Stabilizing Fiber Resonators

Collin Wilkinson
Coe College Iowa

Abstract:

This report summarizes the control loop electronics that were developed and modified to control the thermal properties of a fiber resonator as well as describing the methods used to suppress all external effects on the fiber. It also summarizes the effects of responsivity from both analog and digital circuits and they're different developmental processes all within the scope of gravitational wave detection.

Introduction:

In the early 1900's Albert Einstein mathematically quantized General Relativity, however it did predict a loss of energy that would propagate across space-time in the form of a wave. This wave caused by gravitational attraction would cause space-time to slightly shift, a shift that modern day gravitational wave (GW) detectors can detect (I.E. LIGO, LISA, AET, etc.). The GWs we've been able to detect so far have come from Black Holes however, it is possible to have get signals from several other objects including pulsars and very massive supernova.

These gravitational waves manifest themselves over large areas in space by slightly change the distance between two points in space. These gravitational waves were first theorized in 1917 they remained indirectly detected until LIGO in 2015 confirmed a direct measurement of a gravitational wave[1][2][3][4].

To be able to detect the gravitational waves from black holes your sensitivity has to be in the range of a ten-thousandth of a proton over a range of 4 Km, a barrier that has currently been reached - however to reach further sensitivity the main sources of noise have to be repressed. In Figure 1 you can quite clearly see that a large source of noise is thermal related noise as well as quantum and optical noise. This is because in a standard Michaelson interferometer the only introduction of noise has to come from the laser the lens or the environment. So to be able to counteract these effects many measures are taken, the one this report concerns though is to counteract small frequency changes in the laser. These small changes are called frequency drift, a natural phenomenon that happens in all lasers.

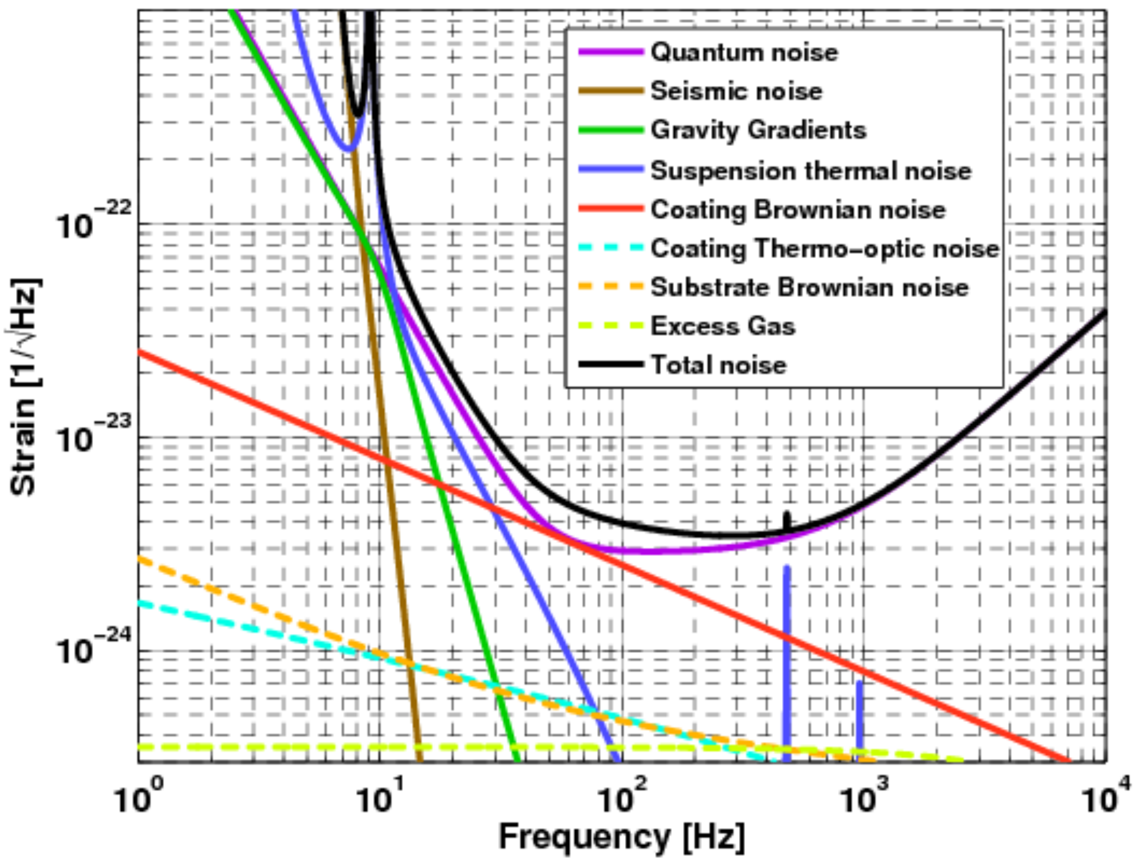


Figure 1 The error curves in today's LIGO setup

The way this is stabilization is preformed (to prevent frequency drift) is by using a method called the Pound-Drever-Hall technique with a laser cavity to counteract any frequency drift (see figure 2). This method is highly reliant on the resonator used in the setup. The standard Fabry-Perot cavity was replaced, in out setup, with a circular ring cavity which allows for a much higher finesse (the free-spectral-range divided by the bandwidth at its resonant frequencies). With this dramatically higher finesse you are able to very accurately control the frequency of the laser over long periods of time using control loops (discussed in detail in a future section). However this control loop is highly reliant on the stability of the resonator, in our case the resonator is subject to thermal, acoustic, and mechanical distress[7].

