The goal of this course is for you to be able to understand how to design, build, and interface sensors with computers or microcontrollers. The sensor serves as the eyes, ears, etc. of an operator and the computer or microcontroller represents the brain. It will collect data from the sensors and then make decisions about the proper response to that data. That response will come in the form of an electronic signal that may be used for record keeping or some kind of process control or both.

The simplest example might be an old style thermostat. One way to keep a room at a comfortable temperature would be to have an on/off switch for the heater or air conditioner. It is left off until the human in the room is uncomfortable, and then the human turns it on. Once the temperature has moved to the correct region, the human would turn it back off. This is clearly a process that needs to be automated. In an older thermostat, the temperature might be measured through the use of a bimetallic strip. Since each metal has its own coefficient of thermal expansion, two strips of different metals bonded together will be strained by temperature changes. The metal with the higher value of $\alpha$ (expansion coefficient) will get longer than the other metal at higher temperatures and will get shorter at lower temperatures. Since the two are bonded together, this will cause the strip to curve one way at high temperatures and the opposite way at low temperatures.

To make a thermostat, all you need to do is pin down one end of this strip and look at the varying positions. As shown below, a heating circuit could be activated by breaking the circuit and connecting the free ends to the black and blue contacts in the left image. A cooling circuit could be activated by using the same procedure for the black and red contacts in the right image (assuming the bronze material has a larger value of $\alpha$ than the grey material).

While a modern-day thermostat would probably be composed of solid-state sensors connected to relays, another important consideration would be the amount of tolerance in the system. Going back to our human operator, assuming this person was extremely sensitive, he/she would quickly tire of turning the AC on when the temperature hit 25°C and then turning it back off as soon as the temperature fell to 24.999°C. Additionally, that kind of rapid on-off cycling is not healthy for any part of the cooling system. For that reason, there is generally a difference between the turn-on temperature and the turn-off temperature. A large gap (10°C) is obviously undesirable, but a small gap is problematic as well.
**Sensors and Signals**

We can broadly define sensors into two types based on their output format: analog and digital. An analog output is a signal that is (theoretically) infinitely fine. If a needle is moving across the face of a dial (such as a car’s speedometer), it might be confined by two pegs to move through an angle of, say, 300°. For an analog instrument, if the needle were long enough and the dial’s diameter large enough, we could read its precision to an arbitrary number of significant figures. Close inspection might tell you that the car’s speed is 62.146238623 km/hr. Whether that degree of precision is at all justified is another matter.

For a digital speedometer, we would probably find that there are only three digits and we are therefore unable to see a distinction between 62 km/hr and 62.146… km/hr. This seems like a significant disadvantage at first, but there are a couple of benefits to this digital display. First, it is already in a form that is easy for a computer or other digital device to manipulate. Second, while we can imagine a giant speedometer so large that we can read the ninth decimal digit, in reality we will find that the finite size of a car’s dashboard means we might actually not be very certain about whether the needle is indicating 62 km/hr or 61 km/hr. The digital display removes guesswork about the digits it shows, but that also means there can be no estimating about values between the digits.

Probably the most common kind of sensor output we’ll see is a voltage. While many others are possible, this will be the most important for us. A wide variety of sensors, including those for measuring magnetic fields and accelerations, provide an analog voltage with an amplitude that represents the size of the field or acceleration. Because microcontrollers and computers don’t natively understand analog quantities, this usually means we have to perform an **analog to digital conversion (ADC)**.

