Error Signal Regulation User Manual

Austin Czyzewski

October 2020

Contents

Instruments:

- HP 8648[C] Signal Generator
- Tektronix 3014 Series Oscilloscope
- GPIB connection to PC

Software:

- python – pyvisa; pyvisa-py
- GPIB_FUNCS.py
- Tag_Database.py
- Master.py
- Error_Signal_Indusoft_Regulation.py
- National Instruments NI-VISA driver installed

1 Introduction

The purpose of this program is to automate the task of regulating the error signal output on the signal generator. This is important as this has a large impact on the phase locking loop that we use to accelerate our electron beam. This task was formerly done by hand, and required the constant attention of an operator to maintain this regulation. One of the major deficiencies to this, is that the operator must focus on the knob rather than more important system diagnostics. This GUI was designed to both reduce the amount of effort needed for it to operate, and to also more quickly adapt it to our needs as our systems mature.

This program automatically regulates the error signal by changing the frequency of the signal generator, and it controls the measurements of an attached oscilloscope. The oscilloscope is required to detect the voltage output. When in regulation, the human user is locked out of the signal generator. However, they are not locked out of the oscilloscope, as the other channels may need to be altered. As a consequence of this, if the measurement on the oscilloscope fails, there is a switch in place to reset it and restore the settings required. This will momentarily pause the regulation loop.

This document will explain the following: The purpose of each button, what it does, and how to identify what state the regulation loop is in. We will also dive into the common edits to be made to the code.

2 Setup

This section will outline the necessary steps to ensure that the regulation GUI will open and run. The GUI is designed so that if there is an error detected the program will fail to open.

2.1 Hardware Connections

To set up the communications between our three instruments we need to set up our GPIB cables in the following configuration.

First: Plug the GPIB Cable in connecting the Signal Generator and the Oscilloscope. Note that the oscilloscope must have the I/O card installed in the back of it.

Second: Plug the GPIB to USB cable into the back of either end of the GPIB cable, but the signal generator is preferred.

Third: Plug the USB end of the GPIB to USB cable into a USB port in the computer and make sure that the light in that same cable turns on for the GPIB side.

2.2 GPIB Addresses

Once the hardware is set up, next comes making sure that the computer knows what to look for. Each device will have it's own GPIB address and they cannot be the same. To check these addresses do the following: locate the ADRS button on the front panel of the signal generator in the lower lefthand corner, press it and the address will come onto the screen. For the oscilloscope press the Utility button in the top right corner of the device. Navigate to the I/O section of the system configuration tab in the lower lefthand corner. Once there, GPIB Talk/Listen should be an option. Press this and check the number in the top right corner of the oscilloscope screen. To change this, use the topmost knob on the oscilloscope.

2.3 Problems with detection

When setting the system up the following common problems and their solutions may come up.

First: If the oscilloscope does not have a GPIB Talk/Listen button this means that the card is not detected. Make sure that the I/O card is fully pressed in. Once done, the oscilloscope will need to be power cycled.

Second: If the light does not come on in the GPIB to USB cable this means that the cable is not receiving any power, try another USB port. If this does not work, the cable may be flawed.

3 Button overview

There are currently 3 buttons and one text box in Indusoft for this script; each subsection here will explain what each of these do and what active state they are in.

3.1 Error Regulation

When this button is activated, the background of the button will become green. This button begins the active regulation of the error signal based on the parameters in this code. This button updates the ”Running” tag to True, which will then execute the regulation script. At the beginning of each loop, it will check if this tag is still set to True. In the case that any of internal script interlocks are tripped, this tag is written to false.

3.2 Reset Oscilloscope

This button, because it can be pressed at any time will have a different effect based on the current mode. If this button is pressed during the active regulation of the error signal, the regulation loop will temporarily stop while the oscilloscope resets. If this button is pressed while the regulation is off, the measurements on the oscilloscope will reset and regulation cannot resume until it is finished (this is very quick). The purpose of this button is to allow an operator looking at the oscilloscope to examine other channels without the fear of breaking the measurement. In this case, either before or during regulation, press this button to reset the measurements.

