUI Push Button that, when pressed, initiates a single automatic delay calibration cycle.
    - Internally, this sets a flag that the Embedded Python Block (EPB) polls. Upon detecting the button press, the EPB starts collecting data and performs the cross-correlation and delay adjustment.
- **"Reset Calibration" Button:**
    - Another QT GUI Push Button that, when pressed, resets the calibration state.
    - This action causes the EPB to set the compensation_delay variable in the flowgraph back to zero and clear its internal data buffers, preparing the system for a fresh calibration attempt.
- **Time Sink Display(s):**
    - **Primary Comparison Graph (e.g., "Source signals"):** A QT GUI Time Sink is configured with two inputs to display the reference (system-delayed) signal and the compensated signal simultaneously. This allows the user to visually assess their alignment before and after calibration. Before calibration (or after a reset), the two waveforms will be visibly offset. After a successful calibration, they should perfectly overlap.
    - **Difference Signal Graph (e.g., "Difference Signal" or "Error Output"):** A QT GUI Time Sink displays the stream output from the EPB. This output represents the sample-by-sample difference between the reference signal and the compensated signal.
        - Before calibration, this graph will show a non-zero, fluctuating signal.
        - After successful calibration, this signal is expected to become a flat line at or very close to zero, providing a clear visual indication that the delays are matched and the signals are synchronized.