**Analog to Digital Conversion**

For an analog signal that can vary as a function of time, that means that we have two choices to make. First, how often do we perform this conversion? For an example, look at the graph below which we can pretend represents a chart recorder reading a voltage as time moves on.
If it takes a significant time period to make our A-D conversion, our digitized output might look like this:

![Graph showing a digital signal with a low sampling rate.](image)

This might be unusable for our purposes. The **sampling rate** is very low here. Below, you can see a representation of the signal with a sampling rate 10 times greater than the one above:

![Graph showing a digital signal with a higher sampling rate.](image)

If we are trying to record sound, for example, the analog output represents the position of a microphone (or speaker) diaphragm as a function of time. It will be moving back and forth very quickly on the scale of things we can observe with our eyes. If we want to accurately represent sound, it’s important to know that we can hear things over a range from about 20 Hz to about 20,000 Hz (this upper end disappears quickly as you age – you might very well be unable to hear above 10,000 Hz unless a sound is very loud).

This does **not** mean that converting our analog waveform by looking at its value every 20,000th of a second and digitizing that number will give us good audio. The Nyquist-Shannon sampling theorem says that to reproduce a wave of frequency $f$, we need to sample (convert) at least twice that often. For this reason, CDs perform conversions 44,100 times per second. The purpose of this is to avoid **aliasing** of high frequencies.

Aliasing happens when a frequency higher than the sampling frequency $f_s$ is incorrectly identified as being a different frequency (lower than $f_s$), as shown below. Imagine that the green vertical lines represent the instants in time when samples are taken. The red line is...
the signal we’re trying to sample, but notice that everywhere the red line intersects a green line, it also intersects the blue line, which is a lower frequency signal. There are an infinite number of sine waves that have the same value as the red line at each sampling interval (green line). We will always assume that the lowest frequency wave that fits the data is the signal actually being detected because it’s the most reasonable assumption. It is up to the people designing/using the equipment to know its limitations and to not exceed them.

The second decision we need to make is the depth of sampling, or the resolution. This is expressed in the number of bits used for each sample. For example, if we did an 8-bit ADC of a signal that could have values from 0-5 volts, this would tell us that the smallest difference in two analog voltages that our digital result could display would be 5 volts / 2^8 or 5 V / 256 = 19.5 mV. Higher resolution means we have a more faithful representation of the analog signal. If we go to 16-bit ADC, that would mean we can now see voltage differences as small as 5 V/ 2^16, or 76 μV. As you might expect, there are some downsides to a higher-resolution conversion. You can expect a higher-resolution ADC to be slower or, for the same speed, a more expensive chip. Obviously, 16-bit conversion will take twice as much memory to hold as 8-bit conversion. For a music CD, 16-bit ADC is used. You can find devices that will record audio using 24-bit resolution, but at a certain point you will probably find that 1) you can’t tell the difference between high resolution and super-high resolution and 2) the rest of your equipment may introduce noise that is larger than the signal difference between two adjacent digital values.

**Digital to Analog Conversion**

As you might expect, if you need to convert from analog to digital so that a computer can understand a signal, you may need to convert from digital to analog so that the computer’s response can be noticed properly in the physical world. In this case, we need to take a number and convert it to a voltage. Since we (theoretically) lost resolution when going from analog to digital, we will not be able to re-create it when we go back to analog from digital.
Our output signal will not have infinite variability, but will rather be one of 256 values if we are doing an 8-bit digital to analog conversion.

One simple way to do this conversion is to use an operational amplifier (op amp) in the summing amplifier configuration.

The voltage output $V_O$ of the circuit above will be

$$V_O = \left( \frac{-V_a}{R_a} + \frac{-V_b}{R_b} + \frac{-V_c}{R_c} \right) R_f$$

Of course, we’ll want an inverter in there to fix the negative sign, but our three voltages $V_a$, $V_b$, and $V_c$ are all going to be either on or off. For most digital circuits, our “on” will be 5 V. This would be a 3-bit DAC. We would need to size the resistors appropriately. For example, if $R_a = 100 \, \Omega$, $R_b = 200 \, \Omega$, $R_c = 400 \, \Omega$, and $R_f = 50 \, \Omega$, a binary value of 110 (which is equal to 6 in decimal) would give us a voltage of (again, assuming we get rid of the negative)

$$\left( \frac{5 \, V}{100 \, \Omega} + \frac{5 \, V}{200 \, \Omega} + \frac{0 \, V}{400 \, \Omega} \right) 50 \, \Omega = 3.75 \, V$$