3.3 Reset Setpoint

The Reset Setpoint button exists in the case that an operator wants to run the error regulation off of 0. This grabs the current value that is being read, and sets this as the new regulation set point. This value will appear in the text box below in mV.

3.4 Text Box

Similar to Reset Setpoint, this button changes where the regulation setpoint is. This box has the purpose of displaying the current setting and allowing the operator to change it to a specific setting. The units in this box are mV.

4 Active modes of regulation

There are three states that the program can be in, not including the temporary states of start-up and resetting the oscilloscope. In both active runs, each time a regulation step is taken, the code checks to see if the measurement is available. If the measurement is not available, there will be no regulation.

4.1 Regulating: CW

In this mode, the loop operates following this principle: If the mean measurement for the IF signal channel exceeds a user-defined value (current default of 2.5 mV), then walk the frequency up or down accordingly.

4.2 Regulating: Pulsing

In this mode, the loop operates following this principle: If the vertical cursor measurement for the IF signal channel at the ramp-down of the pulse exceeds a user-defined value (current default of 0.5 mV), then walk the frequency up or down. The script identifies the pulsing status by reading the pulsing_status tag in the PLC.

4.3 Off

In this state, the script will continuously monitor the status of the Error signal regulation tag from the PLC.

5 User-defined variables

This section contains common changes that may need to be made by an operator. For Error handling, this is not an all inclusive list, but should help with common issues.

5.1 Error handling

- **File ”Regulation_GUI.py”, line 8, in ⟨ module ⟩** This message is raised because there is a problem connecting to the signal generator. Some of the common fixes could be this: make sure that the signal generator is on, the GPIB cable has a firm connection in the back, and any indicator lights on it show that it is connected properly. If this is the case, press the ADRS button on the front panel of the signal generator to ensure that the 2 digit address output there is the same as that contained in between the '::'s in line 8 of the code. If these numbers are not the same change the number in the code to match. If you are still having issues, assess how many devices should be plugged into the GPIB cable, and then look at the first output of the code, this output is the ID of all of the detected devices in the interface bus.
- **File ”Regulation_GUI.py”, line 9, in ⟨ module ⟩**This message is raised because there is a problem connecting to the oscilloscope, similar to the probelm above, please make sure there is a solid connection and that the oscilloscope is powered on. To check the 2 digit address of this device, hit the Utility button (top right corner) of the oscilloscope, then navigate to the I/O section of the System config tab (bottom left corner). Then press the GPIB Talk/Listen button and identify that the 2 digit number is the same in the code as the top right corner of the screen labeled, ”Talk/Listen Address.” Follow similar steps of seeing if the device is detected as above.
- **ModuleNotFoundError: No module named ⟨ module ⟩** This means that the python configuration for the computer does not contain the proper python packages, please refer to the standard python package installation guidelines.

5.2 Regulation parameters

- IF_Channel = 3; This is the channel that the IF signal is going into on the oscilloscope. This may be the most changed tag of them all. So much so that future versions may include this as a type in box.
- Trigger_Channel = 4; This is the channel that the Pulse output is put into. This is necessary for the pulsing section of the code.
- Trigger_Level = 20 /1000 mv; This is the trigger level of trigger channel, change if the triggering is too high or low.
- Measurement = 3; This is the measurement channel. If one were to press the Measure button (top left button in the set of buttons in the top right of the oscilloscope) they would see the measurement settings. This gives you the option to select the measurement channel as to not overwrite any important measurements that may be active.
- Step_size = 40 (Hz); This is the CW step size, in Hz. This is the amount that the frequency is changed with each regulation step. Smaller means that the regulation is slower, but more precise. This may need to change with the Wait tags below.
- Pulse_Step_Size = 10 (Hz); Similar to Step_size, this is the step size while in the pulsing regime.

