Jitter and Timed Loop Performance

Riley Kenyon

May 9th, 2019

Abstract

This report describes the concept of 'Jitter' and how it pertains to timed loops, loop priority, and the effects on performance. Several loop configurations are used to demonstrate how computational complexity increases Jitter and what methods can be applied to limit it.

1 Introduction

Jitter is a measure of deviation from the periodicity of a signal. Although in digital signals Jitter is not completely avoidable, it is important for controlling systems that the metric is bounded, meaning that it will reliably stay within a certain range. Jitter is a quantification of determinism (or how consistent certain timing events are), and is a measure of the difference between an expected event and the actual event. The variation is due to a multitude of factors including variation clock-cycle, and most notable code and loop period. This analysis of Jitter in real-time systems on the NI myRIO compares several different loop configurations in LabVIEW for performing real-time proportional control.

2 Loop Timing

To provide an example of Jitter for context, the LabVIEW script in Figure 1 depicts a setup where an output pin of the myRIO is shifted between high (1V) and low (0V) for every loop iteration (a period of 1 ms). This is achieved in practive by taking the difference between the current clock-time (µs) and the value for the previous iteration of the loop. Here it accomplished by a

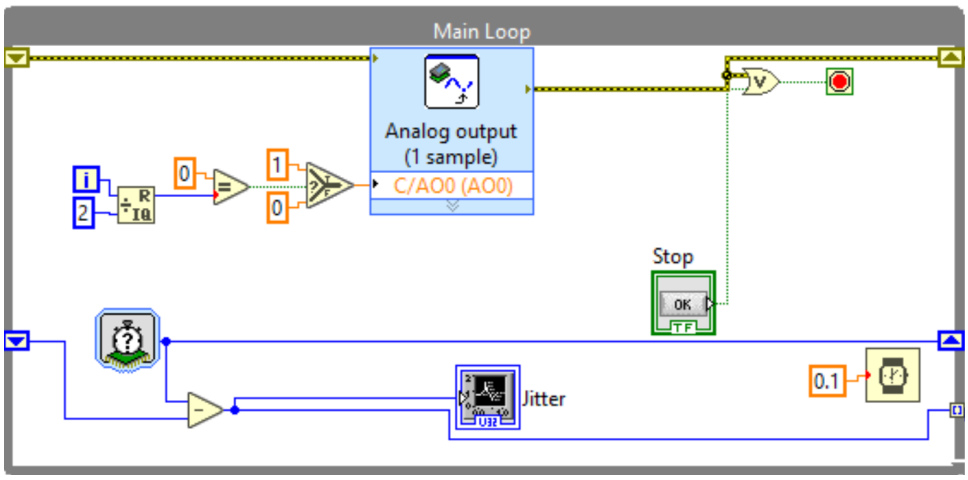

Figure 1: Labview code for providing an example of Jitter, the function generation occurs in the upper-left, where the value switches between zero and 1. Jitter is measured by the microsecond tick-counter (bottom-left), and the delay for the while loop is provided by the wait function (bottom-right).

shift register (the blue arrows on the left and right side of the loop in Figure 1). The analog output pin and ground are monitored with an oscilloscope to visualize the frequency variation of the signal. The square-wave in Figure 2 is observed to have significant variation from the desired frequency, with an average RMS (root mean square) value of 387 µs from the oscilloscope and 363 µs from the myRIO data, where RMS is defined in Equation 1 as

$$T_{RMS} = \sqrt{\frac{1}{n}(T_1^2 + T_2^2 + ... + T_n^2)}, \tag{1}$$

where n is the number of samples, and T is the period of the signal for each loop iteration. This is a measurement of the average period produced by the loop and can be compared against the expected value of the period (set loop time of for code execution). The greater the discrepancy between the T_{RMS} and expected value, the more severe the Jitter.

3 Limiting Jitter

Factors that influence Jitter can be mitigated by simplifying the code within the loop and decreasing memory intensive actions. To quantify how each

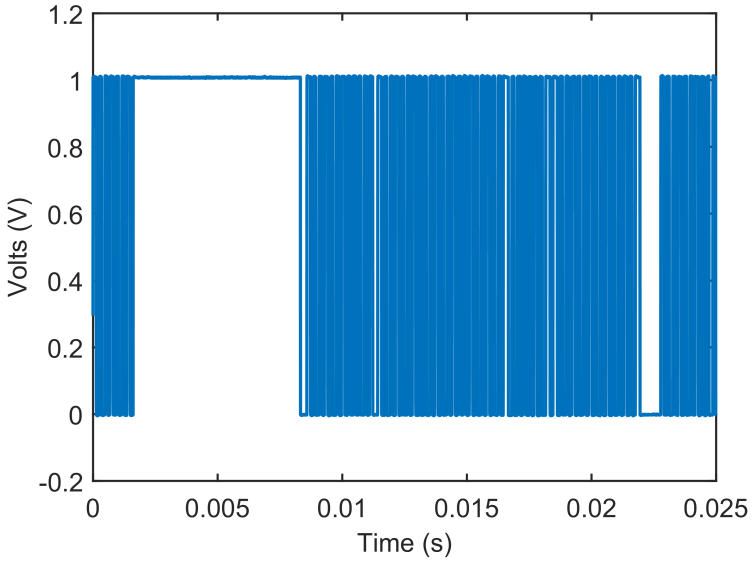

Figure 2: A square-wave produced by switching the analog output between 1V and 0V every loop iteration. Jitter is observed as a manifestation as the variation in the period of the square-wave.

configuration compares to the others being tested, we will use the value of T_{RMS} and a "maximum jitter" as it relates to the expected loop period.