This 3-bit ADC will have a resolution of $5 \, V / 2^3 = 0.625 \, V$. This is not the best possible way to make a DAC. As you can see, if we were to move this up to an 8-bit conversion, our last resistor (carrying the least significant bit) would have a resistance of 12,800 $\Omega$. That’s not a particular problem in itself, but this design of DAC requires us to order a large number of different components.
As to the required precision, assume you are trying to convert the binary number 10000001 to an analog voltage. That would be

\[
\left( \frac{5V}{100 \Omega} + \text{zeros} + \frac{5V}{12800 \Omega} \right) 50 \Omega = 2.5195 V
\]

What if your 100 \( \Omega \) resistor (standard ones might have a tolerance of 10\%, but 1\% resistors are still very cheap) were really 99 \( \Omega \)? Then converting the binary number 10000000 to an analog voltage will give you

\[
\left( \frac{5V}{99 \Omega} + \text{zeros} \right) 50 \Omega = 2.5252 V
\]

This is not unexpected, really; if you want to get 8-bit conversion, that means a resolution of 1 part in 256, which is significantly better than 1\%. Your resistors will have to be of at least that tolerance or better for this scheme to work.

Another DAC design that is useful is known as the R-2R ladder. In this configuration (from http://www.allaboutcircuits.com/vol_4/chpt_13/3.html), we have resistors arranged as below

\[R/2R \ "ladder" \ DAC\]
For this setup, we only need two kinds of resistors (or even one, if we want to make the 2R by combining two of the R resistors in series). We still need each resistor’s precision to be compatible with the number of bits in the ADC. We can’t make a 16-bit ADC with 1% resistors! Modern integrated circuit ADCs that use this technique use resistors inside the chip that have been made by the usual photolithography process and laser-trimmed for high accuracy.

**Signal Conditioning**

One thing many different types of sensors have in common is that their output is an analog voltage. Frequently, that voltage will have to be manipulated in some way before it can be plugged into the chain of information (from the sensor) \( \rightarrow \) decision (from a computer, microcontroller, human, etc.) \( \rightarrow \) action (from a motor, heater, alarm, etc.).

As an example, one useful sensor is known as a linear ratiometric Hall effect sensor. The ratiometric part means that the output voltage is a function of both the magnetic field (which is what you want to measure) and the input voltage powering it. For example, if the chip is powered by a five volt supply, it will output half that (2.5 V) if there is no field. The presence of a S pole approaching the face will drive the output higher and a north pole will drive it lower. If we imagine a strong magnet’s south pole near the chip, we can drive its output to nearly +5 V. The problem is, that output will be “weak”, or high impedance. We can’t get much current out of it, so we can’t do things like drive LEDs, motors, etc.

The way to get around this is to amplify the signal. Keep in mind that this doesn’t always have to mean increasing the voltage. We can use a voltage follower, for example,

![Voltage Follower Diagram](image)

This is about the simplest possible op-amp configuration. Here, \( V_{in} = V_{out} \), but the op-amp has high input impedance and low output impedance, so it can drive much larger loads than the sensor itself. This is also called a Unity-Gain Buffer.

Of course, even with this same example, what if we have a strong north pole near the Hall sensor – we will get a progressively smaller voltage. Perhaps we know that this sensor’s environment will always be in a reasonably large north field, so that its output will be from zero to one volt above ground. The voltage follower might not be enough here, since one volt (or less) will not be enough to drive an LED, even if we can get a large amount of current out of it. In this case, we might want a **non-inverting amplifier** (see below).
Here, the signal we want to amplify ($V_{in}$) will be boosted by a factor of \((1 + \frac{R_f}{R_s})\).

We can now make the 0-1 volt signal as large as we need to drive an LED, a motor, or whatever else we may want.