- Max_Threshold = 10000 (Hz); This is the maximum that we are going to allow the regulation to walk the frequency from the start point. Once reached, the GUI will interlock. This is to alert the operator to make any necessary changes. If this keeps tripping the system, this tag can be changed, or alternatively, once the pressure of the cavity stabilizes, the program could start from scratch.
- Walk_Threshold = 2.5 (mV); This is the threshold that the error signal must exceed in order to begin regulation. The minimum is currently 1.5 mV. If this value were too small, regulation would continue whilst power is off.
- Pulse_Walk_Threshold = 0.5 (mV); This is similar; however, does not have the same minimum because we want tighter regulation while pulsing. For that reason, this must be more actively monitored while pulsing (i.e. when tripping off the regulation should be shut off. Safety features soon to come to prevent this).
- Wait_after_step = 0.0400 Seconds; This is the amount of time the computer waits after a regulation step is taken. This is in addition to the next listed variable.
- Wait_between_reads = 0.0100 Seconds; This is the amount of time between each scan. This is chosen to reduce load on the PC.
- Interlock_Threshold_mv = 30 mv; This is the deviation from the regulation set-point before regulation trips off. This is to protect from frequency chasing after a pressure spike.
- Long = False; leave this be. This is for error handling, for different types of read backs that depends on the exact model and configuration of each oscilloscope. This is here for troubleshooting.
- Loops_Debounce = 1; This is the amount of loops in a row before regulation begins, the higher the number, the longer delay before regulation begins.

6 Regulation Code

6.1 Repeated terms

- GPIB.cursor_vbar_read_mv(OS); this is the function that reads the current value of the cursor bar on the oscilloscope
- GPIB.read_mv; this is the function that reads the value of the measurement set up for CW operation.
- M.Write(Client, Tag, Tag_Value, Bool); writes to the Client (PLC) at the Tag location, the defined value. Bool defines if this is boolean or not.
- M.Read(Client, Tag_Value, Bool); Reads the tag from the PLC, Bool is True if a Boolean tag.
- OS.query("MEASU:Meas:State?".format(Measurement))[-2]; this is a command that checks if the Measurement channel measurement is still active
- GPIB.write_frequency(Device, Frequency); this writes the Frequency to the Device.
- SG.control_ren(6); this sets the control status of the signal generator back to the operator. This term is very important due to the fact that if this is pushed, there likely needs to be rapid operator response on the signal generator.