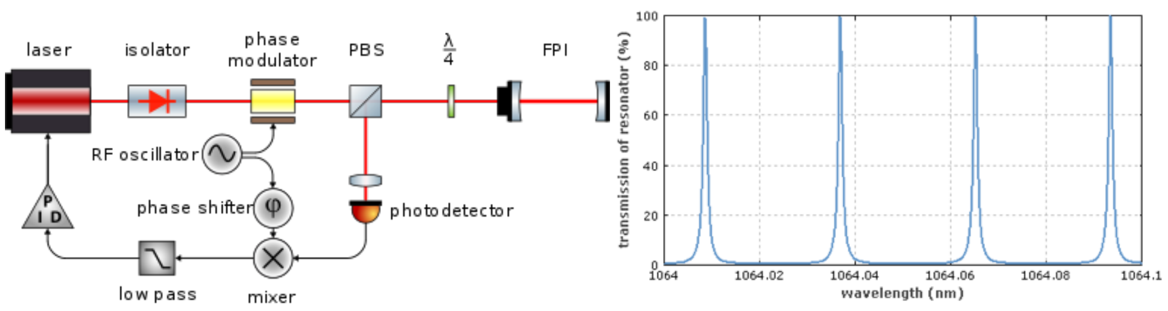


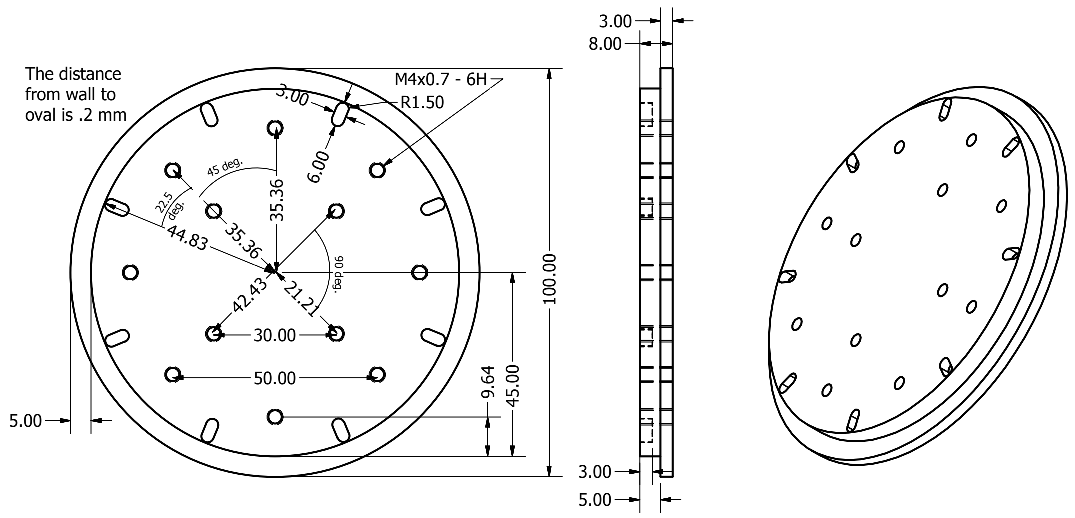
Figure 2: (Left) A typical example of a Pound-Drever-Hall setup, the laser emits and passes through the Faraday isolator and a modulator. The laser than goes through a resonator and returns to interfere with its self after the beam splitter and is then read by a photodetector where the modulation from before can get transformed into an error signal that is fed back into the laser as a voltage allowing for you to correct any drift. (Right) Typical example of wavelengths let through the resonator vs percent transmission, the idea is to contain all frequencies of the laser inside of one of the peaks of the resonator so you can keep a very precise laser frequency for a long period of time.

The rest of this report deals with the concept of stabilizing the resonator so that the laser can lock and hold its stability. This will be achieved through a novel design, some basic classical measures, and more control loops.

Design:

To be able to control this resonator for long period of time and to keep the frequency the same over long period of times you have to have a very consistent environment. The resonator environment is highly dependent on motion, sound, and temperature because any vibration differences in the atoms of a glass structure can cause a temporary change in the refractive index therefor changing the overall behavior of the resonating behavior.

A design was created to be able to counteract the 3 most common sources of instability for a fiber resonator (sound, thermal fluctuations, and movement). It started with a simple base design show below in a CAD drawing. It's consists of a cylinder made of brass that is lowered on top of an aluminum base. In between the base and the plate there is a place large enough to fit a Peltier element and in the cylinder there are places to install sensors in order to closely categorize the heat.



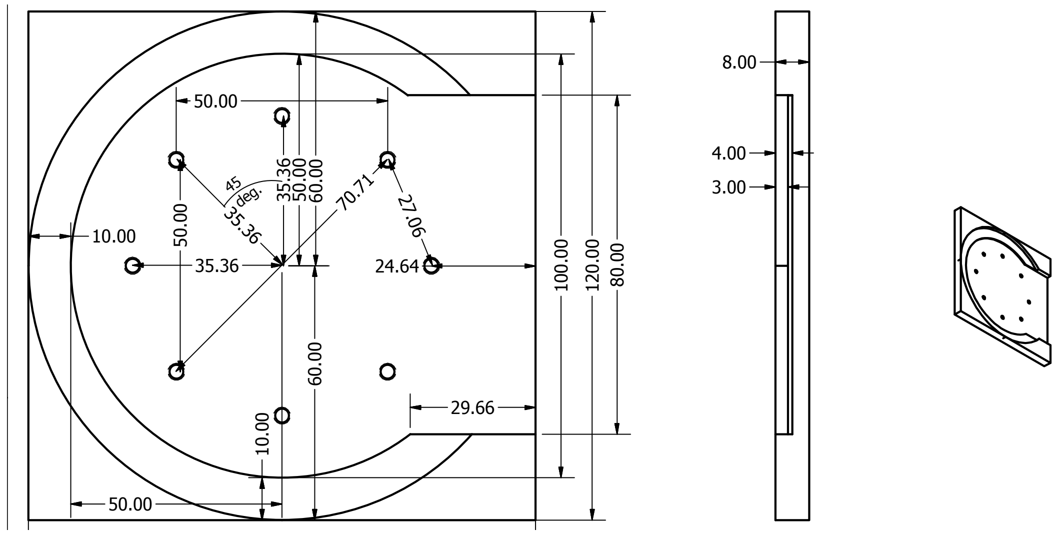


Figure 3 (Top) The top cylinder, consisting of ovals for sensors and screw holes to clamp it down. (Bottom) the bottom plate also consisting of screw holes a place for the Peltier and a cutout so that all the splices and cables can be managed underneath the plate

These parts were machined and then placed inside of a vacuum (to counteract noise interactions) and was placed on soft rubber balls that counteract any dramatic movement, leaving the only problem (and by a long shot the most difficult) long term temperature stabilization. We approached this two distinct ways, with analog signals and digitally.

Control Loops:

Before the design of the circuit can be drawn it is best to first consider the general structure of a control loop. It is generally accepted that the structure of a control loop is as shown in the figure below. This is the structure used in everything from optics to robotics to thermal control. In order to design the best control loops it is best to consider each element in the control loop and make sure that there is a corresponding part in your system.