In other situations, we may want to reduce the amplitude of a signal. The fastest, cheapest, and simplest way to do this is usually through the use of a voltage divider, as shown below.

This is the simplest 2212 resistor problem. We know that the total resistance will be 1,100 $\Omega$ and the potential across that resistance is 9 V, meaning a current of 8.18 mA will flow in the circuit. That means a voltage drop of $1000\Omega \times 8.18$ mA or 8.18 V across the larger resistor, and $100\Omega \times 8.18$ mA or 0.818 V across the smaller one. The voltage across the smaller resistor, then, is equal to
\[ V_{\text{out}} = \frac{R_{\text{small}}}{(R_{\text{small}} + R_{\text{large}})} V_{\text{in}} \]

In a measurement situation, if we are trying to do something like measure the voltage across the terminals of a car battery with something like a microcontroller with an analog input limit of 5 V, we need to set up a voltage divider to make it work. Our max voltage in will be (something like) 14 V, and our max voltage out will be 5 V. We need a pair of resistors such that \( \frac{R_{\text{small}}}{(R_{\text{small}} + R_{\text{large}})} = \frac{5}{14} = 0.357 \). Note that this equation has an infinite number of solutions. Reality is a little different, though. First, resistors are not commonly available in an infinite variety of values. For example, if \( R_{\text{small}} \) is chosen to be 1000 \( \Omega \), that means we need \( R_{\text{large}} \) to be 1801 \( \Omega \). We can get 1.8 k\( \Omega \), which is certainly close enough, but as a general rule we may not be able to get that close easily. We might have to do something like go for one that is (for example) 1900\( \Omega \) instead. We would not want to go in the other direction, because a 1000 \( \Omega \) resistor in series with a 1700 \( \Omega \) resistor will give us a \( V_{\text{out}} \) of 5.19 V, which is too large.

What if we decided to use a 1 \( \Omega \) and a 1.8 \( \Omega \) resistor instead? This would give the same size of a signal, but what we would now notice is that our battery has a load of only 2.8 \( \Omega \) across it. For a car battery, this would mean dissipating something like 14^2/2.8 or about 70 W in these two resistors, which means they have to be huge to avoid a fire. In a more realistic case, we might find that using values that small loads the circuit so much that the voltage source (which we are trying to measure) starts to sag noticeably. For small enough values (1 m\( \Omega \) and 1.8 m\( \Omega \)), we could make even the car battery’s voltage start to drop significantly (assuming we don’t start a fire first). For that reason, we would generally be better off making a voltage divider from large resistors than from small ones. A 10 k\( \Omega \) and 18 k\( \Omega \) resistor together would not be a particularly large load even across a 9 V battery.

This is very important to keep in mind if your sensor has a high output impedance and would be unable to drive a large current.

Filters

One of the other useful ways to manipulate signals is through the use of a filter. A filter is used to block a range of frequencies and allow (or pass) a different range of them. For example, one use of a low-pass filter (which allows only frequencies below some value and attenuates those above it) is to turn a square wave into a sine wave of the same frequency. The idea in this case is that every square wave can be decomposed into a Fourier series of a fundamental sine wave and its odd harmonics. If the higher-order harmonics can be removed from the square wave, what is left is a sine wave.

It’s important to keep a few things in mind – first, a real filter is not ideal in the way you might like. It would be nice to build the low-pass filter mentioned previously (assume we’re trying to take a 1 kHz sound wave and make it a 1 kHz sine wave) and have it completely stop all frequencies from 1001 Hz on up. In reality, what it will do is significantly attenuate
(but not completely remove) those higher frequencies. As the frequency gets higher, the low-pass filter will pass less and less of it.

A simple low-pass filter circuit can be made by placing the load (where you’ll be measuring output) in series with an inductor across the signal source. Because the reactance of the inductor increases linearly with frequency, higher frequencies will see larger and larger resistances in series with the load, making a voltage divider. As the frequency and reactance increase, only a smaller and smaller voltage will appear across the resistor.

