Voltage Control of an RLC Circuit

Riley Ruckman

TCES 455, Autumn 2020

December 12, 2020

Table of Contents

Project Objectives

The purpose of this project is to use the information and tools learned in this course to design a PI controller for the voltage regulation of a RLC circuit using simulation and two implementations: MATLAB/Simulink with Arduino support packages, and Arduino sketches.

The objectives of this project is to:

- Understand the step response of an RLC circuit with multiple system configurations (Open Loop, Closed Loop, Closed Loop with PI Controller)
- Use the rltool() utility in MATLAB to design a PI controller that meets the specified requirements
- Build an implementation of the system in Simulink/Arduino and Arduino Sketch
- Understand the differences in each system type and approach (simulation and implementation) in terms of 2^{nd}-order system behavior and constants.

Measurements/Tests

Open Loop

To start with the testing of the step response, a transfer function for the given system must first be developed. A circuit diagram for the RLC circuit is shown in Figure 1.

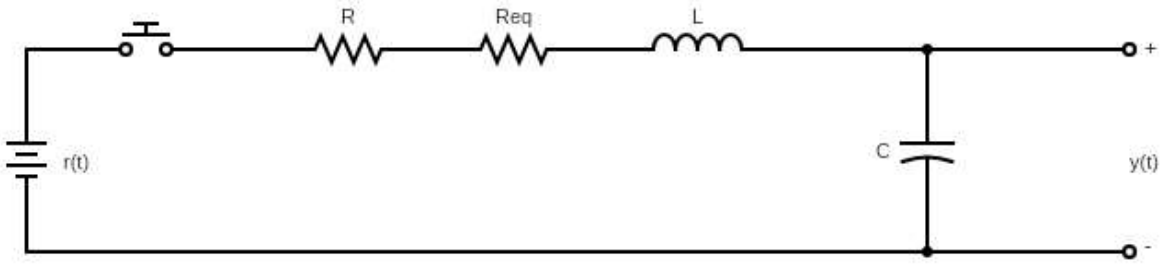


Figure 1: RLC circuit

The output of the system will be the voltage across the capacitor. To analyze this circuit in terms of the transfer function, a Laplace transform of the entire circuit will be conducted. The result of this is seen in Figure 2.

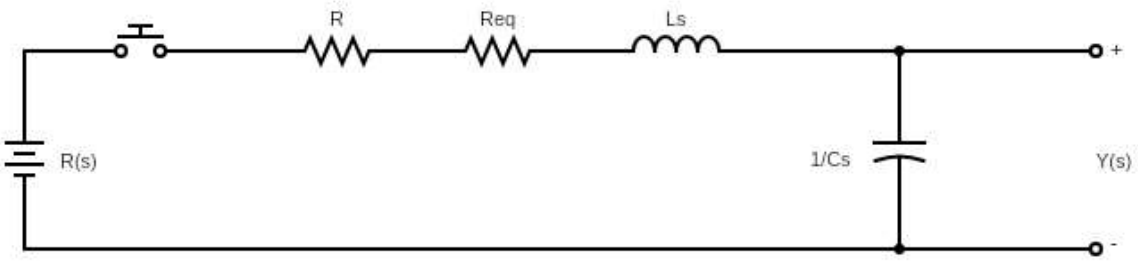


Figure 2: Laplace Transform of RLC circuit

To find the voltage across the capacitor, which is the output of this circuit, the voltage divider equation can be used:

$$Y(s) = V_C = R(s)\frac{\frac{1}{Cs}}{R + R_{eq} + Ls + \frac{1}{Cs}}$$

Then, solving for G(s) = Y(s)/R(s), the transfer function of this circuit is:

$$G(s) = \frac{Y(s)}{R(s)} = \frac{\frac{1}{Cs}}{R + R_{eq} + Ls + \frac{1}{Cs}}$$

Altering this equation gives us,

$$G(s) = \frac{\frac{1}{CL}}{s^2 + \frac{s(R + Req)}{L} + \frac{1}{CL}}$$

which can be written in the form,

$$G(s) = \frac{\omega_n^2}{s^2 + 2\omega_n\zeta s + \omega_n^2}$$

where,

$$\omega_n^2 = \frac{1}{CL} \Rightarrow \omega_n = \sqrt{\frac{1}{CL}}$$

$$\zeta\omega_n = \frac{R + R_{eq}}{2L}$$

$$\zeta = \frac{\zeta\omega_n}{\omega_n} = \frac{R + R_{eq}}{2}\sqrt{\frac{C}{L}}$$

Given the chosen nominal component values of R = 10 Ω, R_{eq} = 41 Ω, L = 1 H, and C = 470 μF, the given transfer function and constants can be found,

$$G(s) = \frac{2128}{s^2 + 51s + 2128}$$

$$\omega_n^2 = 2128 \Rightarrow \omega_n = 46.13$$

$$\zeta\omega_n = 25.5,\ \zeta = 0.553$$

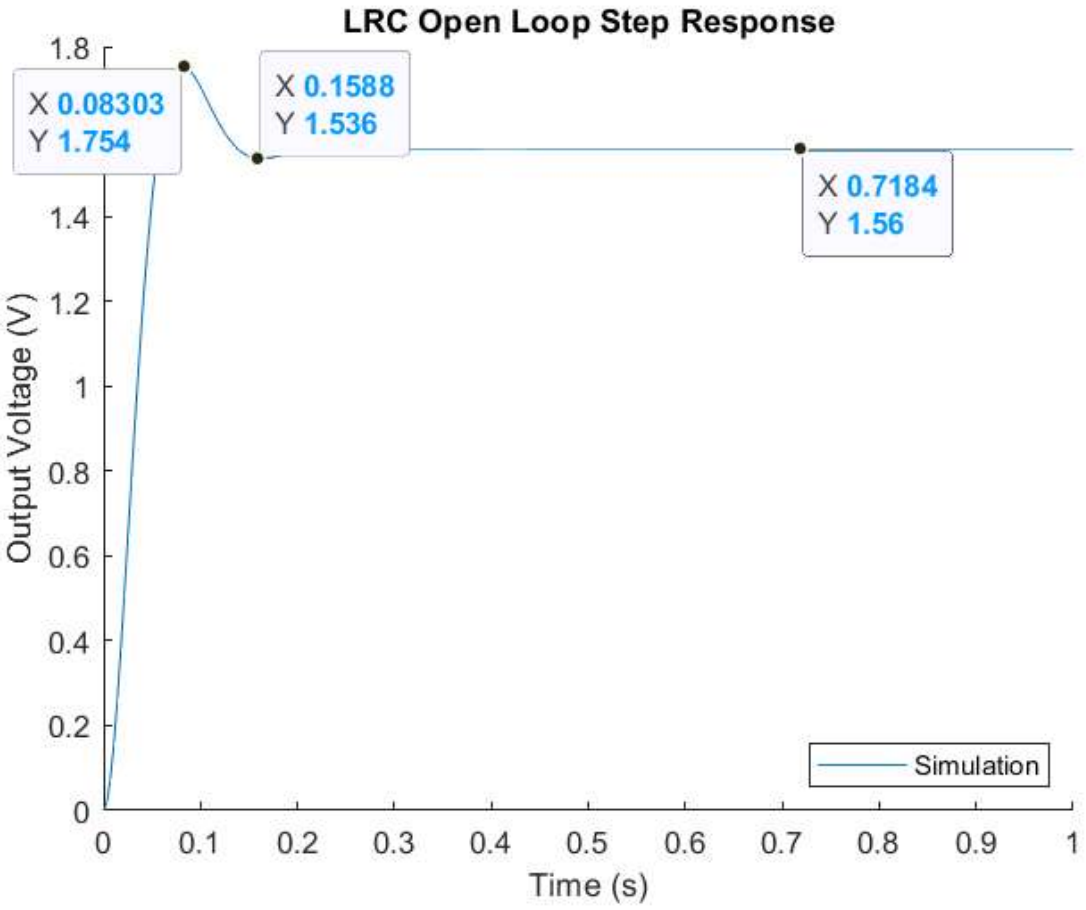
With these values for ζ and ω_n, the model OS% and T_S can be calculated using the following equations:

$$OS\% = 100e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}}$$

$$T_s = \frac{4}{\zeta\omega_n}$$

ζ and ω_n can be solved for in these two equations to give,

$$\zeta = \frac{-ln(OS\%/100)}{\sqrt{\pi^2 + ln^2(OS\%/100)}}$$



$$\omega_n = \frac{4}{\zeta T_s}$$

Figure 3: Simulated Open Loop Step Response

Using this information as a guide, a simulation of the step response of this open loop system was created using MATLAB. Given a step input amplitude of 1.56V to closely simulate the actual voltage of the battery being used in the real circuit, the graph seen in Figure 3 displays the simulated step response of the open loop system.