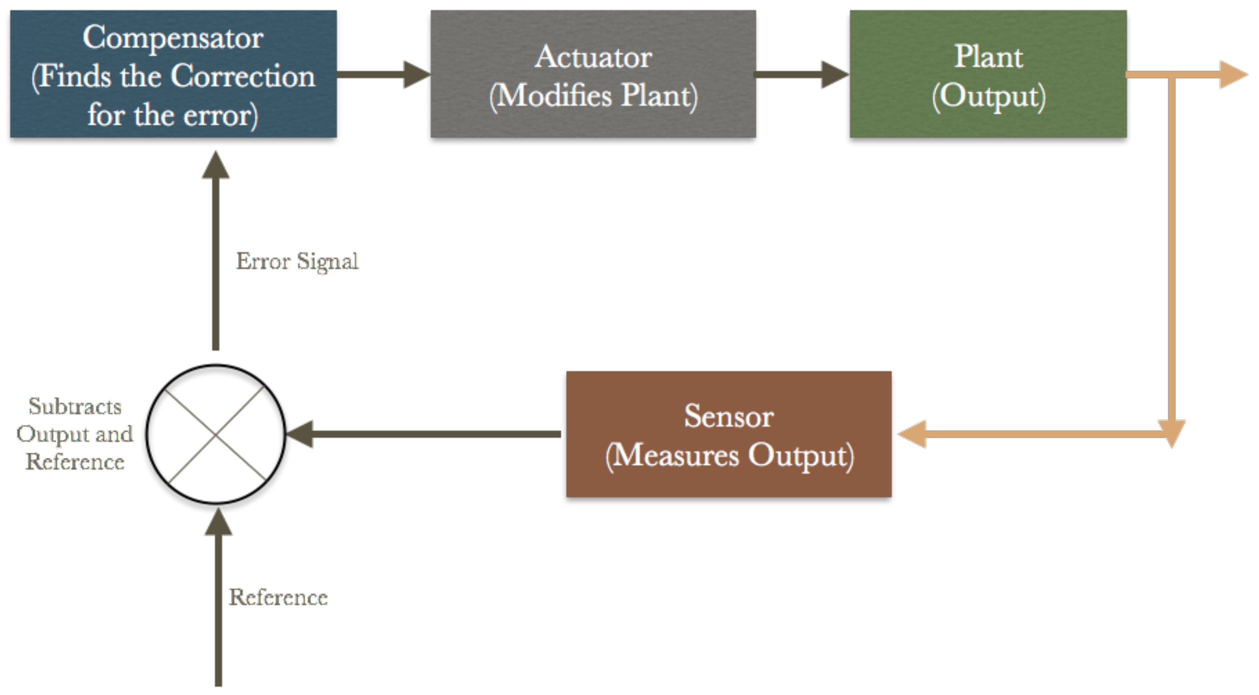

Compensator (Finds the Correction for the error)
Actuator (Modifies Plant)
Plant (Output)
Error Signal
Subtracts Output and Reference
Sensor (Measures Output)
Reference

Figure 4 A standard control loop from theory.

Analog:

In the analog control system, we started with the sensor and used a resistor whose resistivity is solely based on temperature. Then the mixer signal (the ⊗) is the Wheatstone bridge and the filters we added to clean and optimize the signal. The error signal is then sent to the output signal (compensator) which in turn sends the signal to the actuator which is what we refer to as the Peltier driver. The plant then is the Peltier itself. All of these signals are contained within two distinct boxes (besides the detector and the Peltier) the monitoring box and a Peltier driver box.

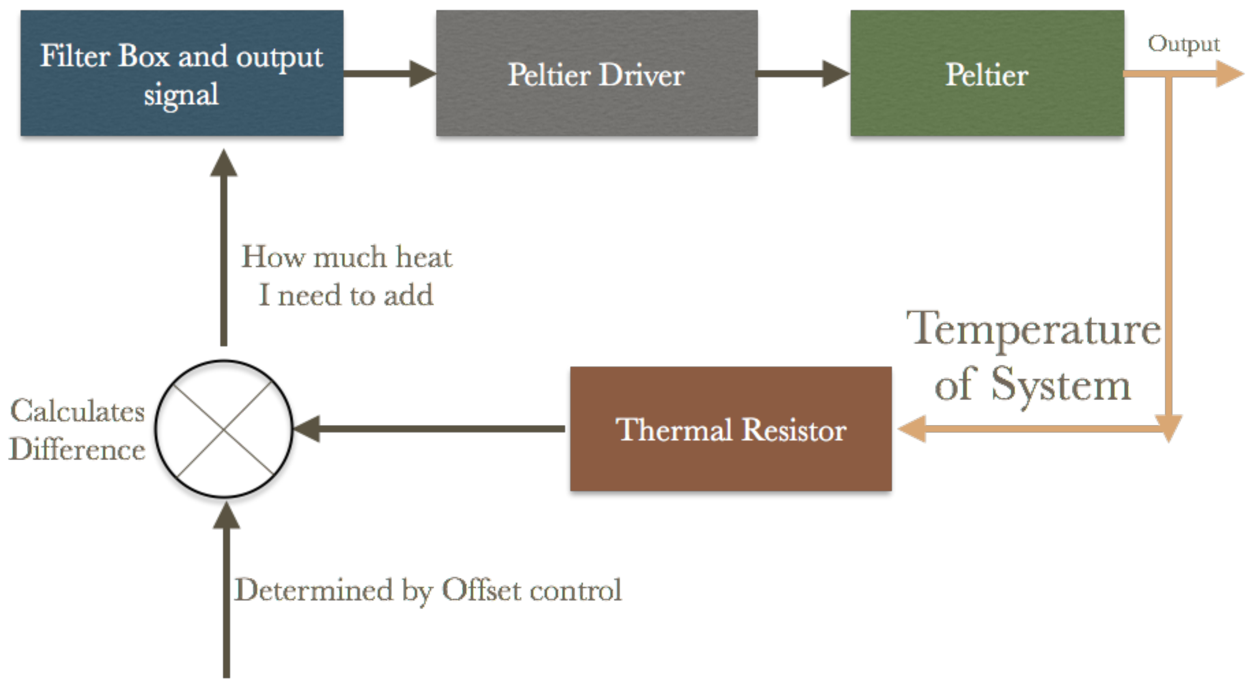


Figure 5 The example control loop we implemented to control the temperature of our system

A potentiometer was added to the general control loop so that there was an easy method to control the reference signal, it's worth noting that the minimum temperature it could reach is 18.1 C due to the fact that 18.0 C is the point in which water begins to condensate on the fiber. We then set the max temperature to 30 C because that is the temperature we started seeing issues with other items in the vacuum chamber. An output port was also added to view the V[Temp_t-Temp_0] (where Temp_t is the temperature desired and Temp_0 is the temperature in which the offset is set to reach). This circuit was then placed inside of what we called our monitoring box. The voltage at this output was then ran in parallel to an oscilloscope and to the Peltier driver.

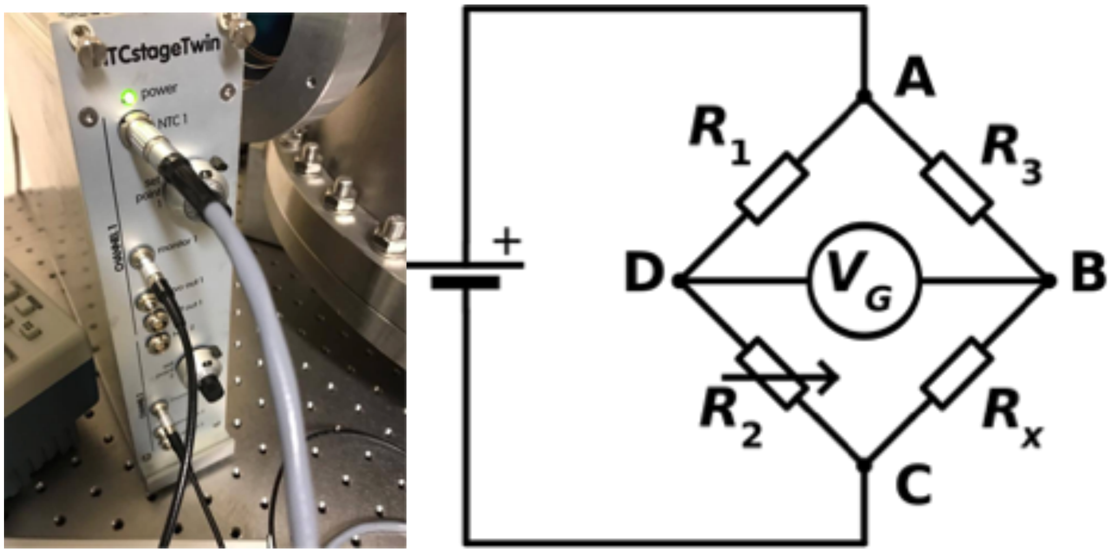
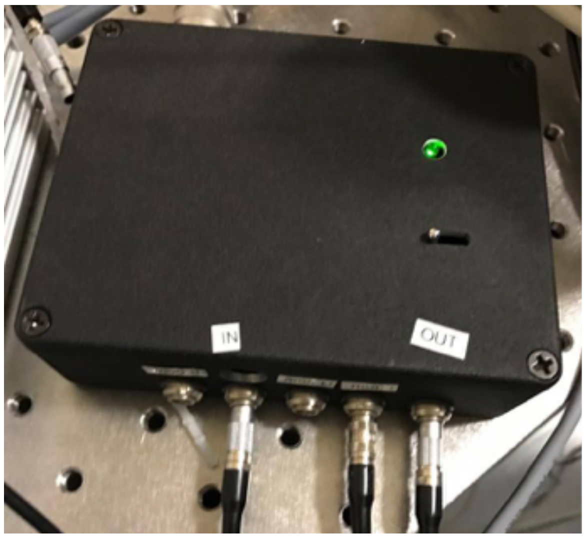