Because of the duality between inductors in series and capacitors in parallel, we could also make a low-pass filter using the arrangement below:

In this case, as the frequencies get higher, the capacitor becomes a lower and lower impedance path to ground. In the limit as frequency goes to infinity, the capacitor completely short-circuits the load and all of the voltage is dropped across resistor $R_1$ (which we do not care about).

Although there is a frequency associated with a filter called the cutoff frequency, again remember that the cutoff is not infinitely sharp. For the capacitive filter above, the cutoff frequency is found from

$$f = \frac{1}{2 \pi R C}$$
but what that really means is that the output voltage is down to 0.707 multiplied by the input voltage (note that $0.707 = 1/\sqrt{2}$). The reduction is greater at higher frequencies.

As you might have guessed, a high-pass filter will look similar to the low-pass filters, but capacitors and inductors will change places.

In the case above, the capacitor will appear as a high impedance device to a low frequency, but it will barely register as being there as the frequency climbs higher and higher. Again, this is essentially a voltage divider with the voltage across the load falling for low frequencies and climbing closer to the input voltage at higher frequencies.

The inductive version works oppositely; lower frequencies see the inductor's impedance almost disappear and will take that route to ground, while higher frequencies see the load as the smaller impedance and will take that path.

One application for these types of filters is in the direction of audio signals to the particular speaker types best suited for reproducing them – high frequencies to tweeters, low frequencies to woofers, etc. While we could make these filters with either inductors or capacitors, in practice capacitors are usually used. Inductors, since they are just loops of wire, are susceptible to picking up noise (essentially, acting as pickup coils) more easily than capacitors, and are also typically more expensive to construct.
Other possible types of filters are band-pass filters, which allow frequencies within a certain range and exclude others, and notch (or band reject) filters, which reject those in some band. Examples are shown below:

|--------|-------------------------|--------------------------|---------------------------------------------------------------------|

Notice that the left side passes low frequencies on to the right side, which passes only high frequencies on to the load.

More involved filters are also possible. A second order filter will have two components (inductors or capacitors – resistors don’t count) and should provide a steeper dropoff to the filter output. What we have seen so far are called **passive** filters, since they only incorporate passive elements like resistors, capacitors, and inductors. If we include things like op-amps, we are talking about **active** filters.

An $n$th-order filter will have a sharper cutoff than a first-order filter, making it closer to ideal. One way to make one is shown below
Notice that it is just two cascaded low-pass filters. We could also make this with a couple of R-L low-pass filters or a couple of capacitive low-pass filters.

Active filters (those involving an op-amp) are able to provide a $V_{out}$ which is larger than $V_{in}$ for those frequencies it passes. This isn’t very surprising since the filter’s usual job is to block some frequencies and pass others, while the op amp is generally used to provide a $V_{out}$ greater than $V_{in}$. The ratio of these two things ($V_{out}/V_{in}$) is known as the **gain**.

If we connect a voltage follower to a first-order passive filter, we’ll get a first order active filter (note that we aren’t *required* to have gain just because the filter is active). The “rolloff” from a first order filter is 20 dB/decade, and each additional order contributes another 20 dB/decade, so that a second-order filter has a rolloff of 40 dB/decade. The figure 20 dB/decade means that, as the frequency goes up by a power of 10 (assuming we are talking about a low-pass filter), the **power** (which goes as the square of the voltage) will drop by 20 dB (which is two powers of 10) compared to where it was before.

---

You can think of low-pass filters as being something like a moving average of data. For example, if you are monitoring the voltage output of a 9 V battery under a light load as a function of time, you might get a series of measurements like 8.80, 8.77, 8.81, 8.83, 8.72, 8.85, 8.89, 8.83, etc. A plot of these numbers would look very flat, assuming you include zero on your y axis. If there was some kind of glitch or a sudden, large load on the battery, maybe one of your numbers in the middle would be 6.55. This will show up as a huge change, but it may only be a measurement error if the next measurements are like the previous set.