3.1 Experimental Setup

The experiment was performed using a NI myRIO to perform real-time proportional control of an RRC circuit. The RRC circuit, shown in Figure 3, is an easy representation of a first-order system with a theoretical DC-gain of 0.5. Although this experiment is centered around quantifying Jitter, the controller is used to provide complexity to the loop. If operations within the loop did not require substantial resources, variations in Jitter between configurations would be less noticeable. The controller was connected to A0 (pins AGND and 0) on channel C to output analog voltage to the plant. The myRIO receives analog input from AI (0^+ and 0^-) on channel C, see Figure 5 for the wiring schematic and Figure 4 for the physical circuit. The Labview code is an implementation of a controller with proportional gain K, Figure 6 is the feedback loop showing how the myRIO is used with the physical circuit.

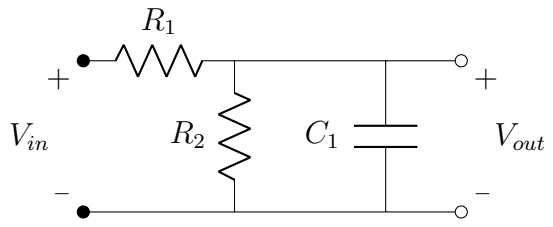

Figure 3: RRC Circuit - a simple first order system representation

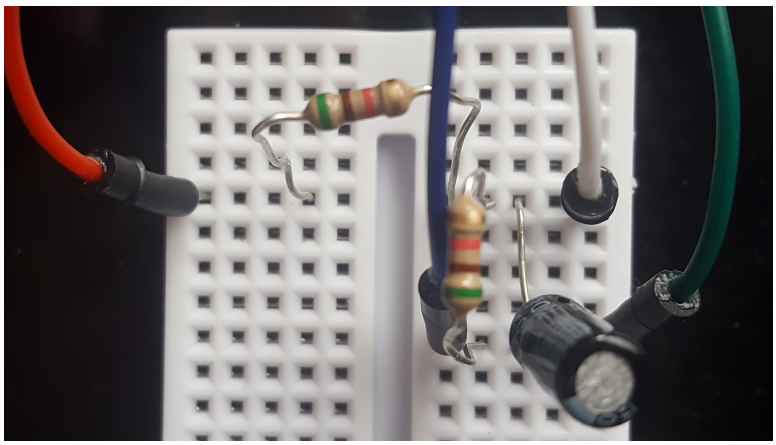

Figure 4: The RRC circuit with values of R_1 and $R_2 = 5\text{k}\Omega$, with a capacitor value of 2.22 µF wired to the NI myRIO with red (AO0), white (AI0+),green (AI0-), and blue (AGND)

Figure 5: The wiring diagram of the myRIO for proportional control of an RRC circuit, using analog input and output.

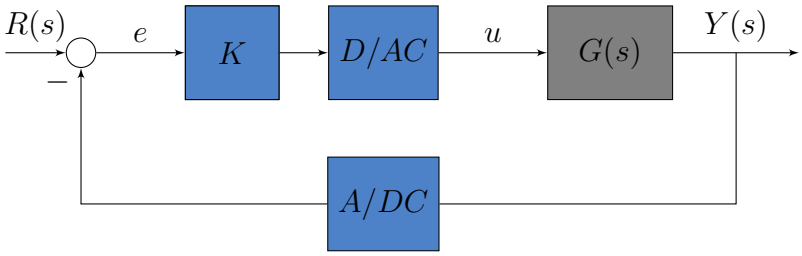

Figure 6: Closed Loop Block Diagram of signal conversion between myRIO(Blue) and input/output of circuit (Gray) using an digital to analog converter(D/AC) and an analog to digital converter (A/DC).

3.2 LabVIEW Setup

The LabVIEW script makes use of a feedback node (or a shift register) to measure the number of micro-second tics between loop iterations. The number of micro-seconds provides insight into how accurately the loop is executing a time specification. This is the data that is collected and analyzed to determine Jitter. The RRC circuit is sampled at frequencies of 10kHz, 1kHz, 100Hz to provide the basis of how each configuration performs at a variety of sampling rates. The number of samples used are 75000, 10000, and 1000 respectively for each setting. The setting of the wait function delay determines the rate at which it is sampled, for the subsequent timed loops, it is a setting that is specified when setting up the loop. Function generation is determined from the proportional constant K, set to $K = 5$ for this report, and the difference between the reference voltage and the output of the circuit. The frequency of the reference wave is determined by an input from the front panel in terms of Hz, then converted to a time index (in terms number of loop iterations at the tested frequency [100Hz, 1000Hz, 10000Hz]). To visualize the proportional controller the error(e), reference(r), controller output(u), and system output(y) are plotted. The data are indexed and saved to a csv file, then transferred off the myRIO into Matlab.

4 While Loop

The while loop is the initial configuration tested to determine how repeatable the timing of the loop timing. This is a loop without any techniques applied to reduce Jitter, and provides a value to improve off for alternate configurations. As seen in the code from Figure 7, all the plotting is performed within the loop and the only timing portion of the configuration is dictated by the 'wait' command (number of milliseconds).The feedback node is used instead of the shift register for simplicity in the loop but performs the same function of calculating Jitter based on a microsecond clock. The logic for function generation is the same, but with a control from the front panel dictating frequency of the square-wave reference. The proportional control occurs by subtracting plant output (y) from the reference voltage (r) and multiplying by a constant also given by the front panel. The Jitter performance T_{RMS} for the while loop is approximately accurate at the sampling rates of 100Hz and 1000Hz. However, as the timer get to the 10kHz sampling rate, the while