The key points on this graph focus on the values relating to the percent overshoot and settling time of the circuit. Just to verify the simulation values, let's calculate the ζ and ω_n using V_{max} and T_s for the simulation plot: V_{max} = 1.754 V, T_s = 0.1588 s. The percent overshoot can also be calculated using,

$$OS\% = 100\left(\frac{V_{max}}{V_{SS}} - 1\right) = 12.436$$

$$\zeta = \frac{-ln(12.436/100)}{\sqrt{\pi^2 + ln^2(12.436/100)}} = 0.552$$

$$\omega_n = \frac{4}{\zeta T_s} = \frac{4}{0.552(0.1588)} = 45.63$$

The value for the ζ is exactly the same as the previous calculation of ζ. ω_n is probably different due to the selected point on the graph, as it is hard to find the best point where the function is with 2% of the steady-state value. Other than that discrepancy, this method of finding ζ and ω_n will be useful for comparing the characteristics of the system as it changes from open loop to closed loop.

Closed Loop

For the closed loop step response, I needed to "close the loop" in the system. This required subtracting the output of G(s) from the input into the system, R(s), creating an error signal e(s). This provides uncontrolled-feedback into the system, which may result in unwanted behavior. A block diagram depicting this feedback loop is shown in Figure 4.

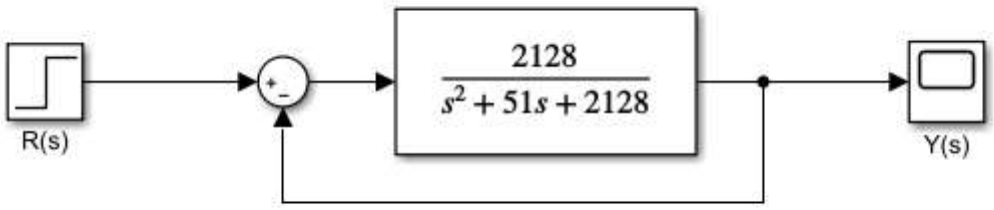


Figure 4: Closed Loop Block Diagram

The same analysis done in the Open Loop section can be applied to the closed loop step response. The closed loop step response from simulation is shown in Figure 5. An immediate difference seen between the open loop and closed loop responses is the increased

SSE in the signals, as both reached around 0.78 V in steady-state. This is a 50% error relative to the input, and will be a key point to fix when designing the controller.

The closed loop step response has a V_{max} of 0.984 V (V_{SS} = 0.78 V) and T_s of 0.1305 s, resulting in a OS%, ζ, and ω_n of 26.15%, 0.393 and 77.99, respectively. Compared to the open loop simulation, this system is less damped and oscillates more. This can be seen in the lower value of the closed loop's ζ, which contributes to the larger overshoot. Also, visibly, it can be seen that there are more oscillatory fluctuations in the step response between 0.1s and 0.2s compared to the open loop. This is likely due to the effect of both the lower ζ and higher ω_n, since the lower ζ allows the response to oscillate more and the higher ω_n is a higher frequency of oscillation. The large SSE is due to the system becoming stable when the output is equal to half of the input. Given that an error signal e(s) is the input signal of the RLC circuit, the system will settle when the input and output of the circuit are the same, or when

$$e(s) = Y(s)$$

Substituting

$$e(s) = R(s) - Y(s)$$

into the previous equation gives

$$R(s) = 2Y(s),$$

$$\text{or } Y(s) = \frac{R(s)}{2}$$

so the output will always become half of the system input in an uncontrolled closed loop system. Overall, this system is not ideal as there is a large SSE and larger overshoot.

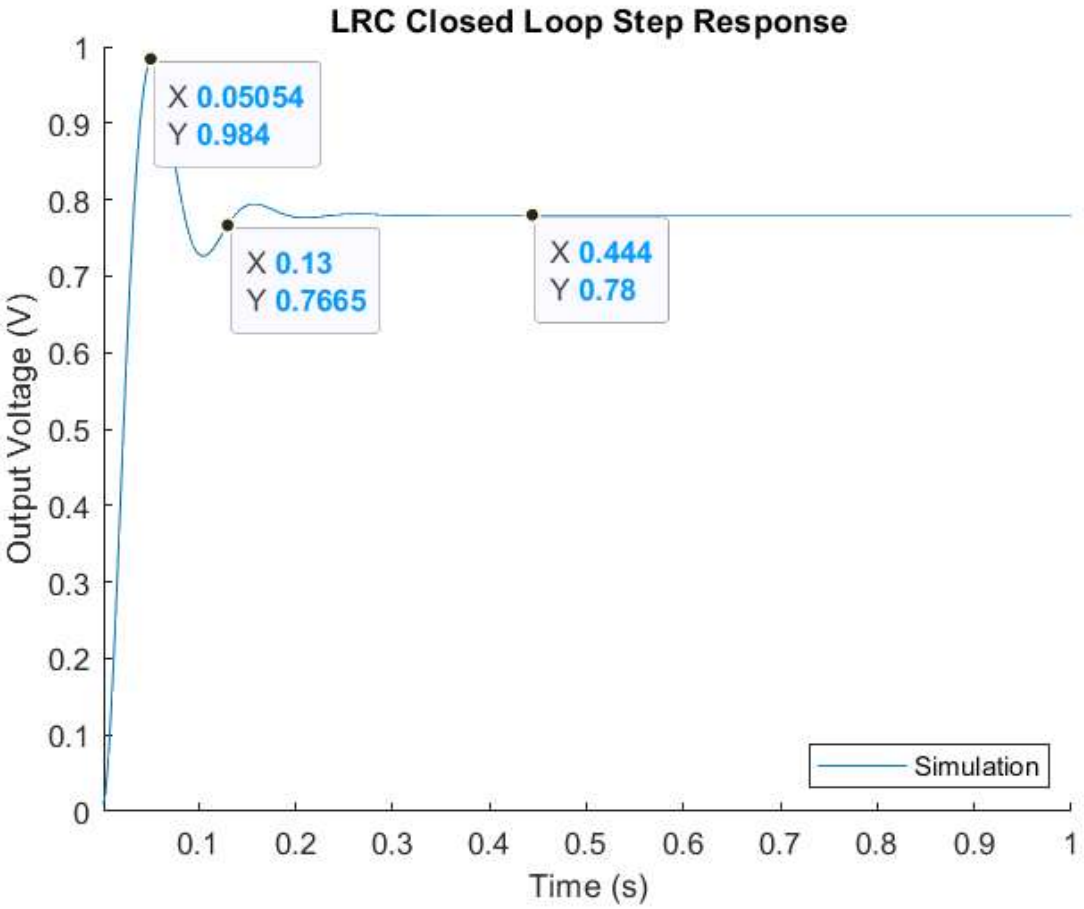


Figure 5: Simulated Closed Loop Step Response

System Design & Operation

The system's purpose is to reach steady-state without a SSE, and to reach that voltage without too many variations in the output and within an acceptable window of time. Therefore, a system is designed with a feedback loop to allow itself to read its output and to adjust accordingly. Along with the feedback, a PID controller is implemented to read in the error so it can adjust the function's output to maintain a designed behavior. This is important for applications like automated speed control for cars or DC motors that need to maintain a steady speed, which requires a specific voltage at the motor. A block diagram of the closed loop system with an included controller is shown in Figure 6.