Figure 6 (Left) The final control box. (Right) The Wheatstone bridge where the thermal resistor is Rx and the potentiometer is R2. There is a potential difference between B and D, that is proportional to Rx.

The Peltier driver is another circuit that includes optional filters to optimize the transfer function of our system as well as a current and voltage monitor. This Peltier driver is then hooked up with the Peltier element and drives it to the temperature as set by the monitoring box. To optimize this circuit however we must first measure the transfer function.

Before we continue on to discuss transfer functions, a peltier elements behavior must be explained. A Peltier is a thermoelectric cooler, that utilizes solid state effects to either heat or cool an object. The effect works by causing heat to flow across two electrical junctions with current flowing through one of them absorbing the heat from the other. This behavior can be hard to categorize and work around that's why we used transfer functions to describe it.

Figure 7 The Peltier driver consisting of ideal filters with monitors.



A transfer function is a useful idea in categorizing and characterizing filters and analog behavior. It's the concept that signal X goes into a black box and signal Y comes out, you can then measure the magnitude of such an idea (telling you about the gain of the signal at different frequencies) and you can learn of the phase of the signal (telling you about the stability of your function at a variety of frequencies) (see equations below) [8][9].

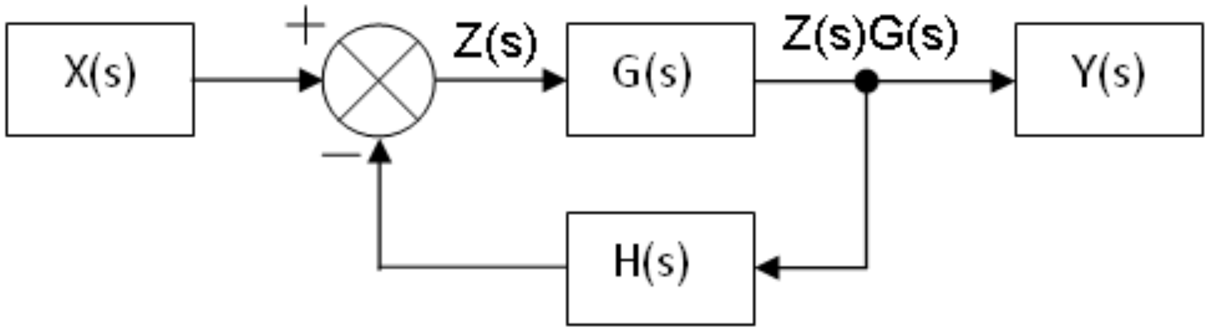


Figure 8 A standard feedback loop presented in box form where the output of G(s) plays a role in the outcome and the modification of the next sample.

Our initial transfer function is shown below.

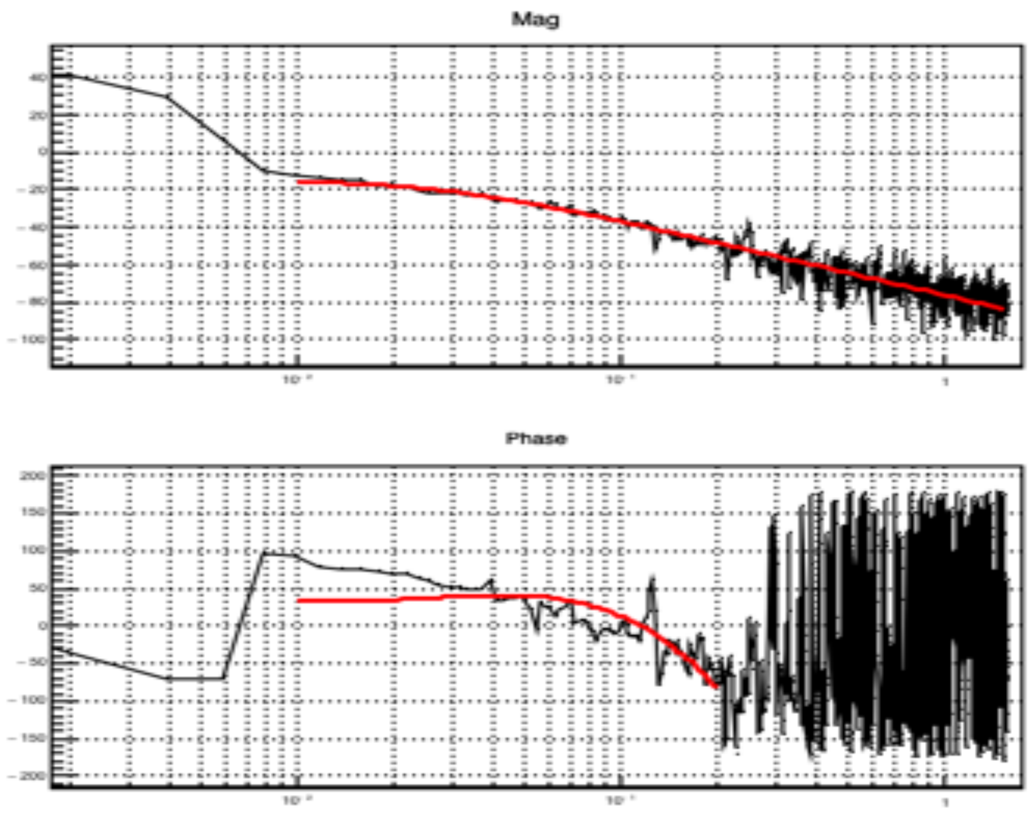


Figure 9 (Top) our initial transfer function (with no filters) we were looking to fit it after the initial points and before the instability. (Bottom) a measure of the phase it becomes unstable when it reaches positive or negative 180 degrees, because it flips the sign in our voltage to the Peltier. You can see this happen at approximately 0.2 in the phase diagram above.

The fit (in red) is then a value we can use to multiply filters with to try to attain as much of our transfer function above 0 dB as possible before we run into an instability. We then proceeded to multiply the fit by different filter's transfer functions to try to optimize the behavior of the transfer function of the entire system. So once multiplied by 2 low pass filters and a time delay filter. We can get a significantly improved transfer function as shown below.