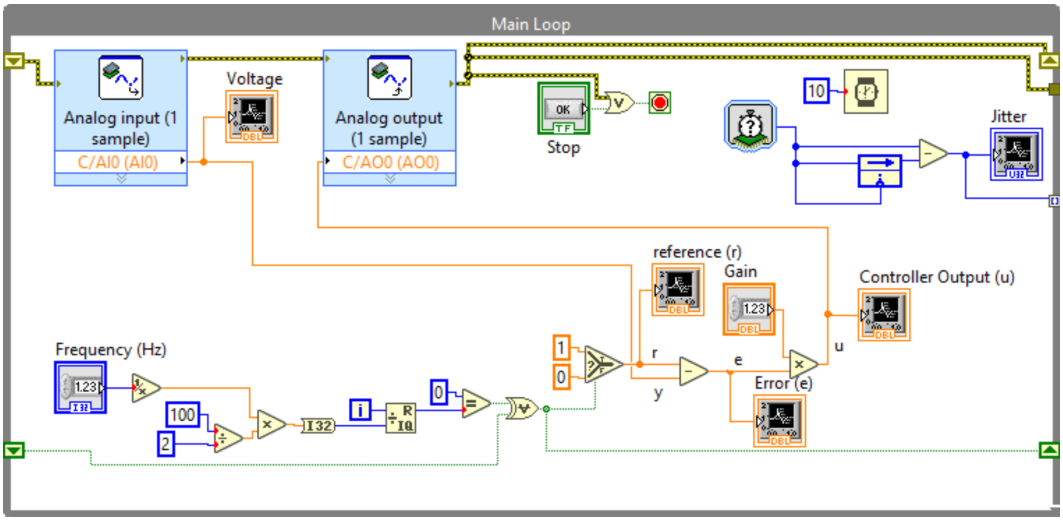

Figure 7: The Labview code for the while loop configuration, all of the plots are internal to the loop and is similar to the code for the initial test.

Frequency (Hz)	Jitter (T_{RMS})	Max Deviation(µs)
100	10246	10613
1000	1331	3676
10000	274	4710

Figure 8: The Jitter analysis of the while loop, notice that 10kHz (100µs) sampling frequency provides substantial deviation from the desired frequency due to the impossible time to complete the code within that given interval

loop cannot perform all the operations within the time span given. Additionally, the input of the wait command is blue(long) and not orange(double) so the method of timing in between loops in unreliable.

5 Timed Loop

The timed loop configuration utilizes the real-time toolbox of the NI myRIO, and replaces the while loop with a timed version. Changing the setup of the loop to use the micro-second clock allows for better timing of the controller. The improvement over the while loop is substantial, and converges to the expected time delays between loops for the 100Hz and 1000Hz sampling

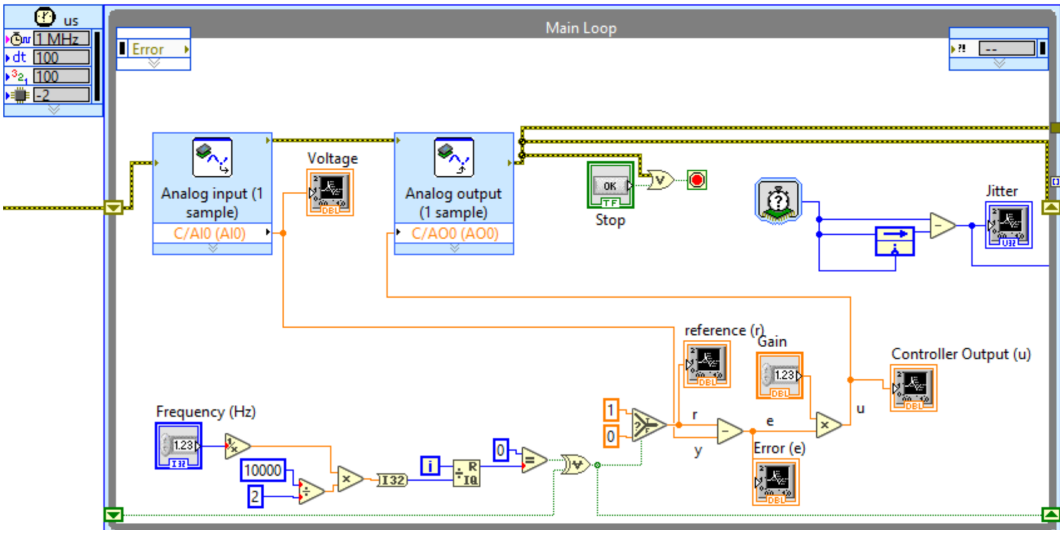

Figure 9: The Labview code for the timed loop configuration, the timing of each loop is based off a micro-second clock (upper-left tab), eliminating the need of the wait function from the while loop.

rates. The improvement is also seen in the 10kHz sampling rate, where the time average $T_{RMS} = 174\mu s$. All of the plotting still occurs within the loop, and indexes the Jitter data to save to a *.csv* file.

6 Plotting Externally

The next configuration takes all plotting out of the timed loop. This allows for the loop to not perform intensive operations such as real-time plotting. The average Jitter period $T_{RMS} = 195\mu s$ for the 10kHz sampling rate. The

Frequency (Hz)	Jitter (T_{RMS})	Max Deviation (µs)
100	10000	10046
1000	1000	1040
10000	174	4223

Figure 10: The Jitter analysis of the timed loop, notice the improvement in the execution time of the 10kHz sampling frequency. The other sampling rates still converge to their expected loop period.