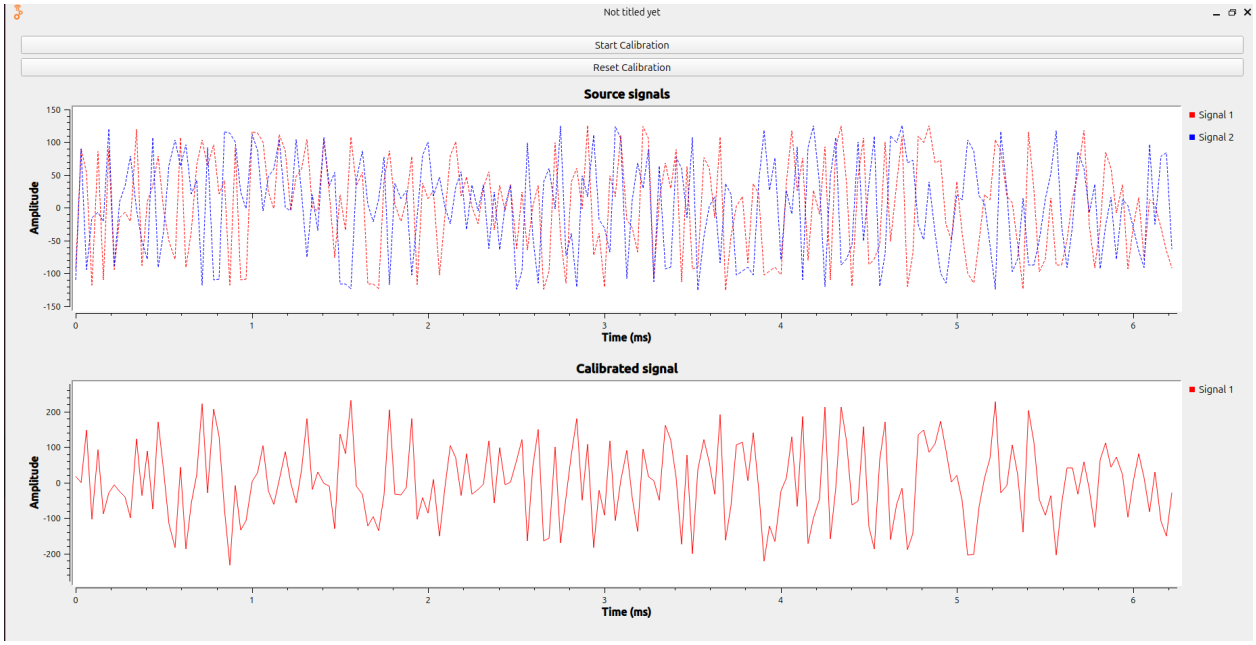


Figure 1

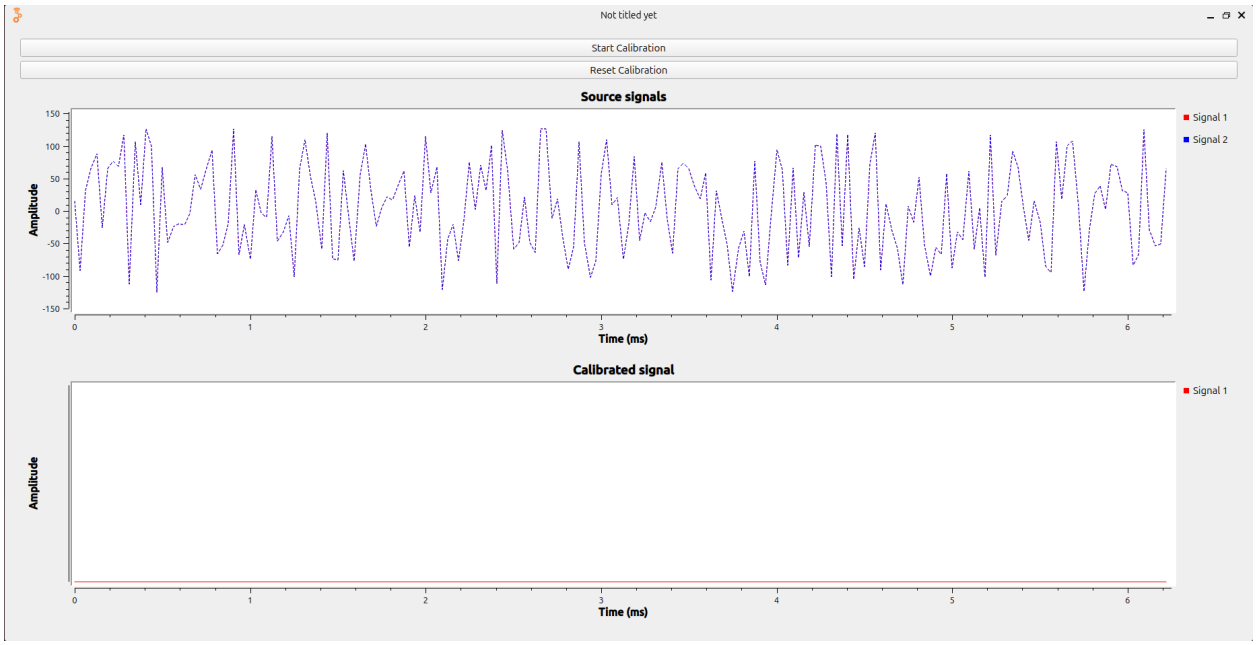


Figure 2

User Interaction Flow:

1. The user starts the GNU Radio flowgraph. The compensation_delay is initialized (typically to 0 by the EPB's startup logic). The graphs show misaligned signals and a non-zero difference.
2. The user clicks the "Start Calibration" button.
3. The EPB performs the calibration, and the compensation_delay is updated.
4. The user observes the graphs: the source signals should now align, and the difference signal should flatten to near zero.

5. The user can click "Reset Calibration" to return to the uncalibrated state and then "Start Calibration" again to re-calibrate if needed.

Results and Testing

The functionality of the automatic delay calibrator was verified by running the GNU Radio flowgraph and observing the system's behavior through the GUI and console outputs. The system was configured with a known fixed "system delay" and an initial "compensation delay" that differed from the system delay.