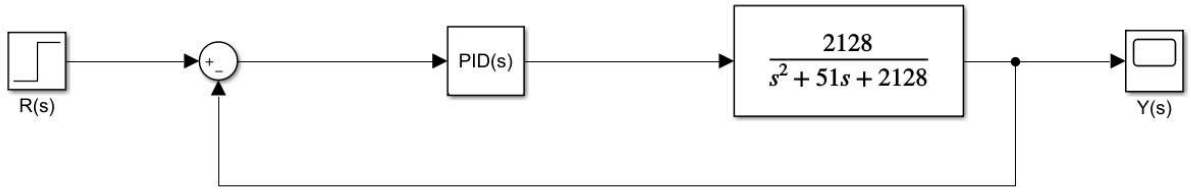


Figure 6: Closed Loop With a PID Controller

The PID controller for this design will be comprised of the Proportional and Integral portions of the controller (will be referred to as a PI controller from now on), as specified by the project's guidelines. The proportional portion only applies a gain of value K_P to the error, while the Integral portion integrates the error and applies a gain of K_I to the error. By carefully selecting the values of K_P and K_I, a controller that fits our desired performance specifications can be produced.

Controller Design - Root Locus

For the controller design for this system, the requirements specified are:

- Remove the SSE
- OS% < 15%
- T_S is acceptable. (Let's generalize this as T_S < 2 s)

The first step of this process is to find the characteristic equation of our system. This typically involves finding the transfer function of the closed loop system with the PI controller, simplifying the fraction, and taking the denominator and modifying it to get it in the form

$$1 + K\frac{n(s)}{d(s)} = 0,$$

where n(s) is the numerator and d(s) in the denominator. After testing, I quickly found that K(n(s)/d(s)) can be calculated by multiplying the controller and G(s) together. The controller is composed of a Proportional gain K_P and an Integral gain and integrator K_I/s, which are added together. Combining these two portions of the controller into one fraction creates,

$$\frac{K_p s + K_I}{s}$$

This product gives a characteristic equation of,

$$1 + \frac{K_p s + K_I}{s}\left[\frac{2128}{s^2 + 51s + 2128}\right] = 0$$

In order to perform a root locus analysis of this characteristic equation, only one gain can be used as a variable. Due to this, K_P is pulled out of the controller function to make,

$$\frac{K_p(s + \tau)}{s}$$

where tau is the ratio K_I/K_P. By setting tau to an arbitrary value, K_P can be adjusted until a valid step response is found.

MATLAB has a useful utility called rltool() just for this, where the characteristic function (except the “+ 1”), is entered into the function as the argument. Using this tool provided a set of K_P and tau values where the requirements of the system are met : K_P = 2.4 and tau = 10, which makes K_I = 2.4(10) = 24. This set of gains for the PI controller results in a OS% of 10.1% and a T_S of 0.351 s, which meets the requirements stated earlier. A screenshot of this tool, and a simulated step response of the closed loop with a PI controller, is displayed in Figure 7.

To find the ζ and ω_n of this system, the peak amplitude and settling time values in Figure 7 can be used. Given V_{max} = 1.1 V (the step input is 1 V in the utility) and T_S = 0.351 s: OS% = 10%, ζ = 0.591, and ω_n = 19.28. Compared to the open loop step response, this response has similar OS% and ζ values, yet has more oscillations when approaching steady state than the open loop. This is probably due to the more-complex nature of the closed loop system since it is influenced by the extra pole created by the Integral portion of the PI controller.

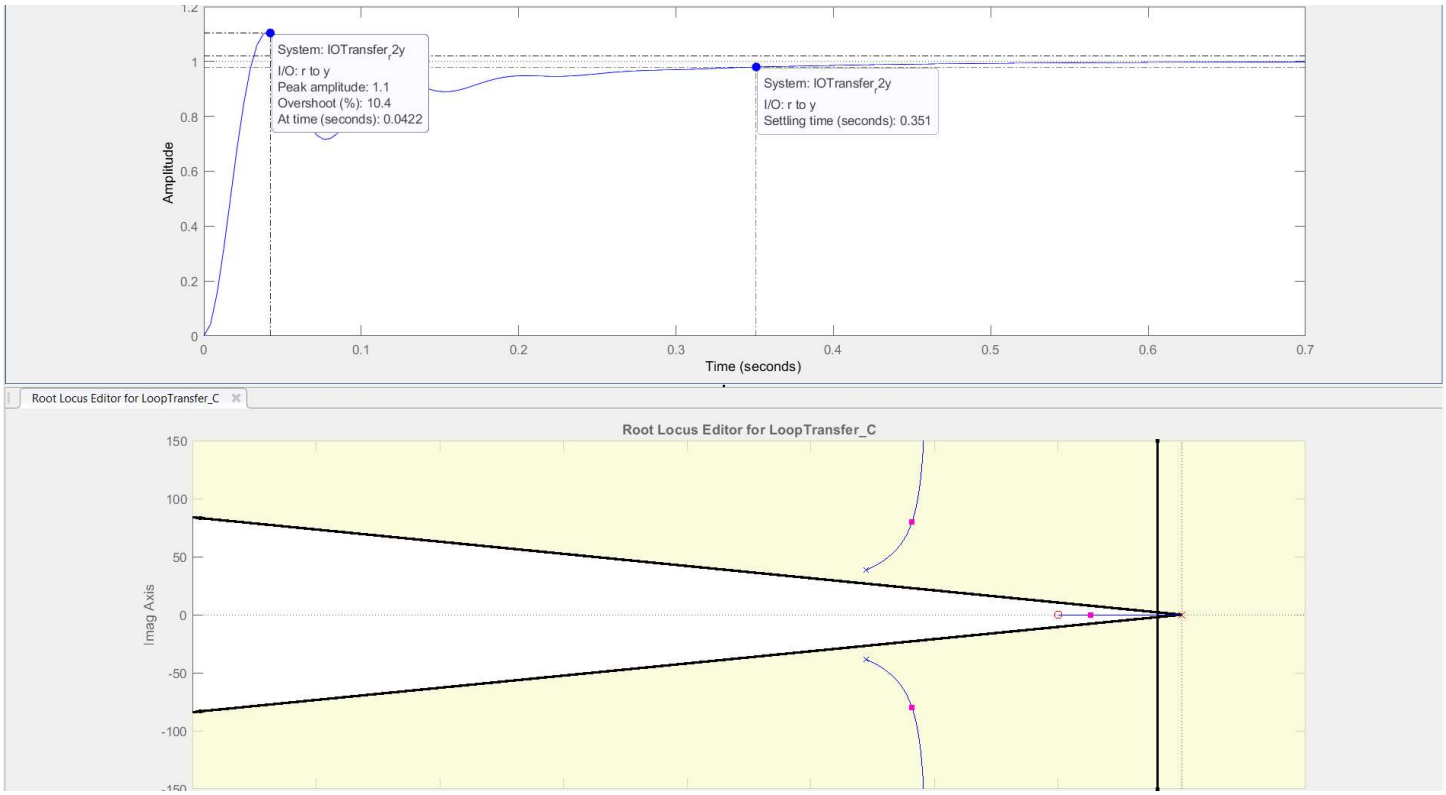


Figure 7: rltool() analysis of characteristic equation, along with respective step response.

Implementation

For both the Simulink/Arduino and Arduino Sketch implementations, the following hardware was used:

- 10 Ω resistor for RLC circuit
- 10 kΩ pulldown resistor for step input
- 1 H inductor (158SA) for RLC circuit
- 470 µF Capacitor for RLC circuit
- Two push buttons (One was used to provide the step input from the battery, and the other was used to easily discharge the capacitor)
- 1.56 V AA alkaline battery & cradle with positive and negative leads
- Arduino Mega 2560 board
- 8 M-M cables

Simulink/Arduino

Open Loop

Implementing this circuit involved the physical construction of the RLC circuit on a breadboard. Since feedback was not introduced at this step of the system design, there wasn't a need to read the voltage of the battery into the Arduino for processing. The battery was connected to a button, which was connected to the resistor so when the button is pressed, it would simulate a step input of 1.56 V into the system. In a Simulink block model, Pin A0 on the Arduino read the voltage of the capacitor, and this data was sent to the MATLAB workspace for processing into usable plots. A sample time of 0.01 s was chosen due to the fast response of the system, and a runtime of 10 s was chosen to allow a large window for the button to be pressed. The Simulink block model is shown in Figure 8, and a plot of the implemented open loop step response can be seen in Figure 9.