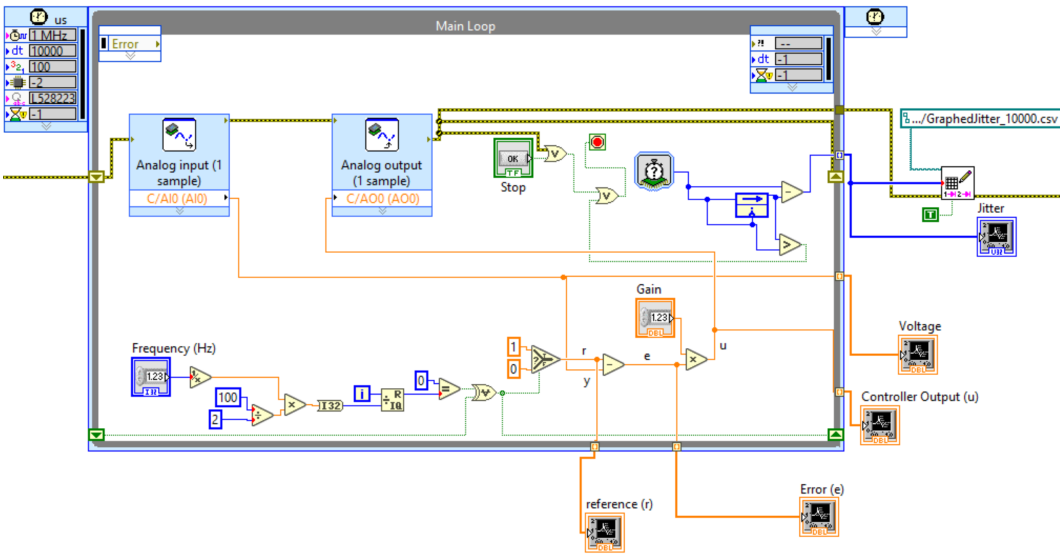

Figure 11: The Labview code for the external plotting configuration, the timing of each loop is still based off a micro-second clock (upper-left tab) but the computation of plotting real-time is eliminated and replaced with the cost of indexing an array to store the data of the plots.

other sampling rates also converged to their expected values for 100Hz and 1000Hz. It is important to note that likely the reason for a greater average value at the highest sampling frequency is due to the allocating of data to an array. The data continually gets added on to the same array as time progresses, considering how high the sampling rate is, the amount of memory needed to store all the information increases quickly. This is why there are spikes seen at specific intervals, when the size of the array needs to be increased, this results in a larger loop period for that specific iteration.

7 Separate While Loop

Another method of decreasing Jitter is to create different priority loops. For this, a while loop is implemented to take care of all of the 'psuedo real-time' plotting. Global variables are written to at each iteration of the main timed loop, and then read in the while loop at a lower priority. This implies that if the loop has enough time left before starting the next iteration, it will

Frequency (Hz)	Jitter (T_{RMS})	Max Deviation (µs)
100	10000	10043
1000	1000	1031
10000	195	9639

Figure 12: The Jitter analysis of the external plotting configuration. The execution time of the 10kHz sampling rate increased from the timed loop, likely due to the amount of array assignment needed to hold the plot data from several sources (r,e,u,y), the maximum deviation occurs at the end of the data.

plot the variables in the lower priority loop. Although the plots are not exactly representative of what is going on with the proportional controller, it provides some insight into how the system is performing without waiting until the end of the program or sacrificing loop time. The caviat of this configuration is similar to the external plotting, as it requires the usage of global variables that are memory intensive. The performance with sampling rates of 100Hz and 1000Hz are the same as in previous timed loops, the T_{RMS} is equal to the expected loop time (10000 µs and 1000 µs respectively. As for the higher sampling frequency of 10kHz, the system performed better than the while loop, but worse than the only timed loop and the external plotting configurations. This is likely because of the usage of global variables for plotting, which is more intensive than indexing an array or simply plotting within the loop.

8 Effects of Sampling Rate

As seen in Figure 15 and 16, the performance for 100Hz and 1000Hz sampling was increased greatly by the use of timed loops. The average value of all configurations except the while loop achieved an RMS value equal to the expected execution time of the loop. The high sampling frequency of 10kHz was enough to cause inaccuracy for loop period. At this rate, it is impossible to execute the code within the set time even with the implementation of a timed loop. Specifically for all the examples where indexing was used to keep track of the elements of an array, timing was progressively worse when the length of the array continued increasing. In this range, timed loop still out performed the priority setting likely due to the use of global variables