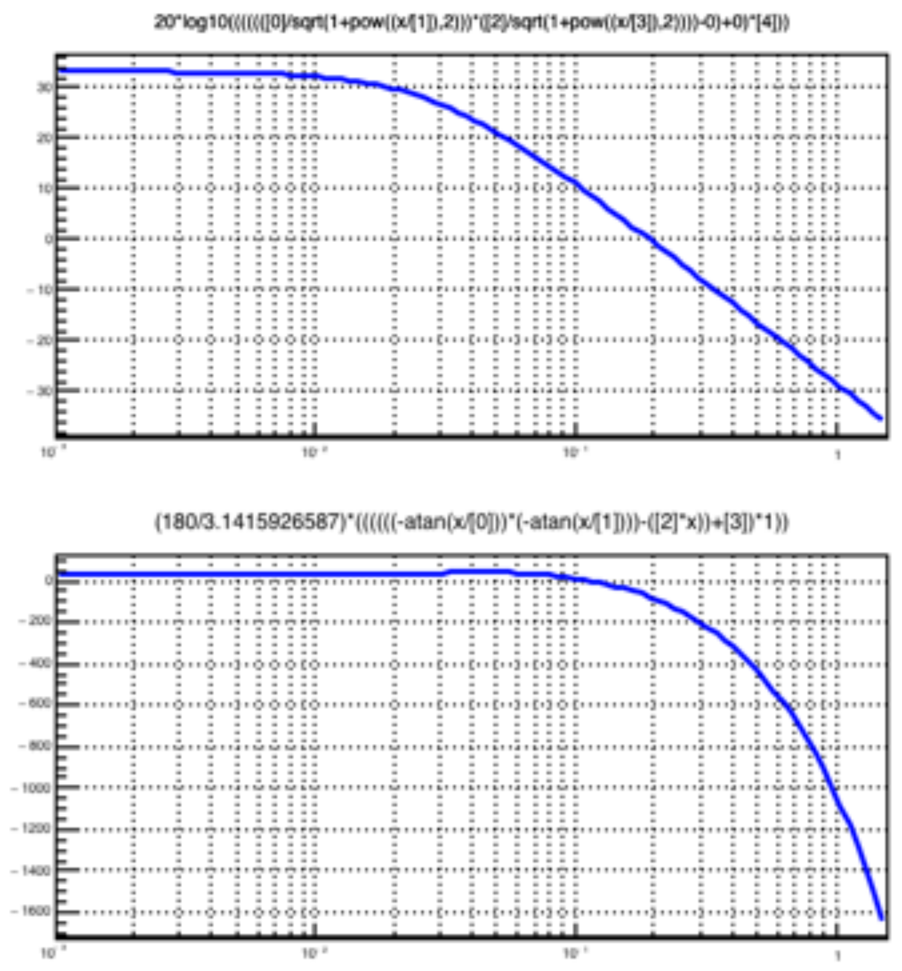


*Figure 10 Our model Transfer functions consisting of our fit*Low Pass Filter^2*A time delay you can see we get a very nice gain for our values before the point of 0.2 where we were before losing control.*

This transfer function is rather nice however this means (because it's instability at 0.2 Hz) it lacks the ability to quickly correct for any fluctuations over 0.2 Hz meaning it's good for slow changes and lacks the ability to quickly correct for fast changes. For thermal interference this is fine however not ideal. At this point we could either attempt to continue adding filters and cleaning and optimizing the signal or another idea could be tested, knowing the ability of digital electronics and micro controllers it was decided to try using microcontrollers for this task.

Digital:

An alternative method was designed using an implementation that is digitally based. In this case we use a very similar system but instead of a monitor box we switch it out with a digital box powered by an Arduino Nano. This also allowed for us to replace the standard analog filters with more accurate digital filters.

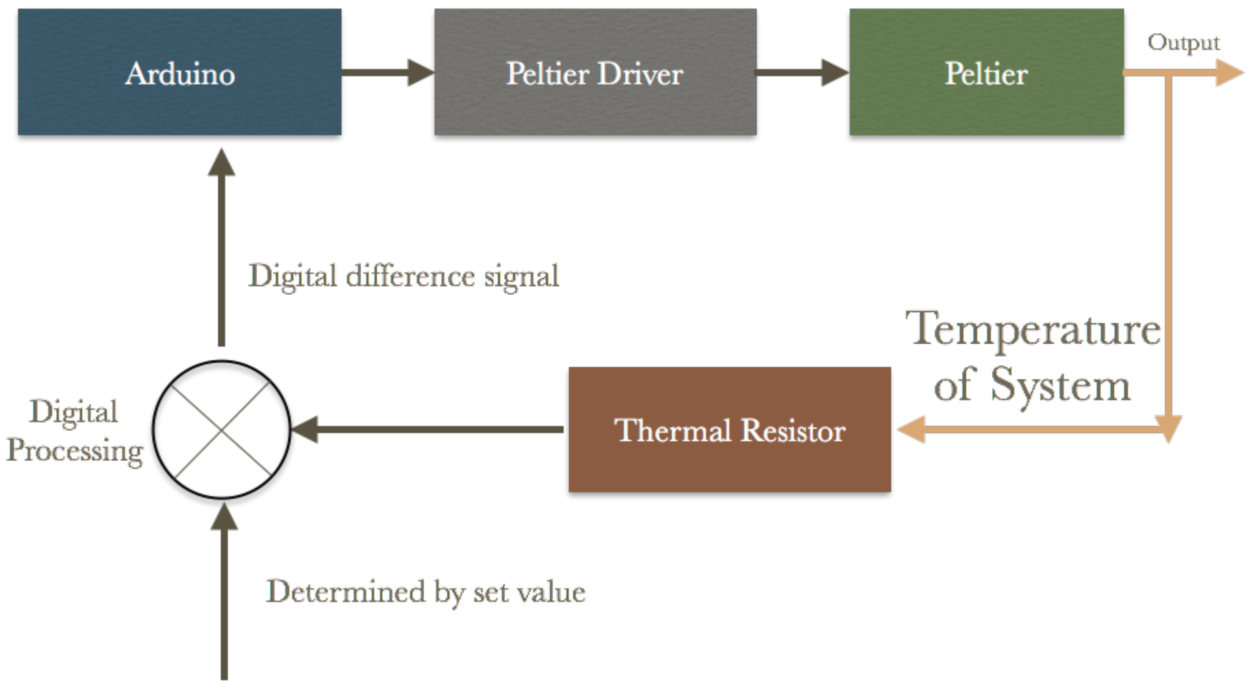


Figure 11 Digital control loop

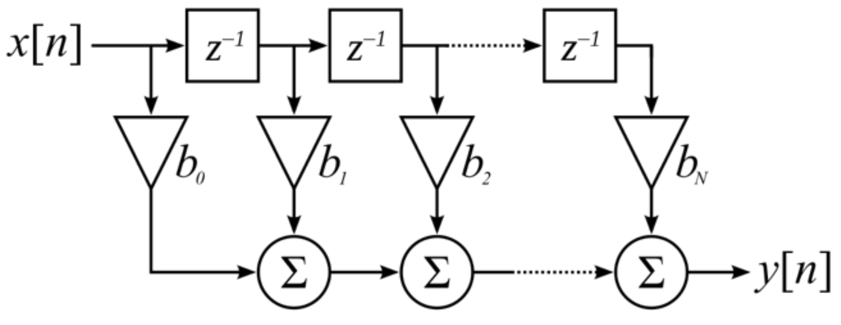


Figure 12 An example weight function that consists of a series of weights multiplied by some entries and summed to create the output.