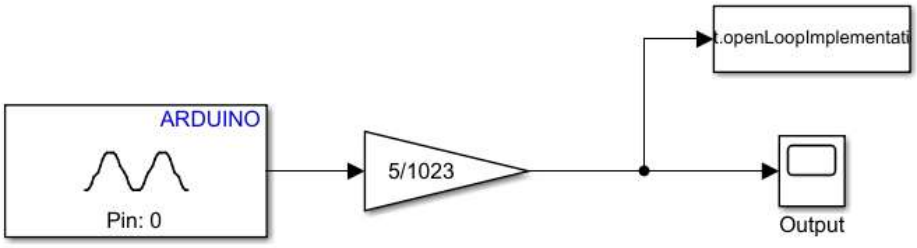


Figure 8: Open Loop Simulink Block Model

Since the block that reads the capacitor voltage is essentially using analogRead(), a conversion rate of 5/1023 needed to be used to convert the 10-bit output to a voltage reading. In conjunction with the simulated responses, an analysis of the ζ and ω_n values in the implemented responses can inform the actual values of the components in the RLC circuit, since there is always a slight difference in the nominal value and the actual value. The SSE is also very minimal (0.009 V), matching the simulated step response. Given a step input of 1.56 V, V_{max} = 1.789 V and T_S = 0.17 s. These give OS% = 14.68%, ζ = 0.521, and ω_n = 45.16.

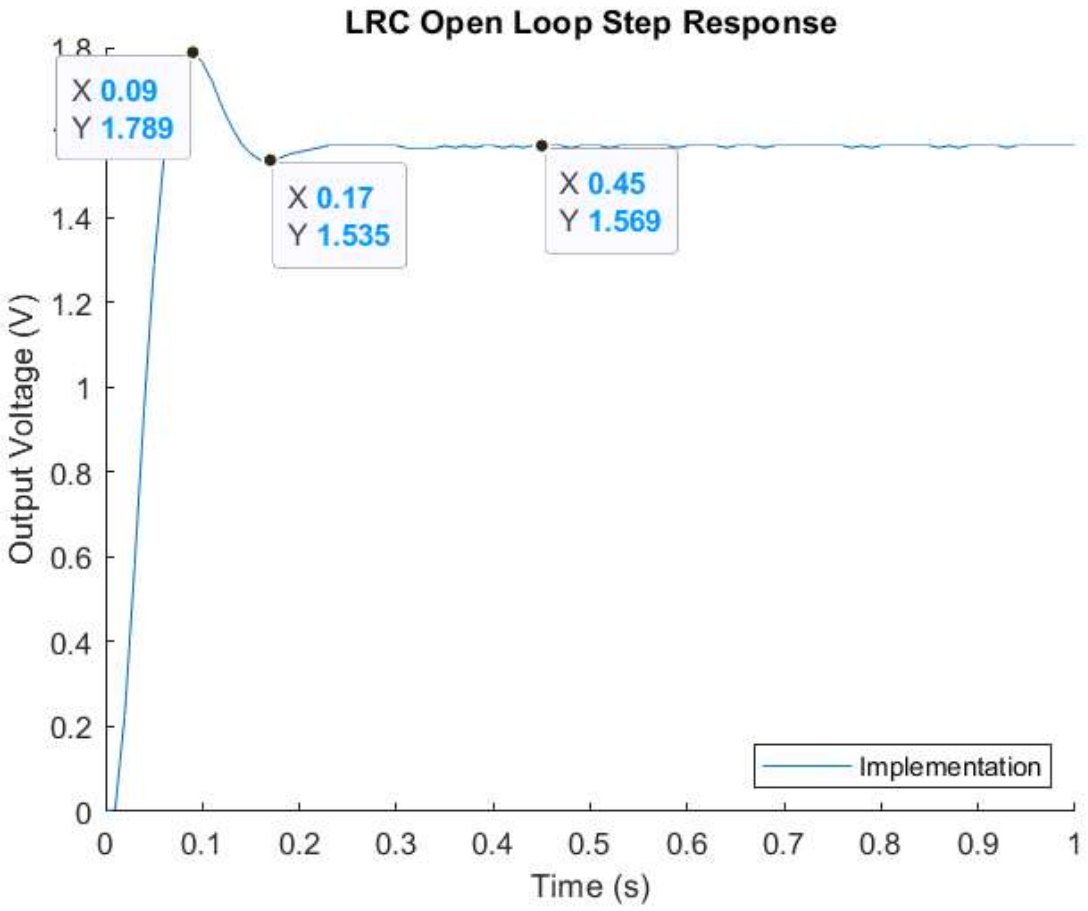


Figure 9: Simulated Open Loop Step Response

Closed Loop

To implement a feedback loop into the implementation, processing of the battery and output voltage by the Arduino is required. To do this, the battery is connected to pin A1 of the Arduino along with a 10 kΩ pull down resistor to ensure a clean reading of the battery voltage. To send the error signal to the RLC circuit, pin 8 was used. This new analogRead() and analogWrite(), for the battery and RLC circuit input, respectively, also require conversions into volts to make sure calculations and data are universal. For the conversion from volts to an analogWrite() value, a 255/5 conversion rate is used, since analogWrite() uses a 8-bit value. The Simulink block diagram and closed loop step response are displayed in Figure 10 and Figure 11, respectively.