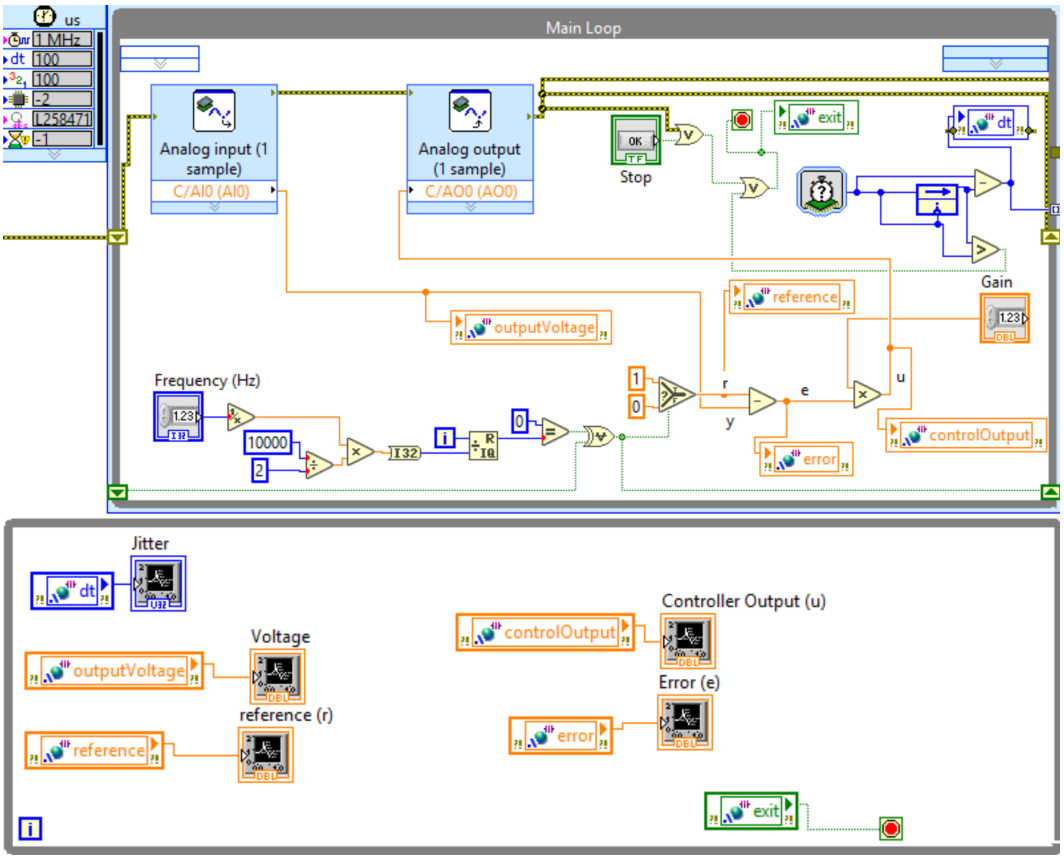

Figure 13: The Labview code for the use of priority loops for plotting 'psuedo real-time', main loop is timed and is used for the proportional control of the circuit using the same logic as previous iterations. Instead of directly plotting the variables, the value is saved to a global variable and used in the while loop beneath to plot when there is excess time after performing the main function of the timed loop.

Frequency (Hz)	Jitter (T_{RMS})	Max Deviation
100	10000	10026
1000	1000	1039
10000	210	4354

Figure 14: The Jitter analysis of priority loops, with the primary loop performing the computation for proportional control and saving values to a global variable. The secondary loop is a while loop that provides 'psuedo real-time plotting' of the control loop.

(writing to and reading from) but graphically both appear to act similarly. External plotting was better for lower sampling frequencies, at the rate of 10kHz the arrays become large quickly and require resources that end up increasing the execution time of the loop.

9 Conclusion

Ensuring that Jitter is bounded is important for real-time control systems. For example, if the time it takes to execute a loop is considerably longer than the expected time for a loop, the system will be exposed to the controller's output for a longer duration. This can result in overshoot or becoming momentarily unresponsive to reference signals. As seen using the methods above, the amount of Jitter in a loop of code can be mitigated using a timed loops, varying priority loops, and externally plotting (due to being less memory intensive). For saving data, the array will become large enough to the point where it is resized, resulting in a "spike" in Jitter to expand the array. This was seen extensively in the external plotting configuration of the loop where multiple variables had to be indexed and plotted after the code was stopped. The amount of resources necessary to hold all the data were seen in the plot of the the 10kHz sampling rate. Although Jitter is unavoidable in digital systems based off clock cycles, timing methods can be implemented to decreased the bounds of Jitter and reducing the amount of plotting decreases the computational expense. This is specifically applicable to high sampling frequencies, at lower sampling frequencies a timed loop may be all that is necessary to bring the system within acceptable bounds.