Ignoring it completely might be a bad idea, but including it (when it could be the only outlier in 100 or 1000 measurements) distorts the graph for no particularly good reason. You could take a **moving average**, where your first “adjusted” data point represents the average of the first 5 actual data points, and then your second adjusted point is the average of actual points 2-6, and the third adjusted point is the average of actual points 3-7, etc.

This way, the low data point is still noticeable, but its influence on the overall picture is reduced. If there are several data points down in the 6.5 range, the adjusted data point graph will eventually get down that low, but a single point is not enough to wreck everything. Sudden, dramatic changes such as this are high-frequency signals. Your $\Delta V/\Delta t$ is very large in this case, since the smallest $\Delta t$ would be equal to the period between...
samples and the $\Delta V$ would be large (2 or more volts, here). The moving average **attenuates**, or reduces, that high-frequency signal.

In an imaging situation, a low-frequency signal might represent the very gradual change in color as you move from one side of a picture of a clear blue sky to the other side. If an image has had a low-pass filter applied to it, sudden dramatic changes get removed. For example, a tiny plane flying high in that sky might be blended into the background until it is very hard to locate or disappears completely.

A high-pass filter, on the other hand, works as kind of an edge detector. When things change dramatically over a small piece of the picture, that change is noticed and effectively amplified by the filtering (removal) of less rapidly changing information around it.

This kind of filtering is very significant in medical imaging. For example, in nuclear medicine scans (PET, SPECT, etc.), the amount of radiation delivered to the patient is a primary concern. Better pictures would certainly be obtained if very high doses of radioactive imaging agents were administered, but the risk of induced cancers would also rise. As a consequence, some of these images are photon-limited; in other words, they look blurry since photons are being detected one at a time, and there aren’t that many of them.

As you can see in the visible-light images above, the one with fewer photons on the left is almost unusable, while the one with about 2000 times as many photons on the right could be used to identify someone. Most SPECT images look much more like the one on the left. The natural way to “fix” this image is to use low-pass filtering. As you can see if you look at what turns out to be the face, although there are spots of darkness, most of the face area is light in the left image. A low-pass filter will tend to smooth this into a more uniform area that is not as bright as the brightest points shown, but is much brighter than background. You’ll get kind of a light gray circle for the face.

The big problem with doing this to medical images is that a white spot in a darker background might be a random pixel with a high photon count (though not statistically unexpected) or it might be the beginning of a tumor. As the color of the image changes dramatically from one pixel to the next like this, that is the definition of a high-frequency signal. Should it be filtered as noise, or highlighted as signal? This is one of the most important considerations in an imaging environment.
In general, cascading $n$ first-order passive filters is not a great way to get an $n^{th}$ order filter, since the filters will affect each other as is the case whenever a new element is added to an existing circuit. To reduce or eliminate this, we can use op-amps. Their very high input impedance and very low output impedance keeps the different stages from interacting.

The first figure on page 11 showed the passive first-order high-pass filter made with an RC network. The active version is shown below.

![High-pass Filter Diagram](http://www.engineer.tamuk.edu/cleung/EEEN4252/1_Active%20Filter.pdf)

Notice that this is essentially just the passive high-pass filter from page 11, but its output is now connected to a voltage follower. The advantage of the isolating effects of the op-amp is that multiple filters can now be cascaded to form higher-order filters without interactions between them. This will make the cutoff sharper than it would ordinarily be.