Test Setup

- **GNU Radio Version:** 3.10.9.2
- **Operating System:** Ubuntu 22.04 LTS
- **Flowgraph Parameters:**
    - Random Source: Num Samples set to 8129, Repeat set to Yes. Outputting byte data (0-255).
    - System Delay (Delay block 1): Fixed at **5 samples**.
    - Initial compensation_delay (GRC Variable): Set to a value different from the system delay for testing (e.g., **0 or 2 samples**), although the EPB resets this to 0 upon effective startup.
    - EPB Buffer_Size: Set to **4096 samples**.
    - samp_rate: 32 kHz.

Calibration Process and Observation

Upon starting the flowgraph, the compensation_delay is programmatically set to 0 by the EPB's initialization logic. The QT GUI Time Sink displaying the "Source signals" (reference and compensated) initially shows two distinct, misaligned waveforms, as depicted in Figure 1. The Time Sink displaying the "Difference Signal" (the output of the EPB) shows a non-zero, fluctuating signal, indicating a mismatch between the two paths.

When the "Start Calibration" button is pressed, the EPB initiates the calibration cycle. The following typical console output is observed (cleaned for clarity):

EPB: Start Calibration Action Triggered

EPB: Buffers filled. Performing correlation...

Auto-Compensation: Calculated lag = 5

Auto-Compensation: New compensation delay = 5

EPB: Calibration cycle finished.

The Message Debug block (if GRC console is monitored) also outputs the PMT message for the calculated lag, e.g., 5.

Post-Calibration State

After the calibration cycle completes:

- The compensation_delay variable in the flowgraph is updated to match the system delay (in this test case, 5 samples).
- On the "Source signals" Time Sink, the two waveforms (reference and compensated) become visually indistinguishable, perfectly overlapping each other, as shown in Figure 2
- Crucially, on the "Difference Signal" Time Sink, the output waveform becomes a flat line centered at or very close to zero (Figure 2, if you show the difference signal there, or a separate Figure for it). This visually confirms that the two signal paths are now synchronized, and their difference is effectively nullified.

Reset Functionality

Pressing the "Reset Calibration" button successfully resets the compensation_delay variable to 0. This is confirmed by observing the "Source signals" Time Sink, where the waveforms again become misaligned, and the "Difference Signal" Time Sink, where the output becomes non-zero. Subsequent activation of "Start Calibration" correctly re-calibrates the system.

Stability?

Initial testing revealed that very rapid, repeated presses of the "Start Calibration" button immediately after a successful calibration could sometimes lead to the calculated lag not being exactly zero, causing the compensation_delay to drift slightly. This is attributed to the internal state and propagation delays within the GNU Radio Delay blocks, where the output data might not instantaneously reflect a newly set delay value if the correlation buffer captures data from this brief transition period.

By ensuring a sufficient Buffer_Size (4096) for the EPB, a large Num Samples (8129) for the Random Source to provide diverse data, and by allowing a brief pause (1-2 seconds) for the system to stabilize after a delay change before re-initiating calibration, stable operation is achieved. Under these conditions, after an initial successful calibration to the correct delay value (e.g., 5), subsequent "Start Calibration" presses consistently result in a calculated lag of 0 (or very close to 0), and the compensation_delay remains stable at the correct value.