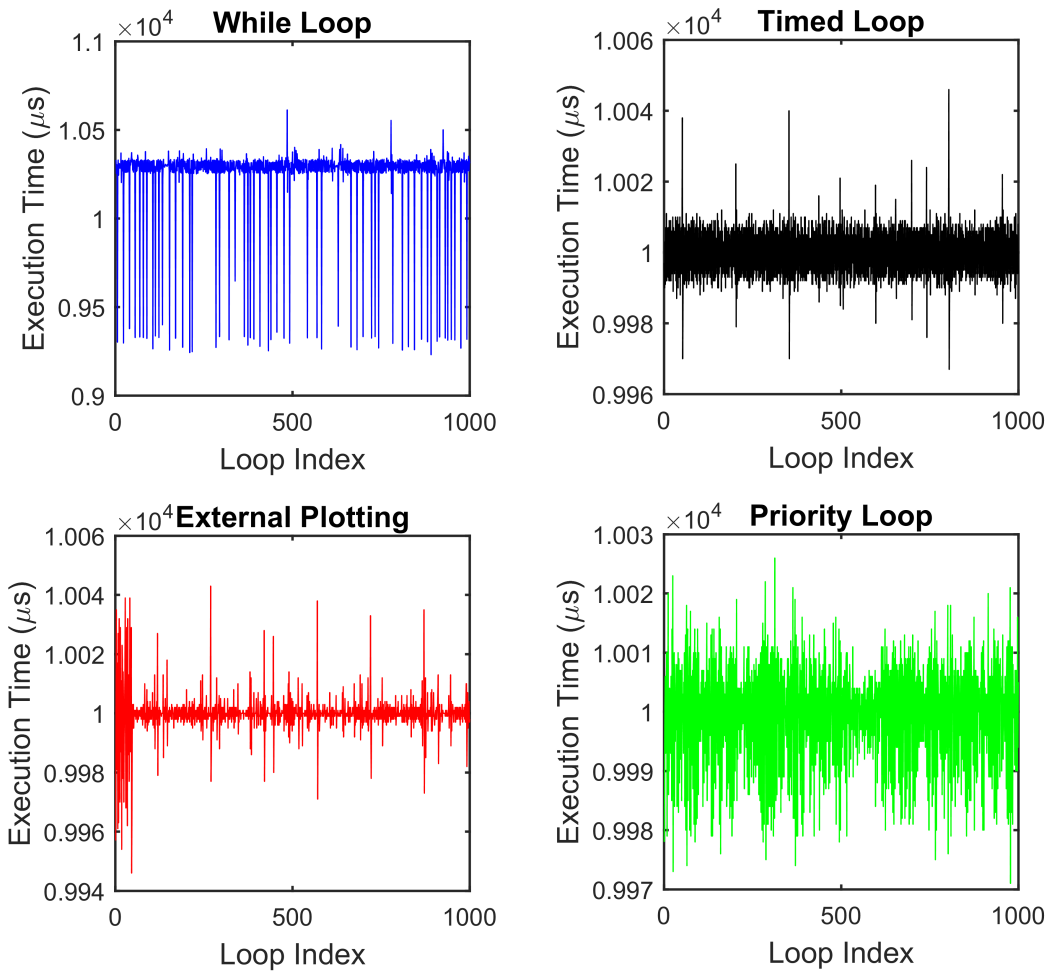

Figure 15: The Jitter resulting from a sampling rate of 100Hz, notice that graphing externally results in a band closer to the desired execution time of 100 µs.

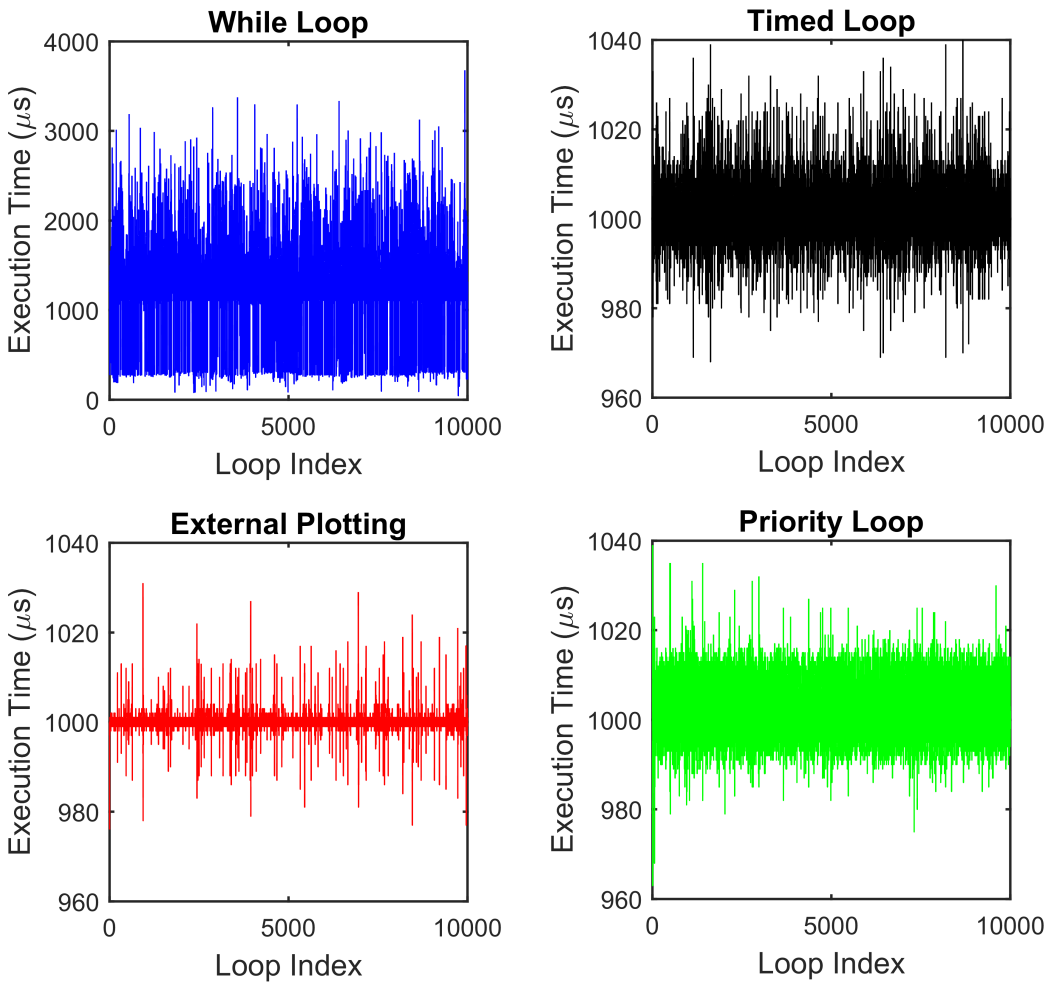

Figure 16: The Jitter resulting from a sampling rate of 1000Hz, notice that graphing externally results in a band closer to the desired execution time of 1000 µs.