The standard plot for filter response is to plot the log of the gain vs. the log of the frequency. Since the gain for a passive filter will have to be less than one, its log will always be negative. In general, this is plotted in decibels.
As you can see, the first-order filter will have dropped the signal by around 35 dB at the maximum frequency shown, while the fourth order filter has dropped it by 80 dB well before the maximum frequency is reached. The circuit whose gain is shown above will look like this:

![Circuit Diagram]

**Figure 16–3. Fourth-Order Passive RC Low-Pass with Decoupling Amplifiers**

**Proportional Integral Derivative Control**

When trying to control a nontrivial process (i.e., trying to control room temperature by adding or subtracting heat energy from the room), there are several ideas to consider. The simplest way we could control the temperature of a room is through turning a heater (or air conditioner) fully on/fully off when the temperature is outside/inside of its selected range. This is essentially how most thermostats work. A particular setpoint (SP) is chosen by the thermostat user, and the system then works to keep the process variable (PV), which in this case is the room’s temperature, near the setpoint.
With this kind of full on/full off control (also known as bang bang control), a graph of the temperature in your house as a function of time probably looks like a sawtooth wave. In the summer, the house will warm and its temperature will rise until the A/C turns on, at which point the temperature starts to drop. As you can see in the graph below, the variability is significant.

Between 11 PM to about 1 AM, the setpoint itself was changed so that the house was more comfortable for sleeping. From about 1 AM until 9 AM, we see the temperature making a slow climb up followed by a rapid drop back down. In this case, the temperature of the house is kept within a range of about 0.8 degrees.

This is fine for most purposes, but we could keep the temperature much closer to our “perfect” value chosen by the setpoint if we had some variability in the A/C output. This is more like the way the cooling system in your car works. You have a few choices of fan speed as well as the ability to mix hotter outside air with the cold A/C air so that you can find the perfect balance. The down side of the car’s system is that, in many cars, there is no automated control; you are manipulating the volume and hot/cold mix of the airflow itself, rather than the temperature of the car. If you have it set exactly like you want it on a cloudy day and you then drive into a region of bright sunlight, you will find that the car’s temperature goes up and it takes manual intervention from you to find the new balance.

If we want to automate this, the simplest way to do it is to introduce proportional control. We watch the difference between the PV and SP (current and desired temperatures here) and use that to determine the output of the air handler. Of course, since we are relating heat output (watts) to temperature (°C), there will have to be some constant connecting them. In this case, it’s known as the gain, or, more specifically, the proportional gain.
If we want to heat a room during the winter, we would arrange this so that, when the PV fell below the SP, we would first calculate the **error signal**, which can be written as either PV-SP or SP-PV, depending on the way our control system works (adding heat or removing it). That error $e$ is then multiplied by the gain and the product determines the heat output of our system. Obviously, a low gain will be a problem. If we’re trying to heat a poorly insulated warehouse and have a gain of 500 W/$^\circ$C, we will probably find that the temperature just keeps dropping. In this case, the gain is too low.

Let’s say the particular conditions of insulation and indoor and outdoor temperatures mean that the warehouse is losing heat at the rate of 2000 J/s or 2000 W. If we have set the SP for 20$^\circ$C, what we will find is that the temperature will keep dropping until the heater is finally putting out heat at the same rate the building is losing it – 2000 W. With our chosen gain, this will not happen until the temperature drops to 16$^\circ$C. At that point, the error will be 4$^\circ$C and the gain of 500 W/$^\circ$C will cause the furnace to output 2000 W.

This is not a great solution. The room is now significantly colder than we had wanted. One solution would be a larger gain – assume we bump it up to 1000 W/$^\circ$C. Now our temperature will only have to fall to 18$^\circ$C for the heat output to match the heat loss. Still, this isn’t the temperature we set. You might just decide to bump the gain up dramatically and call it 20000 W/$^\circ$C. The good part is that now we can keep the PV within 0.1$^\circ$C of the setpoint! The bad part is that we will find, if we increase the gain enough, that a slight drop in temperature causes the heating system to kick in like a blast furnace and send our temperature noticeably **above** the setpoint. In real life, we would wait for it to drop below our setpoint again before turning it on, but what if we had a process we wanted to control from both directions?

One example here would be steering a car down the road. You are in charge of the wheel, and you’re trying to go straight down the road, but if external factors appear (an icy curve), you will have to make adjustments. In this case, if your gain is too low (you don’t turn the wheel enough), you will go off the edge of the road. On the other hand, if your gain is too high, you’ll start weaving back and forth as you overcorrect and you’ll still eventually end up off the road.