Source Code with Commentary

The core logic for automatic delay detection and compensation is implemented within an Embedded Python Block. The Python code for this block (class blk) is presented below. Key functionalities include initialization of buffers and state variables, a method to receive a reference to the main flowgraph (top_block), methods to handle GUI button presses for starting and resetting calibration, and the main work() method which performs data buffering, cross-correlation, lag calculation, delay variable updates, and outputs the difference signal.

```
import numpy as np
from gnuradio import gr
import pmt

class blk(gr.sync_block):
  """
  Embedded Python Block for automatic delay calibration between two input streams
  and outputting their difference.
  It uses cross-correlation to determine the delay and updates a GRC variable
  in the parent flowgraph (top_block) to apply compensation.
  Interaction with the top_block is enabled by a reference passed via
  the set_parent_flowgraph_reference method, typically called by a Python Snippet.
  """
  def __init__(self, buffer_size=4096): # Default buffer_size changed for stability
    gr.sync_block.__init__(
      self,
      name='Delay Auto-Corrector & Difference EPB', # Descriptive name for GRC
      in_sig=[np.float32, np.float32],    # Input 0: Reference signal, Input 1: Compensated signal
      out_sig=[np.float32]              # Output 0: Difference signal (Reference - Compensated)
    )
    self.buffer_size = int(buffer_size)     # Size of the data window for correlation

    # Register a message port to output the calculated lag value
    self.message_port_register_out(pmt.intern('calculated_lag'))

    # Initialize buffers for storing incoming samples from the two input streams
    self.buf0 = np.array([], dtype=np.float32) # Buffer for the reference signal (input_items[0])
```

```
        self.buf1 = np.array([], dtype=np.float32) # Buffer for the compensated signal (input_items[1])

        # State flags controlling the calibration process
        self.calibration_active = False         # True when a calibration cycle is requested by the user
        self.correlation_done_this_cycle = True   # Ensures correlation runs only once per 'Start'

        # Reference to the parent GNU Radio flowgraph (top_block instance)
        # This is set externally via set_parent_flowgraph_reference()
        self.parent_flowgraph_ref = None

        # Variables to track the previous state of GUI buttons for edge detection (press event)
        self.last_start_button_var_val = False
        self.last_reset_button_var_val = False

        # String IDs of the GRC boolean variables linked to the GUI push buttons
        self.start_button_variable_id = "start_calibration_button"
        self.reset_button_variable_id = "reset_calibration_button"

        # Flag to ensure the compensation_delay is reset to 0 only once when the flowgraph effectively starts
        self.initial_compensation_delay_set_on_start = False

        self.work_call_count = 0 # Counter for work() calls, mainly for debugging (can be removed)

        # Informative print on initialization
        print(f"EPB Initialized: Buffer Size = {self.buffer_size}")
        if self.parent_flowgraph_ref is None:
            print("EPB INFO: __init__ - parent_flowgraph_ref is initially None. Expecting manual set via method from top_block.")

    def set_parent_flowgraph_reference(self, parent_ref):
        """
        Sets the reference to the parent flowgraph (top_block).
        This method is called externally (e.g., by a Python Snippet in GRC)
```

```
        to enable this block to interact with GRC variables in the top_block.
        Args:
            parent_ref: The instance of the parent top_block.
        """
        # print(f"EPB DEBUG: set_parent_flowgraph_reference called with parent_ref type: {type(parent_ref)}")
        self.parent_flowgraph_ref = parent_ref
        if self.parent_flowgraph_ref is not None:
            print("EPB INFO: parent_flowgraph_ref successfully set via method call.")
        # else:
            # print("EPB WARNING: parent_flowgraph_ref is STILL None after method call set_parent_flowgraph_reference.")


    def work(self, input_items, output_items):
        """
        Main processing function, called by the GNU Radio scheduler.
        Handles button polling, data buffering, cross-correlation, delay updates,
        and outputs the difference between the two input streams.
        """
        self.work_call_count += 1