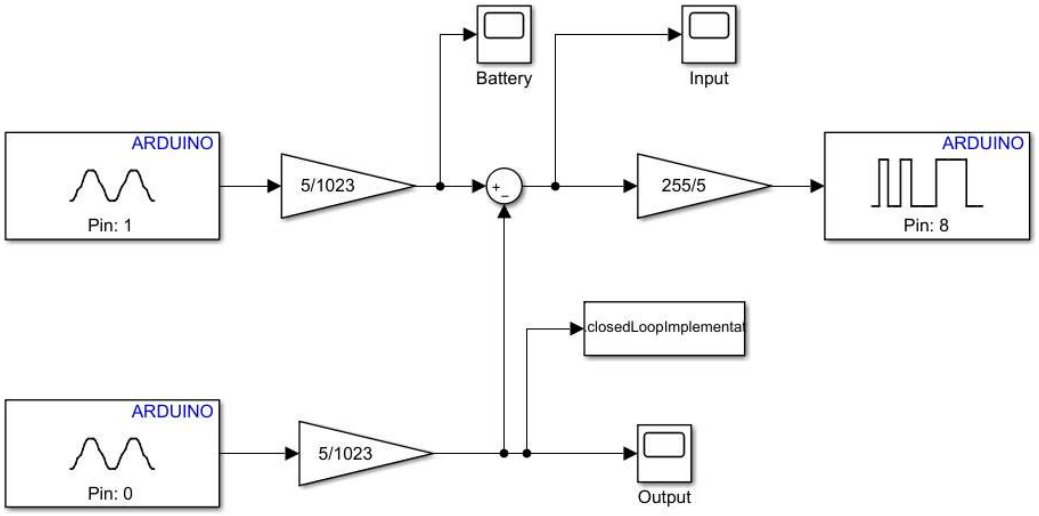


Figure 10: Closed Loop Simulink Block Model

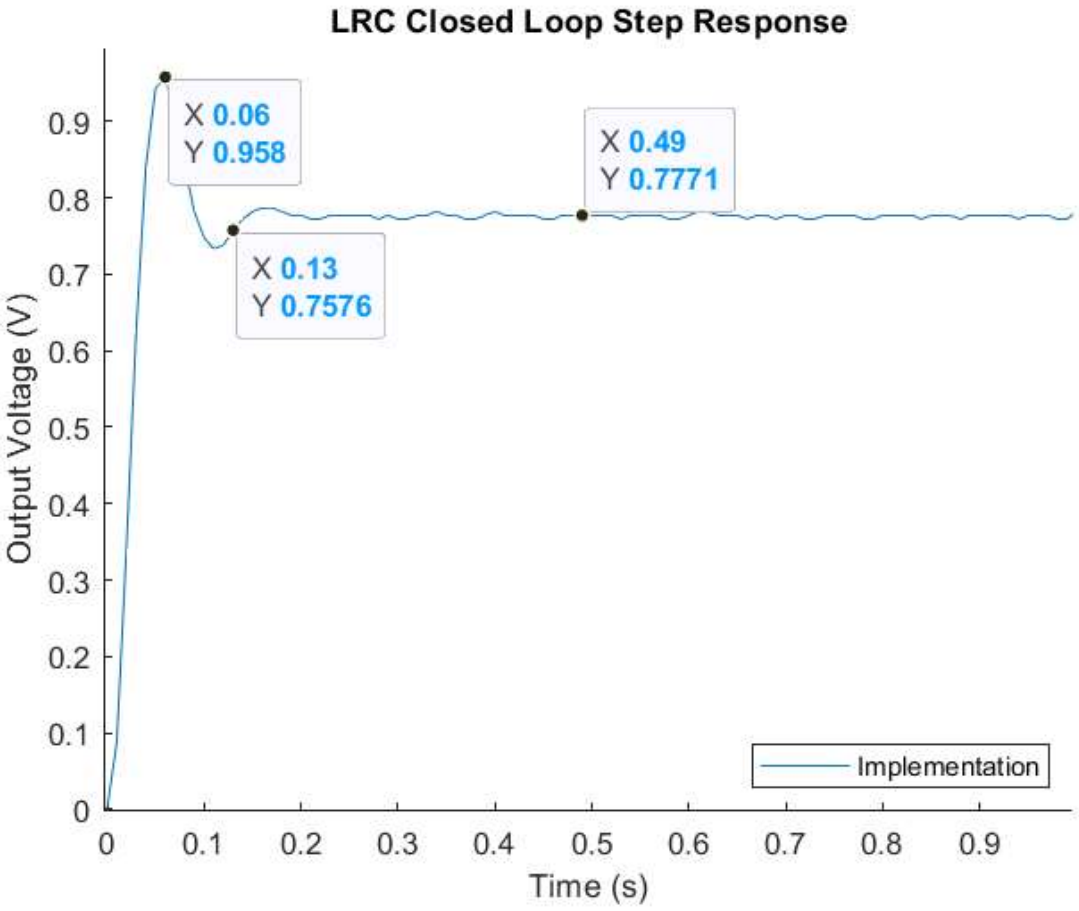


Figure 11: Implemented Closed Loop Step Response

Given V_{max} = 0.958 V (with a V_{SS} of 0.7771 V) and T_S = 0.13 s, the OS%, ζ, and ω_n for this response is 23.79%, 0.421, and 73.09, respectively. As with the difference between the simulated open loop and closed loop step responses, the implemented closed loop step response resulted in a higher OS%, lower ζ, and higher ω_n than the implemented open loop step response.

Closed Loop with Controller

The implementation of the closed loop with a PI controller required some work to get working correctly. For the closed loop system without a controller, the error never ran through a gain before going through the RLC circuit, resulting in an input that never amplified to more than the input (See Figure 11). When the PI controller is implemented, there is a pure gain of more than 1, and another gain with an integrator. This provides a problem when implementing the controller with an Arduino board. For sending and receiving signals to and from the board, positive voltage values are used. To be more precise, there is the analogWrite() function, which takes values from 0 to 255 (corresponding to 0 V - 5 V), and the analogRead() function, which returns values from 0 to 1023 (corresponding to 0 V - 5 V). The problem would occur when the output exceeded the input, wherein a negative error would be calculated and sent to the PI controller. This negative error would be okay in a circuit that included non-digital components, but since Arduino can't output negative values, a negative value given to analogWrite() would result in an underflow, and 5V would be sent to the circuit.

A similar error occurred for the high end values of the PI controller output. Since the output of the PI controller can get large due to larger values for K_P and K_I, a value larger than 255 can occur often, resulting in a constant 5V input into the system, which results in an output that immediately jumps to 5V. To overcome both of these issues, a saturation block was included in the Simulink block diagram to stop the controller from giving the analogWrite() unusable/unrealistic values.

The saturation block consisted of a lower and upper limit, and locked the controller output to one limit if it ever exceeded the limits. For this case, a lower limit of 0 and an upper limit of 80 was used: 0 because the lowest limit for analogWrite() is 0, and 80 because that was the desired SS of the output converted into the range of 0-255. The conversion 255/5 was used to create this result. After this adjustment, a stable behavior came out of the implementation.

The block diagram of this implementation can be found in the Appendix, and a plot of the step response is shown in Figure 12.

Given V_{max} = 1.613 V and T_S = 0.21 s, the OS%, ζ, and ω_n calculated for this response are 3.40%, 0.734, and 25.95, respectively. The response is more damped than expected, but still fits within the design requirements specified for the controller. Compared to the implemented open and closed loop responses, this is the most controlled response.

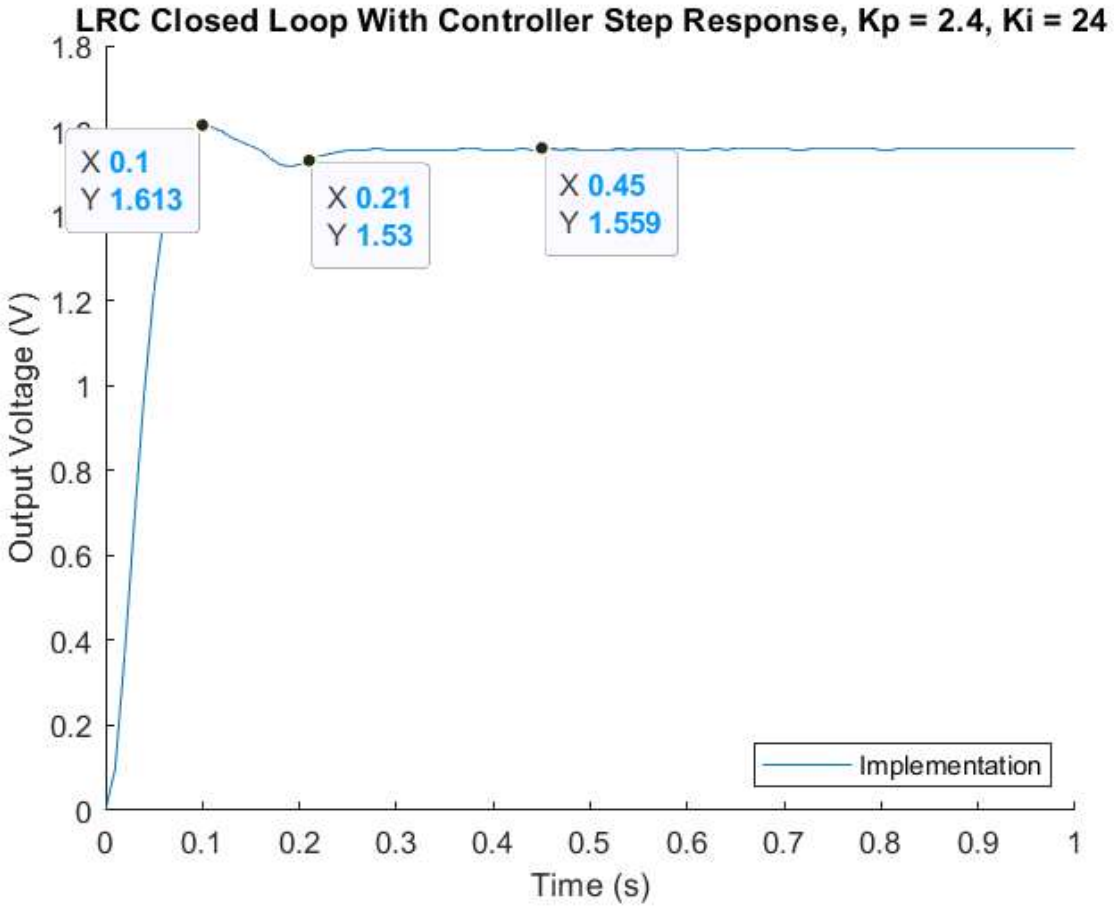


Figure 12: Closed Loop with Controller Step Response

Arduino Sketch

The Arduino sketch implementation consisted of copying the layout of the corresponding Simulink block diagram into the format of an Arduino sketch, which follows a typical coding format with functions, global and local variable declarations, and function calls. The input and

output pins were declared using global variables, and a varying number of variables were used in each phase of implementation, based on how many are needed to make the implementation function. These will be discussed in each section.

Open Loop

For the open loop implementation, double variables were declared for the output of the open loop system and the battery input. The sketch reads in the voltage from the battery and the output (doing a 5/1023 conversion in the process). The battery and system output were then printed to the Serial Monitor, where they were retrieved by a third-party utility and used to create a plot. The plot for the open loop step response is shown in Figure 13.