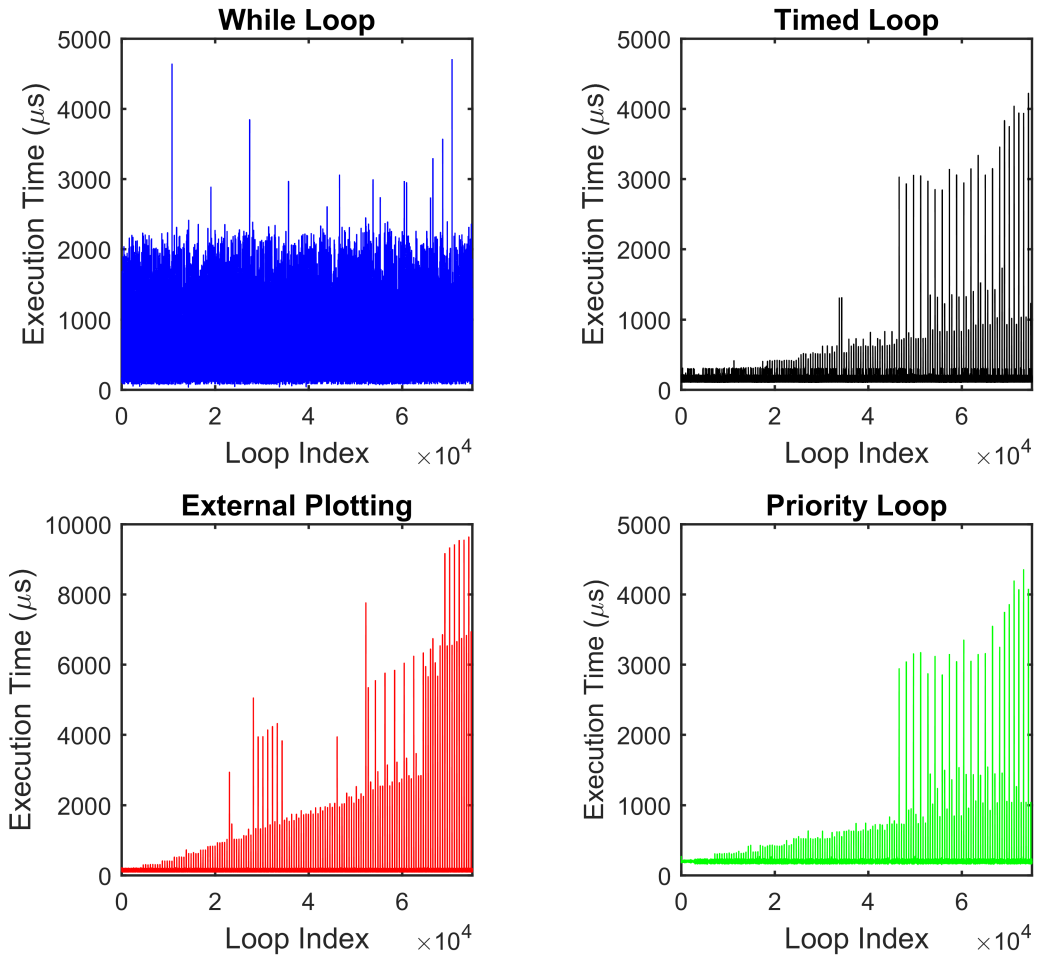

Figure 17: The Jitter resulting from a sampling rate of 10kHz, as the number of samples increases over the course of the script, arrays holding the previous loop iterations of data increase significantly resulting in a greater execution time.