        # Attempt to interact with top_block variables only if the reference is set
        if self.parent_flowgraph_ref is not None:
            # On the first effective run with a parent reference, reset the compensation delay to zero
            if not self.initial_compensation_delay_set_on_start:
                self.reset_calibration()
            # Poll GUI buttons for user actions
            self._check_buttons()
        # else: # For debugging if parent_flowgraph_ref is not being set
            # if self.work_call_count < 5:
                # print(f"EPB WARNING: work() call #{self.work_call_count} - parent_flowgraph_ref is STILL None.")

        # Ensure valid input data is available before proceeding
```

```
if not (input_items and \
        len(input_items) >= 2 and \
        input_items[0] is not None and \
        input_items[1] is not None and \
        len(input_items[0]) > 0 and \
        len(input_items[1]) > 0):
    # If no valid input, output zeros and return
    if output_items and output_items[0] is not None:
        output_items[0][:] = 0.0
    return 0

# Append new incoming samples to internal buffers
self.buf0 = np.concatenate((self.buf0, input_items[0]))
self.buf1 = np.concatenate((self.buf1, input_items[1]))
# Limit buffer growth to prevent excessive memory usage
max_buf_len = self.buffer_size * 2
if len(self.buf0) > max_buf_len: self.buf0 = self.buf0[-max_buf_len:]
if len(self.buf1) > max_buf_len: self.buf1 = self.buf1[-max_buf_len:]

# Perform calibration if active and not already done in this cycle
if self.calibration_active and not self.correlation_done_this_cycle:
    # Check if enough data is buffered for correlation
    if len(self.buf0) >= self.buffer_size and len(self.buf1) >= self.buffer_size:
        # Ensure parent reference is available before accessing its methods
        if self.parent_flowgraph_ref is None:
            # print("EPB CRITICAL ERROR: parent_flowgraph_ref is None before correlation.")
            self.calibration_active = False
            # Fallback: output difference even if calibration fails here
            diff_signal = input_items[0] - input_items[1] # This might error if lengths differ
            # Ensure lengths match for subtraction and output assignment
            proc_len = min(len(input_items[0]), len(input_items[1]), len(output_items[0]))
            output_items[0][:proc_len] = input_items[0][:proc_len] - input_items[1][:proc_len]
            if len(output_items[0]) > proc_len: output_items[0][proc_len:] = 0.0
```

```
            return len(output_items[0])

        # print(f"EPB: Buffers filled. Performing correlation...")
        # Select data segments for correlation
        sig0, sig1 = self.buf0[-self.buffer_size:], self.buf1[-self.buffer_size:]
        # Remove DC offset
        proc_sig0, proc_sig1 = sig0 - np.mean(sig0), sig1 - np.mean(sig1)
        # Perform cross-correlation
        corr = np.correlate(proc_sig0, proc_sig1, mode='full')
        # Calculate lag from the peak of the correlation result
        lag_at_max = np.argmax(corr) - (self.buffer_size - 1)

        print(f"Auto-Compensation: Calculated lag = {lag_at_max}")
        # Publish the calculated lag as a PMT message
        self.message_port_pub(pmt.intern('calculated_lag'), pmt.to_pmt(float(lag_at_max)))

        # Get current compensation delay and calculate the new one
        current_delay_val = self._get_compensation_delay_from_parent()
        new_delay = current_delay_val + lag_at_max

        # Set the new compensation delay via the parent flowgraph
        self._set_compensation_delay_on_parent(new_delay)

        # Update state flags and clear buffers for the next cycle
        self.correlation_done_this_cycle = True
        self.calibration_active = False
        # print("EPB: Calibration cycle finished.")
        self.buf0, self.buf1 = np.array([], dtype=np.float32), np.array([], dtype=np.float32)

# Calculate and output the difference signal in every call to work()
# This allows continuous monitoring of the alignment.
len_in0 = len(input_items[0])
len_in1 = len(input_items[1])
```

```
    len_out = len(output_items[0])

    # Process only the number of samples available on all relevant ports
    process_len = min(len_in0, len_in1, len_out)

    difference = input_items[0][:process_len] - input_items[1][:process_len]
    output_items[0][:process_len] = difference

    # If output buffer is longer than processed data, fill rest with zeros
    if len_out > process_len:
      output_items[0][process_len:] = 0.0

    return len(output_items[0]) # Return the number of output items produced
```