Digital filters are hard to design because they're purely numerical functions. The way numerical filters work consists of adding a series of weights multiplied by the inputs (see figure below). These weights were determined using MATLAB's designfilt functions. The whole circuit then had electronics designed to act on the Arduino. A Wheatstone bridge was built in to be able to read the resistor, a ADC (analog digital converter) and a DAC (digital analog converter) were added to control analog input and output signals. This whole circuit also included an LCD screen so that the output and the filtering can be easily monitored.

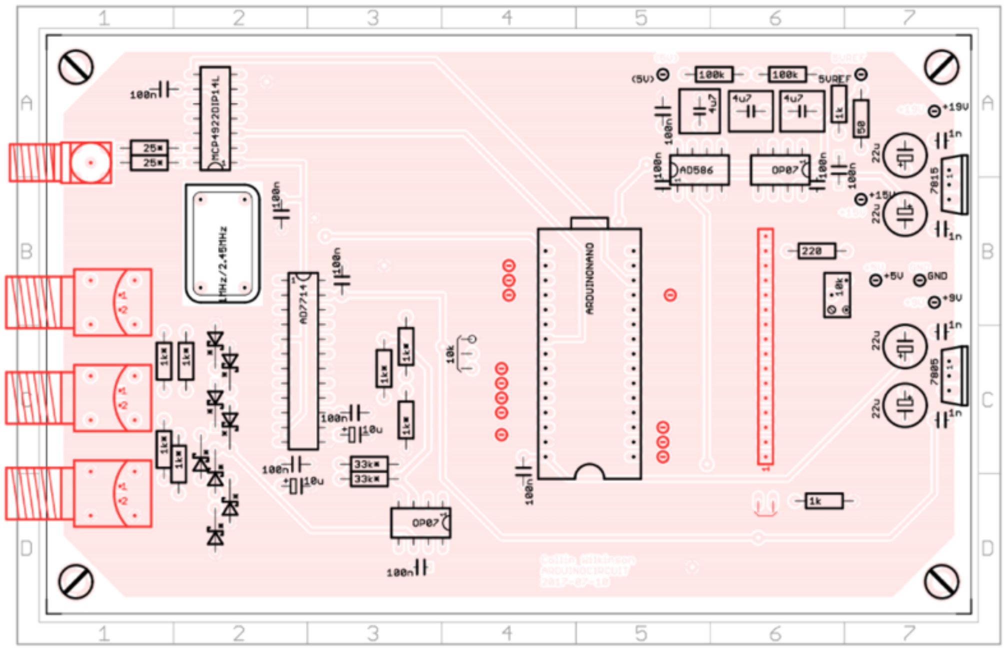


Figure 13 Our finished circuit design for the digital control circuit.

Once all of these digital improvements replace the classic analog box about the same level of responsivity was observed. However, it was believed that if the values of the filters were further refined our responsivity could be improved dramatically.

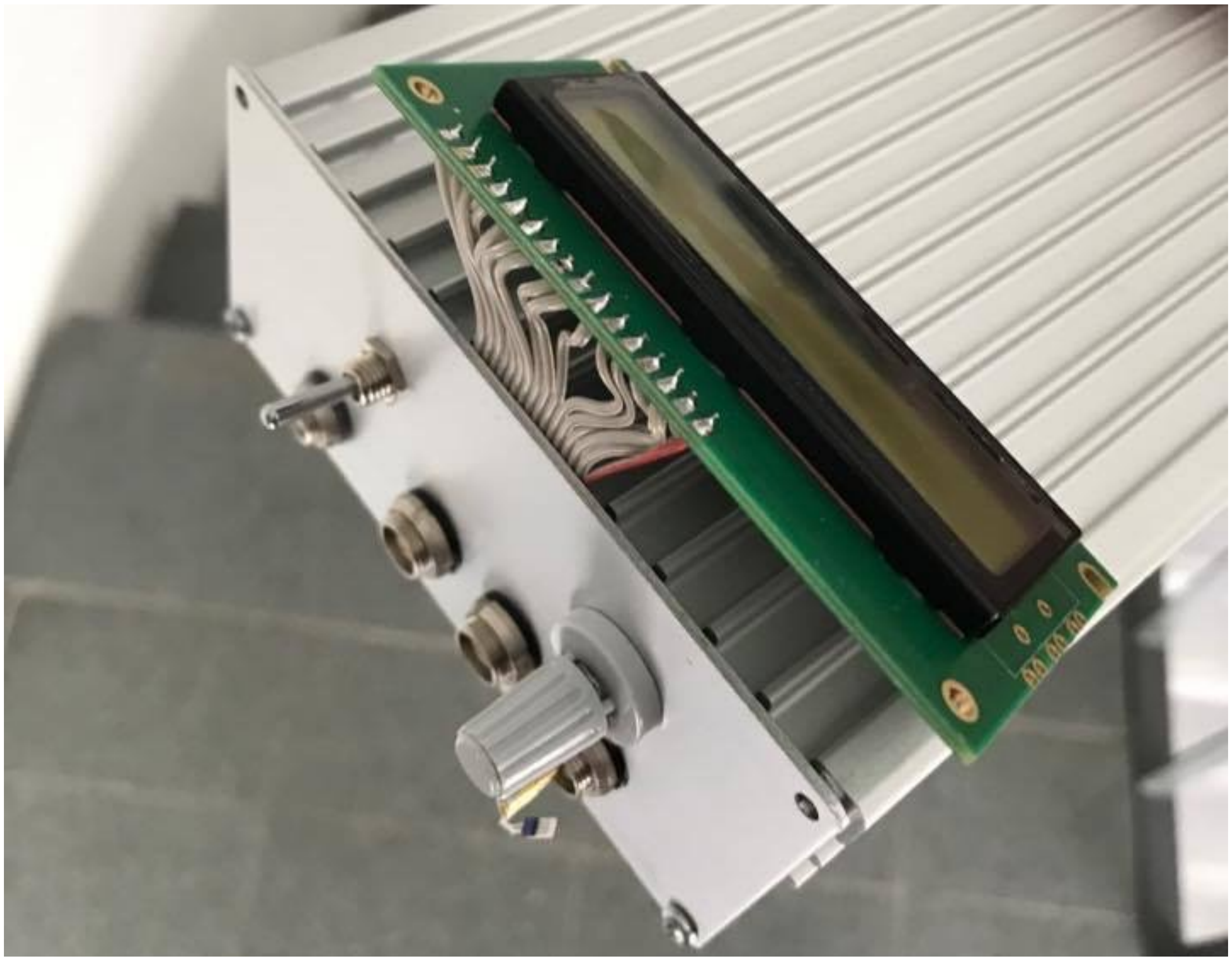


Figure 14 Finished AI box with an LCD, a potentiometer, and a switch for interfaces.