6.2 Python Script

```python
'''
Author: Austin Czyzewski
Last Edit: 10/16/2020
    Edit notes: Deactivate the total walk threshold by changing bounds. Did connection test.

Purpose: IF Signal regulation automation.
'''


import GPIB_FUNCS as GPIB #Importing our GPIB communication functions for easier comprehension and use
import pyvisa #import GPIB communication module
import time #imports time to sleep program temporarily
import Master as M
import numpy as np
import Tag_Database as Tags
from datetime import datetime
import os

def flasher(times):
    for _ in range(times):
        os.system('color DF')
        time.sleep(0.2)
        os.system('color 0F')

Client = M.Make_Client('10.50.0.10')

Pulsing_Tag = Tags.Pulsing_Output #Assign Modbus address here
Running_Tag = Tags.Error_Signal_Regulation #Assign Modbus address here
Reset_Tag = Tags.Oscope_Reset #Assign Modbus address here
Regulation_Setpoint_Tag = Tags.Regulation_Setpoint_Reset #Assign Modbus address here
Regulation_Entry_Tag = Tags.Regulation_Float #The input tag for this guy

RM = pyvisa.ResourceManager() #pyVISA device manager
Resources = RM.list_resources() #Printing out all detected device IDs
print(Resources)
try:
    SG = RM.open_resource(Resources[0]) #Opening the Signal generator as an object
    OS = RM.open_resource(Resources[1]) #Opening the oscilloscope as an object

    Start_Freq = float(SG.query("FREQ:CW?"))
    print("Starting Frequency of Signal Generator: {} Hz".format(Start_Freq))
except:
    SG = RM.open_resource(Resources[1]) #Opening the Signal generator as an object
    OS = RM.open_resource(Resources[0]) #Opening the oscilloscope as an object

    Start_Freq = float(SG.query("FREQ:CW?"))
    print("Starting Frequency of Signal Generator: {} Hz".format(Start_Freq))

IF_Channel = 3 #The channel that the error signal is on
Trigger_Channel = 4 #The channel which shows the SRF pulse
Trigger_Level = 20   /1000 #mv #The level of the pulse trigger
Read_Start_Voltage = True

Measurement = 2 #Measurement channel
Step_size = 30 #(Hz) Change in frequency with each regulation step
Pulse_Step_Size = 20 #(Hz) Change in frequency with each regulation step when pulsing
Max_Threshold = 100000000 #(Hz) Total amount of frequency change before automatically tripping off program ## 1e8 is effectively deactivating this interlock
Walk_Threshold = 0.5 #(mV) Deviation from 0 the error signal needs to be before CW regulation kicks in
Pulse_Walk_Threshold = 0.5 #(mV) Deviation from 0 the error signal needs before pulsing regulation kicks in
Wait_after_step = 0.0400 #Seconds, the time waited after a step is taken, necessary to allow oscope measurements to change
Wait_between_reads = 0.0100 #Seconds, currently not used, supplemented by GUI time between reads
Interlock_Threshold_mv = 40 #mv, this is the amount of deviation before regulation trips off
```

```python

Loops_Debounce = 1
Trip_Debounce = 3
Long = False #The form that our measurement is output from the o-scope, depending on the way it is set up this can be in either a short or long form
## additional tweak in testing 200417

Error_signal_offset = 0 # (mV) want to pulse off zero
M.Write(Client, Regulation_Entry_Tag, Error_signal_offset)
reset_on_start = False #This tag is in case we want to reset the entire oscilloscope on startup

GPIB.measurement_setup(OS,IF_Channel, measurement = Measurement) #Setting up the required measurement for regulation

# These reset the oscilloscope on startup, only the one above is needed.
#GPIB.channel_settings_check(OS, IF_Channel) #Setting up the vertical and horizontal settings for the error signal
#GPIB.trigger_settings_set(OS, Trigger_Channel, Trigger_Level) #Sets up the vertical settings for trigger channel and trigger parameters
#GPIB.vertical_marker_pulsing(OS, IF_Channel) #Sets up vertical cursor bars to read edge of pulse

Ups = 0 #number of steps taken up before taking one down
Downs = 0 #visa versa
i = 0 #total number of iterative loops gone through, only present to show differences in command line readouts
ups_debounce_counter = 0
downs_debounce_counter = 0

try: #Quick test to determine short or long form oscilloscope output
    short_test = float(OS.query("MEASU:MEAS{}:VAL?".format(Measurement)))
    if Read_Start_Voltage == True:
        Error_signal_offset = short_test
    pass
except:
    long_test = float(OS.query("MEASU:MEAS{}:VAL?".format(Measurement)).split(' ')[1].strip("\n"))
    Long = True
    if Read_Start_Voltage == True:
        Error_signal_offset = long_test
    pass

#print statement
print("\n\n\n")
print("-" * 60)
print("Beginning modulation")
print("-" * 60)
print("\n\n\n")

reset = False
regulation_setpoint_change = False
pulsing = False

########################################
# Reset button loop
########################################
while True:

    reset = M.Read(Client, Reset_Tag, Bool = True)
    running = M.Read(Client, Running_Tag, Bool = True)
    pulsing = M.Read(Client, Pulsing_Tag, Bool = True)
    regulation_setpoint_change = M.Read(Client, Regulation_Setpoint_Tag, Bool = True)