A Code

A.1 Creating Plots

```
%--------------------------------------------------------------------------
%% Industrial Automation Lab 4 - Jitter
% Riley Kenyon 4/9/2019
%--------------------------------------------------------------------------
clear all; close all; clc;
%% Example of Jitter
oscilloscope_100 = csvread('jitter100.csv',3,0);
%oscilloscope_100(:,1) = oscilloscope_100(:,1) + oscilloscope_100(1,1);
num = 0;
temp = 0;
sum = 0;
j = 1;
% Finding oscilloscope vs myRIO data
for i = 2:length(oscilloscope_100(:,1))
    if oscilloscope_100(i,2) < 0.5  && oscilloscope_100(i-1,2) > 0.5
        temp1 = oscilloscope_100(i,1);
        if temp ~= 0
            num = num+ 1;
            disp(temp1-temp)
            diff(j) = temp1-temp;
            j = j+1;
        end
        temp = temp1;
    elseif oscilloscope_100(i,2) > 0.5 && oscilloscope_100(i-1,2) < 0.5
        temp1 = oscilloscope_100(i,1);
        if temp ~= 0
            num = num+1;
            disp(temp1-temp)
            diff(j) = temp1-temp;
            j = j+1;
        end
        temp = temp1;
    end
end
```

```
RMS.Theoretical = rms(diff);
oscilloscope_jitter = csvread('OscilloscopeJitter_100.csv',100000); %set towards e
RMS.oscilloscope = rms(oscilloscope_jitter);
fprintf("%4.0f \n",RMS.oscilloscope);
[pk,index] = findpeaks(oscilloscope_100(:,2));
figure()
plot(oscilloscope_100(:,1),oscilloscope_100(:,2));
%xlim([0,0.025]);
xlabel('Time (s)');
ylabel('Volts (V)');
%% While Loop configuration
WhileLoop_0 = csvread('whileJitter_0.csv',2);    % "0" millisecond delay
WhileLoop_0 = WhileLoop_0(1:10000);
WhileLoop_100 = csvread('whileJitter_100.csv',2);   % "0.1" millisecond delay
WhileLoop_100 = WhileLoop_100(1:75000);
WhileLoop_1000 = csvread('whileJitter_1000.csv',2);  % 1 millisecond delay
WhileLoop_1000 = WhileLoop_1000(1:10000);
WhileLoop_10000 = csvread('whileJitter_10000.csv',2);
WhileLoop_10000 = WhileLoop_10000(1:1000);
RMS.WhileLoop(1) = rms(WhileLoop_0);
RMS.WhileLoop(2) = rms(WhileLoop_100);
MAX.WhileLoop(1) = max(WhileLoop_100);
RMS.WhileLoop(3) = rms(WhileLoop_1000);
MAX.WhileLoop(2) = max(WhileLoop_1000);
RMS.WhileLoop(4) = rms(WhileLoop_10000);
MAX.WhileLoop(3) = max(WhileLoop_10000);
fprintf("While Loop RMS\n----------------------\n");
fprintf("T_s = 0 us: %4.0f\n ",RMS.WhileLoop(1));
fprintf("T_s = 100 us: %4.0f %4.0f \n",RMS.WhileLoop(2),MAX.WhileLoop(1));
fprintf("T_s = 1000 us: %4.0f %4.0f \n",RMS.WhileLoop(3),MAX.WhileLoop(2));
fprintf("T_s = 10000 us: %4.0f %4.0f \n",RMS.WhileLoop(4),MAX.WhileLoop(3));

%% Timed Loop Configuration
TimedLoop_100 = csvread("TimedJitter_100.csv",2);
TimedLoop_100 = TimedLoop_100(1:75000);
TimedLoop_1000 = csvread("TimedJitter_1000.csv",2);
TimedLoop_1000 = TimedLoop_1000(1:10000);
TimedLoop_10000 = csvread("TimedJitter_10000.csv",2);
```

```
TimedLoop_10000 = TimedLoop_10000(1:1000);
RMS.TimedLoop(1) = rms(TimedLoop_100);
RMS.TimedLoop(2) = rms(TimedLoop_1000);
RMS.TimedLoop(3) = rms(TimedLoop_10000);
MAX.TimedLoop(1) = max(TimedLoop_100);
MAX.TimedLoop(2) = max(TimedLoop_1000);
MAX.TimedLoop(3) = max(TimedLoop_10000);
fprintf("\nTimed Loop RMS\n-----------------------\n");
fprintf("T_s = 100 us: %4.0f %4.0f\n",RMS.TimedLoop(1),MAX.TimedLoop(1));
fprintf("T_s = 1000 us: %4.0f %4.0f\n",RMS.TimedLoop(2),MAX.TimedLoop(2));
fprintf("T_s = 10000 us: %4.0f %4.0f\n",RMS.TimedLoop(3),MAX.TimedLoop(3));

%% Moved Graph Configuration
GraphedLoop_100 = csvread("GraphedJitter_100.csv",2);
GraphedLoop_100 = GraphedLoop_100(1:75000);
GraphedLoop_1000 = csvread("GraphedJitter_1000.csv",2);
GraphedLoop_1000 = GraphedLoop_1000(1:10000);
GraphedLoop_10000 = csvread("GraphedJitter_10000.csv",2);
GraphedLoop_10000 = GraphedLoop_10000(1:1000);
RMS.GraphedLoop(1) = rms(GraphedLoop_100);
RMS.GraphedLoop(2) = rms(GraphedLoop_1000);
RMS.GraphedLoop(3) = rms(GraphedLoop_10000);
MAX.GraphedLoop(1) = max(GraphedLoop_100);
MAX.GraphedLoop(2) = max(GraphedLoop_1000);
MAX.GraphedLoop(3) = max(GraphedLoop_10000);
fprintf("\nMoved Graphs RMS\n-----------------------\n");
fprintf("T_s = 100 us: %4.0f %4.0f\n",RMS.GraphedLoop(1),MAX.GraphedLoop(1));
fprintf("T_s = 1000 us: %4.0f %4.0f\n",RMS.GraphedLoop(2),MAX.GraphedLoop(2));
fprintf("T_s = 10000 us: %4.0f %4.0f\n",RMS.GraphedLoop(3),MAX.GraphedLoop(3));

%% Priority Loops and Local Variables
PriorityLoop_100 = csvread("Priority_100.csv",2);
PriorityLoop_100 = PriorityLoop_100(1:75000);
PriorityLoop_1000 = csvread("Priority_1000.csv",2);
PriorityLoop_1000 = PriorityLoop_1000(1:10000);
PriorityLoop_10000 = csvread("Priority_10000.csv",2);
PriorityLoop_10000 = PriorityLoop_10000(1:1000);
RMS.PriorityLoop(1) = rms(PriorityLoop_100);
```

```
RMS.PriorityLoop(2) = rms(PriorityLoop_1000);
RMS.PriorityLoop(3) = rms(PriorityLoop_10000);
MAX.PriorityLoop(1) = max(PriorityLoop_100);
MAX.PriorityLoop(2) = max(PriorityLoop_1000);
MAX.PriorityLoop(3) = max(PriorityLoop_10000);
fprintf("\nPriority Loops RMS\n-----------------------\n");
fprintf("T_s = 100 us: %4.0f %4.0f\n",RMS.PriorityLoop(1),MAX.PriorityLoop(1));
fprintf("T_s = 1000 us: %4.0f %4.0f\n",RMS.PriorityLoop(2),MAX.PriorityLoop(2));
fprintf("T_s = 10000 us: %4.0f %4.0f\n",RMS.PriorityLoop(3),MAX.PriorityLoop(3));

%% Graphical Representation of each
figure()
subplot(2,2,1); plot(WhileLoop_100,'b');title("While Loop");ylabel("Execution Time
subplot(2,2,2); plot(TimedLoop_100,'k');title("Timed Loop");ylabel("Execution Time
subplot(2,2,3); plot(GraphedLoop_100,'r');title("External Plotting");ylabel("Execu
subplot(2,2,4); plot(PriorityLoop_100,'g');title("Priority Loop");ylabel("Executio

figure()
subplot(2,2,1); plot(WhileLoop_1000,'b');title("While Loop");ylabel("Execution Tim
subplot(2,2,2); plot(TimedLoop_1000,'k');title("Timed Loop");ylabel("Execution Tim
subplot(2,2,3); plot(GraphedLoop_1000,'r');title("External Graphs");ylabel("Execut
subplot(2,2,4); plot(PriorityLoop_1000,'g');title("Priority Loop");ylabel("Executi

figure()
title("10000 Microseconds");
subplot(2,2,1); plot(WhileLoop_10000,'b');title("While Loop");ylabel("Execution Ti
subplot(2,2,2); plot(TimedLoop_10000,'k');title("Timed Loop");ylabel("Execution Ti
subplot(2,2,3); plot(GraphedLoop_10000,'r');title("External Graphs");ylabel("Execu
subplot(2,2,4); plot(PriorityLoop_10000,'g');title("Priority Loop");ylabel("Execut
```