AI Theory:

When originally the digital filter was being programmed it reminded the author of another project he was working on, producing a 2D Neural Networks (NN) that can process simple mathematics using a similar weight system as used in digital filters, however NN's contain the ability to go back through and change their weights based on an idea called back propagation. The back propagation algorithm takes each weight and refines it based on calculated error, in this case due to the desire to filter, finding the error based on the exact values would be foolish so instead RMS values were used to determine error. We calculate the amount each node is off based on the error gradient multiply it by the learning rate and then subtract it from the current weight allowing for the weight to shift. As the whole system shifts down the error gradient we reach an ideal filter for the system in which it is being used. In our case our filter becomes ideal to block all small changes in the environment and produces an ideal digital filter. The backpropagation algorithm is expressed further in appendix I.

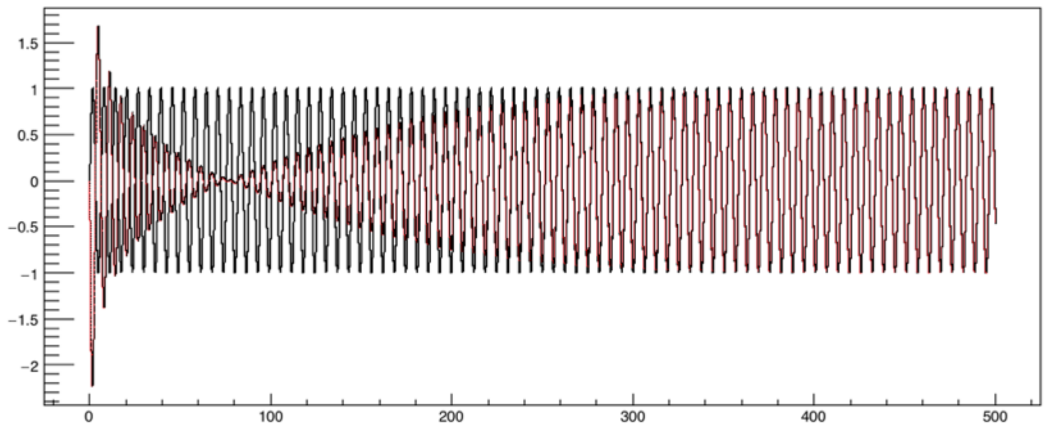


Figure 15 In this figure the Red is the filtered signal and the black is the input signal. The X-axis is in hundredths of a second.

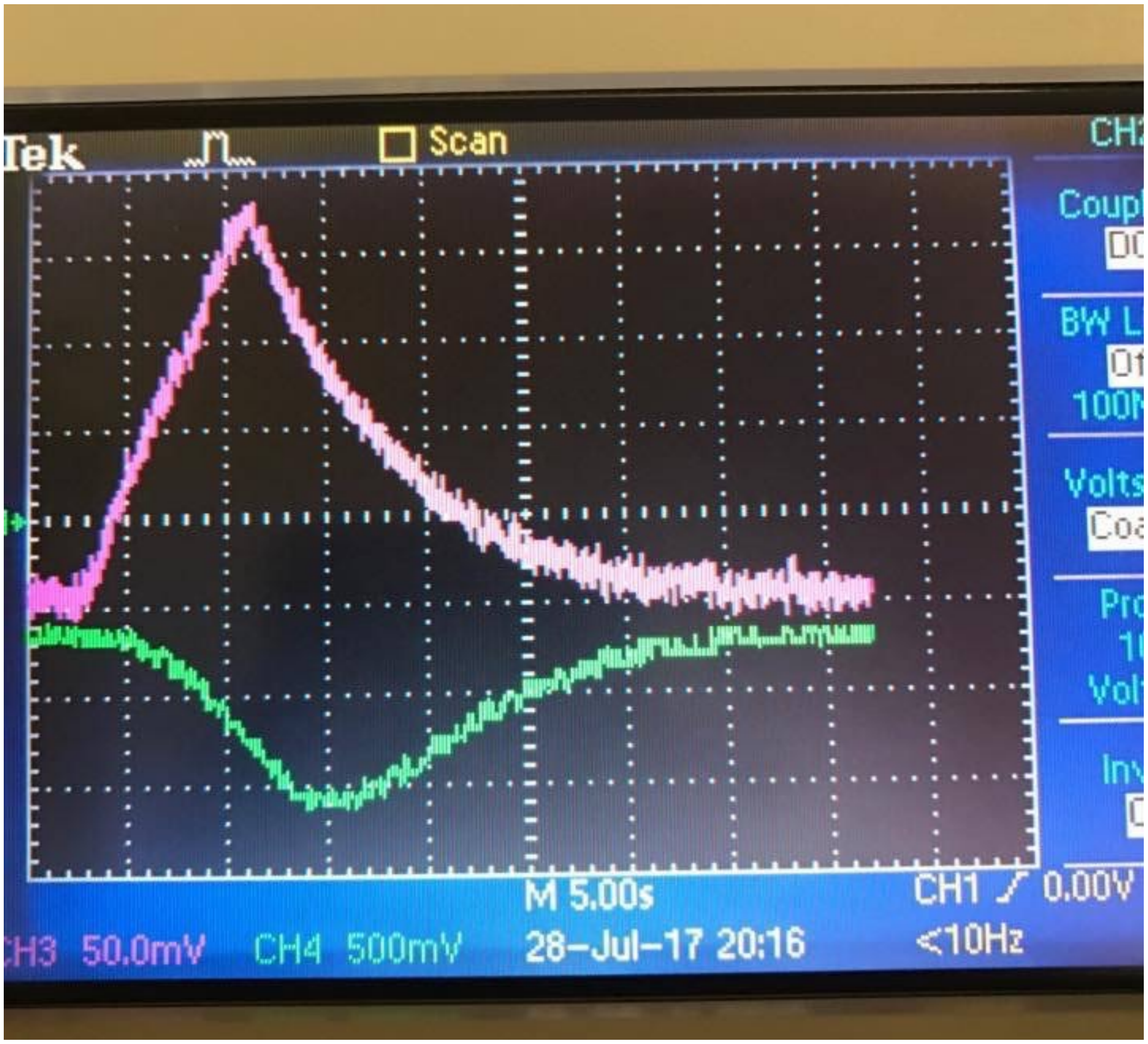


Figure 16 An example of the response we see from the digital circuit where pink is the driving current and the green is the temperature. We generated the gap in temperature by blowing hot air on the base.

The system's ability to react is highly dependent on the time allowed to train (a button toggles training) on the system. When the system trains for ~3 seconds we're able to match the analog system's response. If you let it train for ~5 minutes you can get a response of around one hertz, a dramatic improvement! Due to time constraints the results for long term training were not available until the author had already left the institute.

Conclusions:

Through the construction and categorization of the three boxes (Analog monitor box, digital box, and peltier driver) we have been able to offer two plausible control loops that would easily able to thermally control the system in questions with relatively reasonable response times for the systems we're using. The boxes have the added benefit of being relatively cheap and easy. These boxes in conjunction with a vacuum and soft balls allow for the entire system to be controlled and dampened, allowing for (hopefully) long term control.

Acknowledgments:

I would like to specially thank the Albert Einstein Institute, Fabian Thesis who guided me for most of the summer, Paul Opperman who guided me for the rest of the summer, and Benno Willke who allowed me into his labs in the first place.

References:

[1] K.D. Kokkotas, *Gravitational Wave Physics* (2002).

[2] C. Bradaschia, and R. Desalvo, "A global network listens for ripples in spacetime", *CERN Courier* (2007).