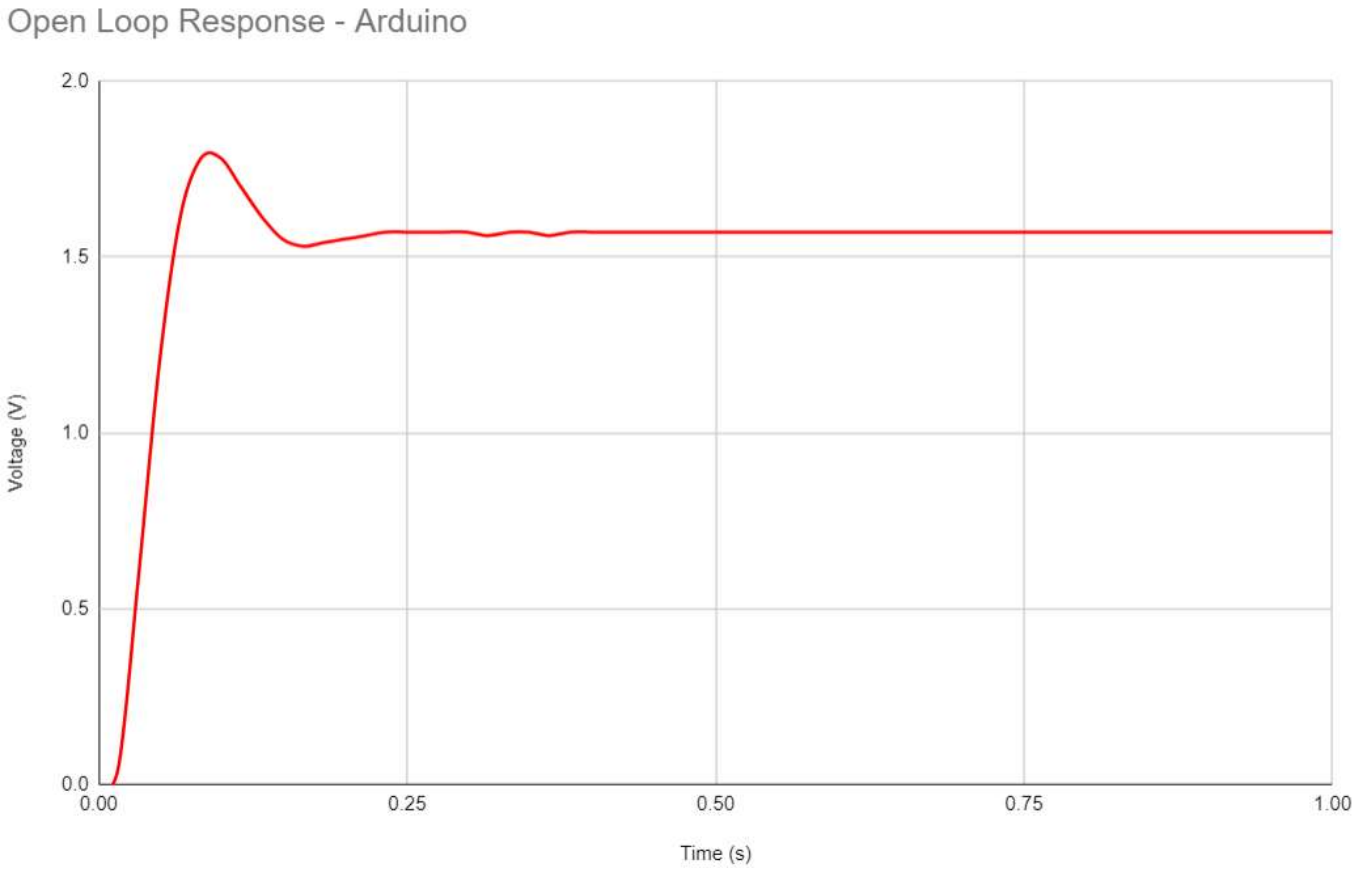


Figure 13: Open Loop Step Response - Arduino Sketch

I couldn't display the V_{max} or T_S normally like in the Matlab plots, but since this data was being processed in a spreadsheet, a couple functions, and some by-hand searching, were

utilized to find these values. For the open loop step response, a V_{max} of 1.78 V and T_S of 0.133 s were found, giving an OS%, ζ, and ω_n of 14.10%, 0.529, and 56.85, respectively. These results are expected when compared to the previous implementation's open loop step response.

I noticed that a different result is possible if the battery voltage read in is written to the RLC circuit like in the closed loop system, instead of the battery being connected directly to the RLC circuit. The different result is shown in Figure 14, and shows a dampened response compared to Figure 13. This may be a result of the Arduino not providing the right input signal to allow the same overshoot as the original method.

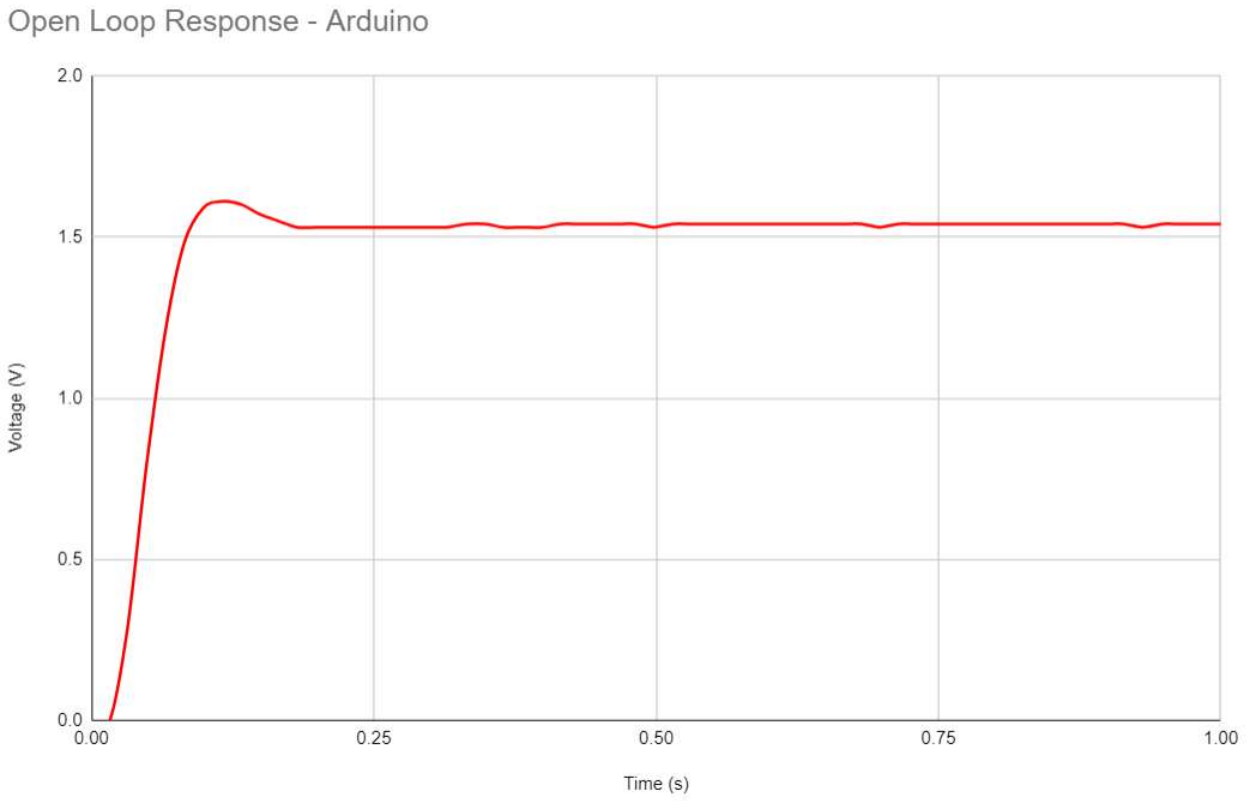


Figure 14: Open Loop Step Response with Different Result - Arduino Sketch

Closed Loop

Implementing the closed loop system involved the inclusion of the variable error to store the difference between the battery input and the output of the system. This error is then sent to the RLC circuit, and the corresponding step input is shown in Figure 15. Looking through the data gives us a V_{max} of 0.98 V (V_{SS} = 0.77 V) and a T_S of 0.15 s. The OS%, ζ, and ω_n of this data is then 27.27%, 0.382, and 69.81, respectively. This is similar to previous results, and still show a large OS% and SSE that is not desired for this system.