    Error_signal_offset = M.Read(Client, Regulation_Entry_Tag)
    #print(Error_signal_offset)
    i += 1

    if reset: #Checks the reset parameter and runs if True
```

```python
        print("-"*60 + "\n\n\nResetting Oscilloscope\n\n\n" + "-"*60) #print visual break to indicate reset

        GPIB.measurement_setup(OS,IF_Channel, measurement = Measurement) #Same as beginning parameters above
        GPIB.channel_settings_check(OS, IF_Channel)
        GPIB.trigger_settings_set(OS, Trigger_Channel, Trigger_Level)
        GPIB.vertical_marker_pulsing(OS, IF_Channel)
        os.system('cls')

        M.Write(Client, Reset_Tag, False, Bool = True)


    #########################################
    # Checking to see if we want to update regulation setpoint
    #########################################
    if regulation_setpoint_change:

        if pulsing:
            Error_signal_offset = GPIB.cursor_vbar_read_mv(OS)
            print("New regulation setpoint: {:.3f} mV".format(Error_signal_offset))
        else:
            Error_signal_offset = GPIB.read_mv(OS, long = Long, measurement = Measurement)
            print("New regulation setpoint: {:.3f} mV".format(Error_signal_offset))

        if Error_signal_offset > 10000:
            print("Cannot reset Error signal offset with measurement off")
            print("New regulation setpoint: 0 mV")
            Error_signal_offset = 0

        M.Write(Client, Regulation_Setpoint_Tag, False, Bool = True)
        M.Write(Client, float(Regulation_Entry_Tag), Error_signal_offset)
    #########################################
    # Checking for the regulation loop if on
    #########################################

    if running:  # running parameter, if True, runs this loop
        #print("Step 3")
        #########################################
        # Loop for pulsing operation
        #########################################

        if pulsing: #Checks pulsing tag and runs this loop if true

            read_value = GPIB.cursor_vbar_read_mv(OS) #Takes the current value of oscilloscope vbar

            if read_value > (Error_signal_offset + Pulse_Walk_Threshold): #Checks to see if that value is outside of threshold
                Ups += 1
                Downs = 0
                downs_debounce_counter = 0
                if Ups > Loops_Debounce: #Effective debounce
                    temp_freq = GPIB.freq(SG) #Gathers the current frequency
                    GPIB.write_frequency(SG, (temp_freq + Pulse_Step_Size),"HZ") #Writes calculated frequency to the signal generator
                    print("Raised Frequency ", i) #Shows that we took a step in frequency
                    if (temp_freq + Pulse_Step_Size) > (Start_Freq + Max_Threshold): #Sees if the new frequency is outside of bounds
                        print("Error: Broken on too many steps upward")
                        M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                        flasher(5)
                        SG.control_ren(6) #Returns to local control
                    if OS.query("MEASU:Meas{}:State?".format(Measurement))[-2] != str(1): # Sees if the measurement is still active
                        print("Error: Measurement Off")
                        M.Write(Client, Running_Tag, False, Bool = True) #Breaks running
```

```
loop on interlock
                    flasher(5)
                    SG.control_ren(6)
                if read_value > (Error_signal_offset + Interlock_Threshold_mv):
                    print("Error: Deviation too far; read at {:.3e} mV".format(read_value))
                    ups_debounce_counter += 1
                    downs_debounce_counter = 0
                    if ups_debounce_counter > Trip_Debounce:
                        print("Too many reads above threshold, regulation off")
                        M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                        flasher(5)
                        SG.control_ren(6)
                        ups_debounce_counter = 0
                        downs_debounce_counter = 0
                time.sleep(Wait_after_step) #Sleep for after step debounce time

        ########################################
        # Repeat above loop but below the threshold instead of above
        ########################################


        if read_value < (Error_signal_offset - Pulse_Walk_Threshold):
            Downs += 1
            Ups = 0
            ups_debounce_counter = 0
            if Downs > Loops_Debounce:
                temp_freq = GPIB.freq(SG)
                GPIB.write_frequency(SG, (temp_freq - Pulse_Step_Size),"HZ")
                print("Lowered Frequency ", i)
                if (temp_freq - Pulse_Step_Size) < (Start_Freq - Max_Threshold):
                    print("Broken on too many steps downward")
                    M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                    flasher(5)
                    SG.control_ren(6)
                if OS.query("MEASU:Meas{}:State?".format(Measurement))[-2] != str(1):