[3] O. Svelto, *Principles of Lasers* (Springer 1998).

[4] A. Abramovici, and J. Chapsky, *Feedback Control Systems* (Kluwer Academic Publishers, 2000).

[5] R. Paschotta, *RP Photonics Encyclopedia.*

[6] F. Zhang, and J.W.Y. Lit, "Direct-coupling single-mode fiber ring resonator", *Journal of the Optical Society of America,* **5**, 1347-1355 (1988).

[7] L.F. Stokes, M. Chodorow, and H.J. Shaw, "All-single-mode fiber resonator", *Optics Letters,* **7**, 288-290 (1982).

[8] F. Thesis. Personal Correspondence

[9] P. Opperman. Personal Correspondence

[10] B. Willke. Personal Correspondence

Appendix I

```
class SmartFilter{
  public:
    SmartFilter(int o);
    double useFilter(double raw);
    double dumFilter(double raw);
    double setV(int place, double v1, double v2);
    double getB(int place);
    double getF(int place);
    double checkError();
    double getSum();
    void randomize();
  private:
     double errTol = 0.0001;
     bool trained = false;
    int currIndex;
    int order;
    int window;
    double * ffCoeff;
    double * fbCoeff;
    double * in;
    double * out;
    double learningRate;
    double weightCoeff;
    int Shift(int shift);
    void DoGradientDescent();
    void GetLossGradient(double * ffGrad, double *
fbGrad);
    void GetVarGradient(double * ffGrad, double *
fbGrad);
};

SmartFilter::SmartFilter(int o){
  order = o;
  window = 2*(order+1);

  learningRate=0.001;
  weightCoeff =0.33;

  currIndex=0;

  fbCoeff = new double[order +1];
  ffCoeff = new double[order +1];
  in = new double[window]();
  out = new double[window]();

  for (int ab = 0; ab < window; ab++){
    in[ab]=0;out[ab]=0;
  }
  for (int ab = 0; ab < order+1; ab++){
    ffCoeff[ab]=0;fbCoeff[ab]=0;
  }
}

int SmartFilter::Shift(int shift){
  int sz = window;
  int temp = currIndex+shift;
  if (0<=temp&& temp<sz)return temp;
  else if(temp>=sz)return temp-sz;
  else return temp+sz;
}

double SmartFilter::setV(int place, double v1, double
v2){
   fbCoeff[place]=v1;
   ffCoeff[place]=v2;
}

double SmartFilter::checkError(){
  double tot=0;
  trained = true;
  for (int i = 0; i < (order+1)&&trained; i++){
    tot+=abs(in[Shift(-i)]-out[Shift(-i)])/abs(in[Shift(-
i)]);
    if ((abs(in[Shift(-i)]-out[Shift(-i)])/abs(in[Shift(-
i)]))>errTol){
      trained = false;
    }
  }
  return (tot);
}

double SmartFilter::useFilter(double inV){
  currIndex = Shift(1);
  in[currIndex]=inV;
  out[currIndex]=0;
  for (int i = 0; i < order+1; i++){
    out[currIndex]+=in[Shift(-i)]*ffCoeff[i] -
out[Shift(-i)]*fbCoeff[i];
  }
  if(!trained){
    DoGradientDescent( );
  }
  //checkError();
  return out[currIndex];
}

double SmartFilter::dumFilter(double inV){
  currIndex = Shift(1);
  in[currIndex]=inV;
  out[currIndex]=0;
  for (int i = 0; i < order+1; i++){
    out[currIndex]+=in[Shift(-i)]*ffCoeff[i] -
out[Shift(-i)]*fbCoeff[i];
  }
  return out[currIndex];
}

void SmartFilter::DoGradientDescent(){
  double * ffDiffGrad = new double[order+1]();
  double * fbDiffGrad = new double[order+1]();
  GetLossGradient( ffDiffGrad, fbDiffGrad );

  // Get gradient from signal loss error
  double * ffVarGrad = new double[order+1]();
  double * fbVarGrad = new double[order+1]();
  GetVarGradient( ffVarGrad, fbVarGrad );

  for ( int i = 0; i < order + 1; ++i ){
   ffCoeff[i] -= learningRate *
( weightCoeff*ffDiffGrad[i] + (1.0 -
weightCoeff)*ffVarGrad[i] );
   if ( i != 0 )
   {
      fbCoeff[i] -= learningRate *
( weightCoeff*fbDiffGrad[i] + (1.0 -
weightCoeff)*fbVarGrad[i] );
   }
  }

  delete[] ffDiffGrad;
  delete[] fbDiffGrad;
  delete[] ffVarGrad;
  delete[] fbVarGrad;
}

void SmartFilter::GetLossGradient( double * ffGrad,
double * fbGrad ){
    for ( size_t i = 0; i < order + 1; ++i ){
      ffGrad[i] = 0;
      fbGrad[i] = 0;
    }

    for ( size_t i = 0; i < order + 1; ++i ){
     double temp = ( 2.0 / double( order+1 ) ) *
( out[Shift( -i )] - in[Shift( -i )] );


    for ( size_t j = 0; j < order + 1; ++j ){
     ffGrad[j] += temp*out[Shift( -(i+j) )];
     if ( j != 0 ){
       fbGrad[j] -= temp*out[Shift( -(i+j) )];
     }
    }
   }
 }


 // Gradient from error due to variance
 void SmartFilter::GetVarGradient( double * ffGrad,
double * fbGrad )
 {
    double sumFilt = 0.0; // const across all weights
    for ( size_t i = 0; i < order + 1; ++i ){
     ffGrad[i] = 0;
     fbGrad[i] = 0;
     sumFilt += out[Shift( -i )];
    }
    for ( size_t i = 0; i < order + 1; ++i ){
      double ffTemp = 0.0;
      double fbTemp = 0.0;
      for( size_t j = 0; j < order + 1; ++j ){
        ffGrad[i] += out[Shift(-j)]*in[Shift( -(i+j) )];
        ffTemp += in[Shift( -(i+j) )];
        if ( i != 0 ){
          fbGrad[i] -= out[Shift(-j)]*out[Shift( -(i+j) )];
          fbTemp -= out[Shift( -(i+j) )];
        }
      }
     ffGrad[i] = ( 2.0 / double( order+1 ) ) * ( ffGrad[i]
- ( 1.0 / double( order+1 ) )*sumFilt*ffTemp );
     fbGrad[i] = ( 2.0 / double( order+1 ) ) *
( fbGrad[i] - ( 1.0 / double( order
+1 ) )*sumFilt*fbTemp );
    }
 }

 double SmartFilter::getB(int place){
  return fbCoeff[place];
 }
 double SmartFilter::getF(int place){
  return ffCoeff[place];
 }
```

```
void SmartFilter::randomize(){
  currIndex = 0;
  for (int ab = 0; ab < window; ab++){
    in[ab]=0;out[ab]=0;
  }
  for (int i = 0; i < order+1; i++){
    setV(i,((double)random(-100,100))/100.0,
((double)random(-100,100))/100.0);
    //setV(i,0,0);
  }
}

double SmartFilter::getSum(){
  double total = 0;
  for (int i = 0; i < order+1; i++){
    total = total+(getB(i));
    total = total+(getF(i));
  }
  return total;
}
```