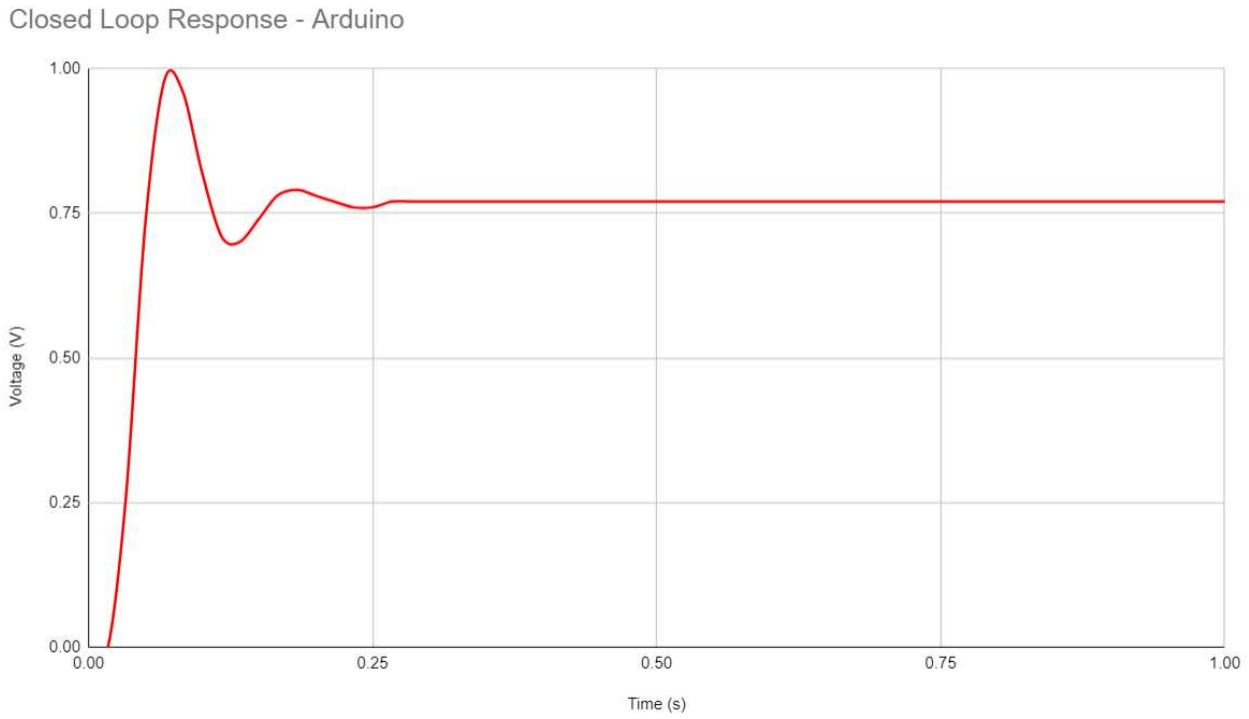


Figure 15: Closed Loop Step Response - Arduino Sketch

Closed Loop with Controller

The same problems with unusable analogWrite() values that occured in the Simulink/Arduino section also appeared in Arduino sketch. Due to this, an upper and lower limit to the controller output was also incorporated into the Arduino sketch. This was done using an if and else if statement to check if the controller output was less than 0 or greater than 80, just like the saturation block in the Simulink block diagram.

The PI controller was incorporated through a function called computePID() that accepted the output of the system as an argument and returned an input signal for the system. Multiple

global variables for this function were needed, but the most important ones for customizing the controller were the Kp, Ki, and Setpoint variables. The Setpoint variable was the target signal that the PI controller was going to match the output to. In the case of the project, Setpoint was set to the battery voltage because the system design demanded that the system output be equal to the system input with no SSE. computePID() included calculations for the Derivative portion of the PID controller, but to avoid that section, Kd was made equal to 0 instead of removing the related code in the function.

The flow of the program was as follows: the battery and system output voltage were read and converted to volts (5/1023), Setpoint was set to the battery voltage and the system output was passed to computePID(), computePID() returned a system input signal, which was then converted to 8-bit (255/5) and sent to the RLC circuit with analogWrite(). The resulting plot of this sketch can be seen in Figure 16.

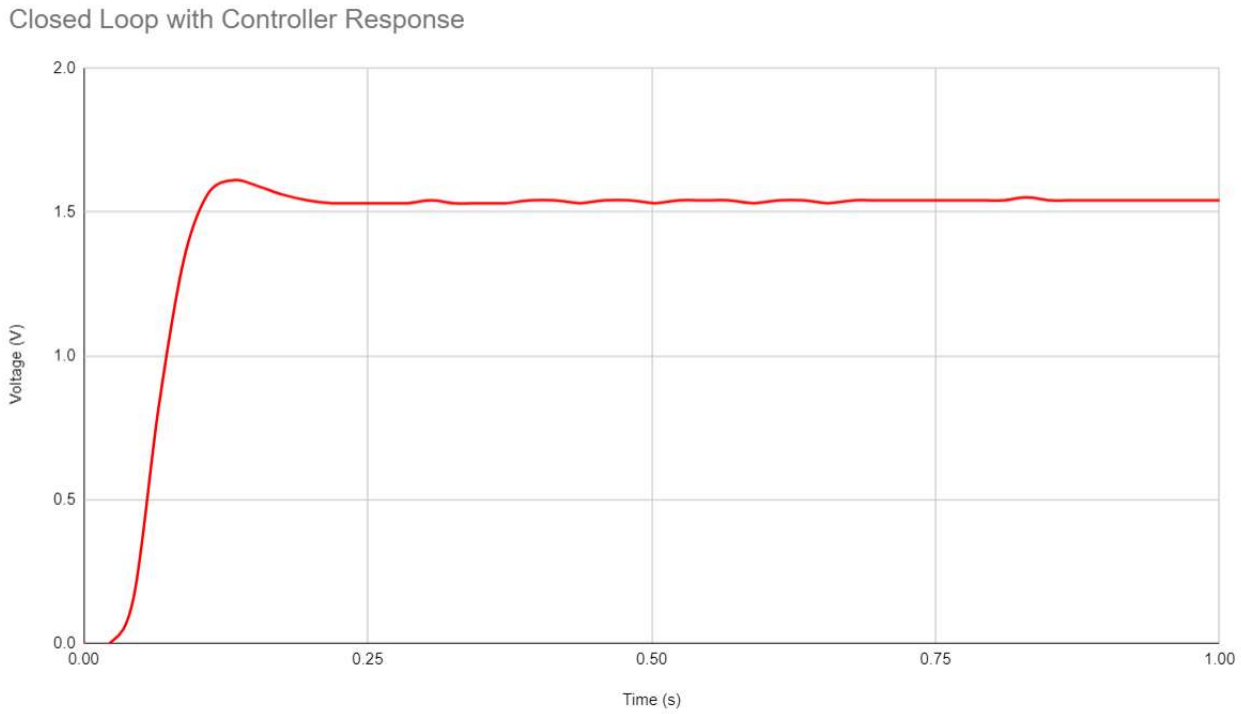


Figure 16: Closed Loop with Controller Step Response - Arduino Sketch

Finding V_{max} = 1.61 V and T_S = 0.153 s, that gives an OS%, ζ, and ω_n of 3.21%, 0.738, and 19.29, respectively. Just like the Simulink/Arduino implementation, the system seems to be more damped and has a great step response to the step input while following the requirements specified before.

Conclusions

Throughout the implementations, there were slight differences in the values of ζ and ω_n. Given the theoretical equations for ζ and ω_n,

$$\omega_n = \sqrt{\frac{1}{CL}}$$

$$\zeta = \frac{R + R_{eq}}{2}\sqrt{\frac{C}{L}}$$

it's possible that some of the components' actual values are slightly different than their nominal values. It's also possible that different interferences in the creation of the step responses, like the Arduino or noise somewhere in the RLC circuit, might be interfering with the step responses. Although these are possibilities, the characteristics of these implementations are very close to the simulations and can be counted as being equal in terms of performance.