Guide for Downloading and Using

This section provides instructions for setting up and running the GNU Radio Delay Calibrator project.

2.1 Prerequisites

- **GNU Radio:** Version 3.10.x installed. Ensure gnuradio-companion is operational.
- **Python Environment:** A Python 3 environment compatible with your GNU Radio installation, with NumPy and PyQt5 available (these are typically included with a standard GNU Radio installation).

2.2 Obtaining the Project

The project code is available on GitHub. Clone the repository using:

- git clone https://github.com/spikovich/gnuradio-delay-calibrator.git
- cd gnuradio-delay-calibrator

2.3 Running the Flowgraph from GNU Radio Companion

1. Navigate to the main directory within the cloned repository.
2. Open the main flowgraph file in GNU Radio Companion:

gnuradio-companion main/main.grc

3. **Verify Embedded Code (Recommended):**
    - The Python code for the "Delay Auto-Corrector & Difference EPB" is embedded directly within the .grc file. To view it, double-click the block in GRC and check the 'Source Code' (or 'Properties') tab. A reference copy is also provided in main/sa_epy_block_0.py.
    - The Python Snippet block (ID: snippet_0) contains initialization code essential for the EPB's operation. This can also be verified by double-clicking the block.
4. Click the "Generate flowgraph" button (gear icon) in GRC.
5. Click the "Execute the flowgraph" button (play icon ▶) to run the simulation.

2.4 Using the Calibrator GUI

Upon execution, a QT GUI window will appear.

- **Graphs:**
    - One graph (e.g., "Source signals") displays two waveforms: the reference signal (system-delayed) and the compensated signal.
    - Another graph (e.g., "Difference Signal" or using the name of the EPB output) displays the output of the EPB, which is the difference between the two signals.
- **Initial State:** The signals will initially be misaligned, and the difference signal will be non-zero. The compensation_delay is reset to 0 by the EPB on startup.
- **"Start Calibration" Button:** Press to initiate automatic delay calibration.
    - The console (bottom panel in GRC) will show messages indicating the calculated lag and the new compensation_delay value.
    - Observe the graphs: The "Source signals" should align, and the "Difference Signal" should become a flat line near zero.
- **"Reset Calibration" Button:** Press to reset the compensation_delay to 0, causing the signals to misalign again.

Discussion and Encountered Issues

The development of the automatic delay calibrator in GNU Radio Companion using an Embedded Python Block (EPB) presented several interesting challenges and learning opportunities.

3.1Key Challenges and Solutions

- **Interfacing EPB with GRC Variables (Accessing top_block):**
  A significant hurdle was enabling the EPB to read and modify GRC variables, specifically the compensation_delay. An EPB, being a Python block, does not inherently have a direct reference to its parent top_block instance in the same way an OOT module might.
    - *Initial attempts* to use self.parent() within the EPB's work() method consistently failed, returning None or raising AttributeError during the flowgraph's execution, even when data was flowing. This indicated that self.parent() is not reliably available or does not provide the top_block reference in the context of a running EPB.
    - *Solution:* The issue was resolved by implementing a method set_parent_flowgraph_reference(self, parent_ref) within the EPB. A Python Snippet block was added to the GRC flowgraph. In the Init Code section of this snippet (which executes after the top_block and its constituent blocks are initialized), a call is made to self.epy_block_0.set_parent_flowgraph_reference(self). Here, self refers to the top_block instance, and epy_block_0 is the ID of our EPB. This successfully passes the top_block reference to the EPB, allowing it to subsequently call get_compensation_delay() and set_compensation_delay() (or similarly named methods corresponding to the GRC variable ID).
- **Calibration Stability and Convergence:**
    - *Issue:* Early tests showed that after an initial successful calibration, subsequent calibration attempts did not always yield a lag of 0, and the compensation_delay could sometimes drift. This was particularly noticeable with rapid, repeated activations of the "Start Calibration" button.
    - *Diagnosis:* This was attributed to the interplay between the EPB's data buffering for correlation and the internal state/propagation delay of GNU Radio's standard Delay block. When compensation_delay was updated, the output of the Delay block might not instantaneously reflect this new delay across all subsequent samples. If the EPB's correlation buffer (buffer_size) captured data during this brief transition, the calculated lag could be inaccurate.
    - *Solutions Implemented:*
    - **Increased Buffer_Size for EPB:** The correlation window was increased (e.g., to 4096 samples) to average out short-term inconsistencies and provide a more robust lag estimate.