                    print("Measurement Off")
                    M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                    flasher(5)
                    SG.control_ren(6)
                if read_value < (Error_signal_offset - Interlock_Threshold_mv):
                    print("Error: Deviation too far; read at {:.3e} mV".format(read_value))
                    ups_debounce_counter = 0
                    downs_debounce_counter += 1
                    if downs_debounce_counter > Trip_Debounce:
                        print("Too many reads below threshold, regulation off")
                        M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                        flasher(5)
                        SG.control_ren(6)
                        ups_debounce_counter = 0
                        downs_debounce_counter = 0
                #time.sleep(Wait_after_step)

        #time.sleep(Wait_between_reads)

    ########################################
    # Loop for CW operation, use same logic
    ########################################

    else:
        read_value = GPIB.read_mv(OS, long = Long, measurement = Measurement)

        if read_value > (Error_signal_offset + Walk_Threshold):
```

```
                Ups += 1
                Downs = 0
                downs_debounce_counter = 0
                if Ups > Loops_Debounce:
                    temp_freq = GPIB.freq(SG)
                    GPIB.write_frequency(SG, (temp_freq + Step_size),"HZ")
                    print("Raised Frequency ", i)
                    if (temp_freq + Step_size) > (Start_Freq + Max_Threshold):
                        print("Broken on too many steps upward")
                        M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                        flasher(5)
                        SG.control_ren(6)
                    if OS.query("MEASU:Meas{}:State?".format(Measurement))[-2] != str(1):
                        print("Measurement Off")
                        M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                        flasher(5)
                        SG.control_ren(6)
                    if read_value > (Error_signal_offset + Interlock_Threshold_mv):
                        print("Error: Deviation too far; read at {:.3e} mV".format(read_value))
                        ups_debounce_counter += 1
                        downs_debounce_counter = 0
                        if ups_debounce_counter > Trip_Debounce:
                            print("Too many reads above threshold, regulation off")
                            M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                            flasher(5)
                            SG.control_ren(6)
                            ups_debounce_counter = 0
                            downs_debounce_counter = 0
                    #time.sleep(Wait_after_step)

            ########################################
            # Repeat above loop but below the threshold instead of above
            ########################################

            if read_value < (Error_signal_offset - Walk_Threshold):
                Downs += 1
                Ups = 0
                ups_debounce_counter = 0
                if Downs > Loops_Debounce:
                    temp_freq = GPIB.freq(SG)
                    GPIB.write_frequency(SG, (temp_freq - Step_size),"HZ")
                    print("Lowered Frequency ", i)
                    if (temp_freq - Step_size) < (Start_Freq - Max_Threshold):
                        print("Broken on too many steps downward")
                        M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                        flasher(5)
                        SG.control_ren(6)
                    if OS.query("MEASU:Meas{}:State?".format(Measurement))[-2] != str(1):
                        print("Measurement Off")
                        M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                        flasher(5)
                        SG.control_ren(6)
                    if read_value < (Error_signal_offset - Interlock_Threshold_mv):
                        print("Error: Deviation too far; read at {:.3e} mV".format(read_value))
                        ups_debounce_counter = 0
                        downs_debounce_counter += 1
                        if downs_debounce_counter > Trip_Debounce:
                            print("Too many reads below threshold, regulation off")
                            M.Write(Client, Running_Tag, False, Bool = True) #Breaks running loop on interlock
                            flasher(5)
```

```
                            SG.control_ren(6)
                            ups_debounce_counter = 0
                            downs_debounce_counter = 0
                    #time.sleep(Wait_after_step)

    else:
        SG.control_ren(6) #Returns to local control

        #time.sleep(Wait_between_reads)
        ups_debounce_counter = 0
        downs_debounce_counter = 0
        i += 1 #Update iterator
```