Open Loop

Other than slight variations in the values of ζ and ω_n, the simulations and implementations functioned so similarly that it would be difficult to distinguish between the two. This is probably due to the open loop implementations having the battery directly power the circuit instead of the Arduino reading in the value of the battery voltage and sending that to the circuit using PWM signals, since the simulation is expecting an step input that's capable of offering an analog signal instead of a digital one.

Closed Loop

Again, the same thing as the open loop can be said for the closed loop. The simulations and implementations are so similar that they're hard to tell apart, but in this case, the system was powered using the PWM signal from the Arduino. This difference in input that provided a

different result in open loop, but offers an extremely-similar result in closed loop, is fascinating and I'd like to know why it works.

The closed loop for the Simulink/Arduino never required a lower limit for its error signal, which fascinated me because I needed to include one in the Arduino sketches. This problem was explained in the Arduino Sketch subsection of the Implementation, but the error message would sometimes be negative, which doesn't "play" well with the analogWrite() function's input range of 0-255. In this case, I made an if statement that would make error = 0 if it ever became negative so no random 5V signals would be sent to the RLC circuit. Other than that difference, the simulation and implementations performed identically.

Closed Loop with Controller

This is where it was more interesting, as the controller designed for this project had simulated values of OS% = 10.4% and T_S = 0.351 s. This is in stark contrast to the implemented system, which had OS% = 3.40%, T_S = 0.21 s and OS% = 3.21%, T_S = 0.153 s for the Simulink/Arduino and Arduino Sketch implementations, respectively. The implementations had a better response than the simulation, which was not expected at all. This may be due to the saturation limits imposed by the saturation block/system input checks in the implementations to handle any wild controller outputs. This may also be due to the differing values of the components in the actual RLC circuit, and how they might actually help dampen the oscillatory behavior of the system when it's closed loop with a PI controller.

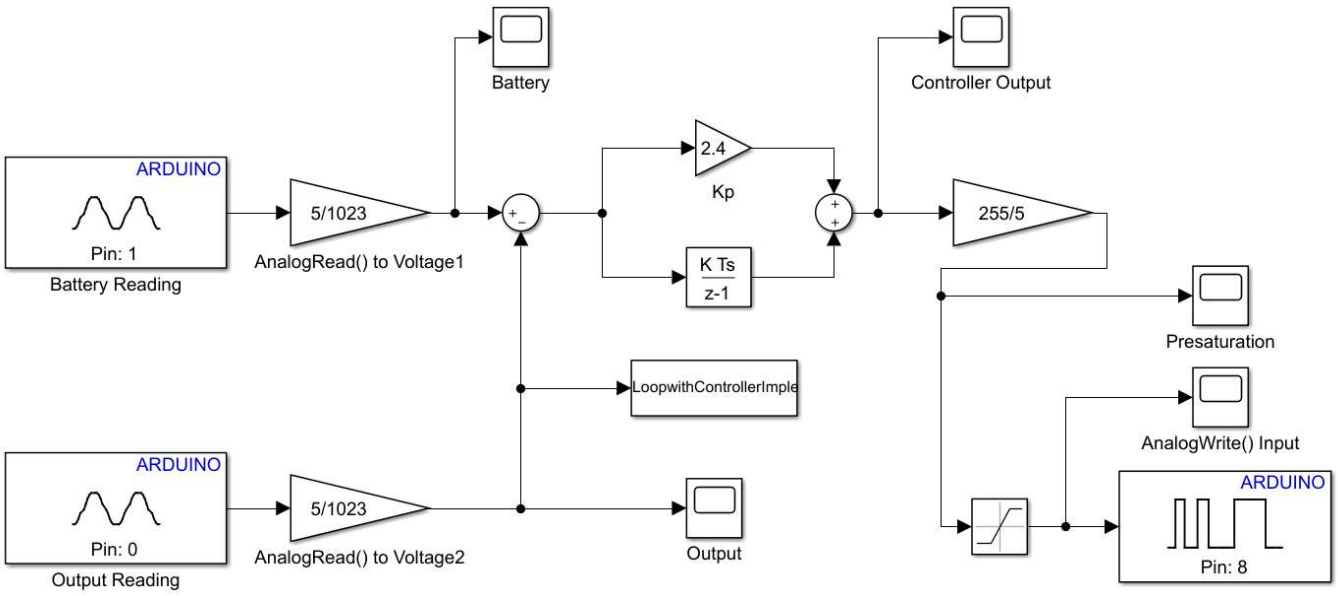
In terms of the difference/preferences between Simulink/Arduino and Arduino Sketch, I found the Simulink implementation to be easier to understand due to its use of interconnected blocks that produce a visible flow and connection between all of the conversions and components in the system. This block diagram system also can be clunky at times, as blocks need to be placed so as to not crowd the window, they need to be entered to change and verify values, and Simulink takes a while (10 s - 15 s) to run the code on the Arduino board. This code also only lasts as long as the specified runtime, so there's a limited-time window before you need to run the code again and wait another 10-15 seconds.

In contrast, the Arduino Sketch is written similarly to any other high-level programming language. The connection between each part of the system can be more confusing to keep track of in this format, but the code is quicker to write, and faster to compile and run. The code also runs as long as you want, enabling quick successive retrievals of data without wasting time. The Arduino Sketch can also support included libraries that may improve on the ease of use

and functionality of the sketch, which might be helpful for more complex projects or for components that need a library to work at all.

Appendix

Simulink/Arduino - Closed Loop With Controller Simulink Block Diagram



Arduino Sketch - Closed Loop with Controller Arduino Sketch

```
// Riley Ruckman
// TCES 455, Au20
// Project - RLC circuit with unity feedback and PI controller

const int batteryPin = A1;
const int outputPin = A0;
const int inputPin = 8;

double battery, Controller2Input;
```

```
//PID constants
double kp = 2.4;
double ki = 24;
double kd = 0;

unsigned long currentTime, previousTime;
double elapsedTime;
double error = 0;
double lastError = 0;
double input, output, Setpoint;
double cumError = 0, rateError = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(batteryPin, INPUT);
  pinMode(outputPin, INPUT);
  pinMode(inputPin, OUTPUT);
  Serial.begin(9600);

  battery = analogRead(batteryPin) * ((double)5/1023);
  output = analogRead(outputPin) * ((double)5/1023);

  Setpoint = battery;
}

void loop() {
  // put your main code here, to run repeatedly:

  // Reads voltage of battery in Volts
  battery = analogRead(batteryPin) * ((double)5/1023);

  // Sets Setpoint to the current voltage of the battery
  Setpoint = battery;
```

```
  // Reads output of RLC circuit in Volts
  output = analogRead(outputPin) * ((double)5/1023);

  // Calculates controller output given the current output of the RLC circuit
  // and the current voltage of the battery
  Controller2Input = computePID(output) * ((double)255/5);
  delay(10); // Give time for computePID() to calculate

  // Saturation Block
  if (Controller2Input < 0) {
    Controller2Input = 0;
  } else if (Controller2Input > battery*((double)255/5)) {
    Controller2Input = battery*((double)255/5);
  }

  // Sends Controller2Input to RLC circuit
  analogWrite(inputPin, Controller2Input);

  // Prints the current execution time in milliseconds
  Serial.print(millis()); Serial.print("\t");
  // Prints voltage of battery in Volts
  Serial.print(battery); Serial.print("\t");
  // Prints output of RLC circuit in Volts
  Serial.print(output); Serial.print("\t");
  // Prints Controller2Input value in Volts
  Serial.print(Controller2Input * ((double)5/255)); Serial.println();
}

// Computes PID output based off input, Setpoint, and the set gain values Kp, Ki, and Kd
double computePID(double inp){

  currentTime = millis();                          // get current time
```

```
  elapsedTime = (double)(currentTime - previousTime);   // compute time elapsed from previous
computation

  error = Setpoint - inp;                          // determine error
  cumError += error * elapsedTime;                    // compute integral
  rateError = (error - lastError)/elapsedTime;          // compute derivative

  double out = kp*error + ki*cumError + kd*rateError;   // PID output

  lastError = error;                               // remember current error
  previousTime = currentTime;                          // remember current time

  return out;                                     // have function return the PID output
}
```