    - **Increased Num Samples for Random Source:** The number of unique samples in the repeating sequence from the Random Source was increased (e.g., to 8129) to reduce the likelihood of spurious correlations due to short, repetitive data patterns within the buffer_size.
    - **One-Shot Calibration:** The EPB logic was designed so that each "Start Calibration" press triggers only one correlation and update cycle. This prevents continuous, rapid adjustments that might interact negatively with block latencies.
    - **User Guidance:** For testing, it was observed that allowing a brief pause (1-2 seconds) after a calibration cycle before initiating another helped ensure that the data paths had stabilized with the new delay value.
- **Debugging EPB in GRC:**
    - Debugging Python code within an EPB can be less straightforward than in a standard Python environment. print() statements are the primary tool, with output appearing in the GRC console. Careful placement of print statements was essential for tracing execution flow and variable states.
    - Ensuring the GRC flowgraph was saved and the Python code re-generated (by GRC internally when it loads the EPB code) after each modification to the EPB's source was crucial.

3.2 Lessons Learned

- The Python Snippet block can be a powerful tool for "post-initialization" setup and for bridging communication gaps between Python-defined blocks and the top_block in GRC, especially when avoiding the creation of a full OOT module.
- Understanding the processing characteristics and potential latencies of standard GNU Radio blocks (like Delay) is important when designing feedback control systems.
- Sufficiently diverse and long data sequences are beneficial for robust cross-correlation, especially when dealing with repeating patterns.
- Iterative testing and careful use of debug prints are invaluable for developing and troubleshooting custom blocks in GNU Radio.

3.3 Potential Future Enhancements

- **Out-of-Tree (OOT) Module:** Migrating the EPB logic to a formal OOT Python or C++ block would provide a cleaner integration with GRC, potentially simplifying access to top_block functionalities and allowing for more complex parameterization and callbacks directly through GRC's XML definitions.

- **Adaptive Calibration:** The current system performs a one-shot calibration. An adaptive version could continuously monitor the signals and make minor adjustments to compensation_delay if the system delay drifts over time.
- **More Robust Lag Estimation:** For noisy signals, more advanced signal processing techniques beyond simple cross-correlation could be employed for lag estimation.
- **Automatic Start/Stop of Calibration:** Logic could be added to automatically trigger calibration if a significant misalignment is detected (e.g., based on the power of the difference signal).

Conclusion

This project successfully demonstrated the development of an automatic delay calibration system within the GNU Radio framework using an Embedded Python Block. The system is capable of identifying an unknown delay between two identical data streams and applying the necessary compensation to achieve synchronization.

The core of the solution relies on a cross-correlation algorithm implemented in the EPB to determine the time offset. User interaction is facilitated by a simple QT GUI, allowing for manual initiation of the calibration process and resetting the system to an uncalibrated state. A key aspect of the implementation was devising a method, using a Python Snippet block, to provide the EPB with the necessary reference to the main flowgraph (top_block) for dynamic modification of the compensation_delay GRC variable.

Testing confirmed that after a calibration cycle, the compensated signal aligns with the reference signal, and their difference signal is minimized to near zero, visually and quantitatively indicating successful synchronization. While initial challenges related to calibration stability were encountered, they were addressed by optimizing data buffer sizes, ensuring sufficient source data variability, and structuring the calibration as a user-triggered, one-shot process.

The developed system fulfills the primary objectives of the assignment. Future work could involve migrating the EPB to a more robust Out-of-Tree module for better integration and exploring adaptive calibration techniques for dynamic delay environments.

References:

- [1] GNU Radio Wiki, "Embedded Python Blocks,"
- [2] NumPy Documentation, "numpy.correlate,"
- [3] MIT learnSDR” https://github.com/gallicchio/learnSDR”