We see that excessive gain results in **oscillations**, which we don’t want. Insufficient gain means a PV permanently below (or above, if approaching from the other direction) the setpoint. One other complication we will find is that there will almost always be a delay between the application of some disturbance (including your control signal, like the heater turning on) and its affect on the PV. The temperature does not instantly start climbing the microsecond that the heater kicks on. Although your response would be quicker, it’s general practice not to mount the thermostat in your house in the path of an air vent. You don’t want it to turn off as quickly as it turns on!

This delay makes controlling things noticeably harder. If you have ever seen one of the “drunk driving simulator” vehicles police occasionally demonstrate, the idea behind them is that there is a delay of a second or more between the driver’s application of control (gas, brakes, steering) and the car’s actual implementation of it. This tends to cause things like
oversteering followed by overcorrecting. Higher proportional gain just means more violent
turns back towards the center of the road. A longer time delay means more travel in the
wrong direction before things are corrected (or overcorrected!)

One way to fix this is through the use of Integral control. This adds a term to the simple
equation of Output = K_p e where K_p is the proportional gain and e is the error term. We
include a term of the form

$$K_i \int e \, dt$$

where K_i is known as the integral gain. The gain can also be written in the form 1/\tau_i where
\tau_i is the integral time constant. In this case, if the output under proportional control were to
settle at a lower value such as our 16°C example, the constant 4°C error would persist. The
integral of that error would grow linearly with time. When a positive gain is
multiplied by this integral, we see that the output at 16°C will increase from the 2000 W
proportional-only term. As the integral term adds to the heat output, bringing it above 2000
W, the room’s temperature will start to climb and the integrated error will not get as large
as fast as it had been. The contribution from the integral term will continue to increase the
heat output, but only until the temperature reaches the setpoint. Above that, the error will
be in the other direction and the integral term will inhibit the proportional controller from
delivering as much heat.

This addition is enough in many cases to give us the control we want. Between the two
influences, the PV should settle near the SP rather quickly. The other kind of control we
can consider is known as derivative control. It will be of the form

$$K_d \frac{de}{dt} \text{ or } \frac{1}{\tau_d} \frac{de}{dt}$$

The purpose of this term is to dampen wild swings caused by rapid changes in the error.
There can also be a constant bias term, b, in the output. The bias is the value of the
controller’s output when the PV equals the SP (so that there is zero error). That leaves our
final ideal PID equation for the output (m) as

$$m = K_p \ e + \frac{1}{\tau_i} \int e \, dt + \tau_d \ \frac{de}{dt} + \ b$$

Digital Communication

Part of the function of a sensor or microcontroller is to exchange data; the sensor needs to
send data to its supervising computer or microcontroller, and that device may need to send
the information on to a higher-level machine. This is where digital communication comes
in. The advantages over analog communication are multiple and significant. First, the
computers “speak” digitally as their first language. Second, digital communications are
much less susceptible to noise. As an example, assume we are sending an analog voltage
signal from a thermistor or other device down a long wire to a microcontroller, where the microcontroller’s ADC will turn it into a digital number we can use for decision making. The length of wire involved will necessarily have some resistance, and that means there will be a voltage drop across it. The voltage that is sent is not the voltage that is received. Electrical noise on the wire also contributes to this effect.

If, on the other hand, the ADC is positioned near the sensor and its result is sent to the microcontroller as a series of high and low values (ones and zeroes), it takes much larger values of noise and/or resistance to cause problems with a digital signal. Unless the voltage drop is so large that the 5 V (for example) pulse sent by the sensor drops down to around 3 V or less, it won't matter. The microcontroller will still record 4.5 V as a one. Only an extremely noisy environment could cause the 5 V signal to degrade to the point that it looks more like a zero (maybe 2 volts or less) than a one.

For these and other reasons, it is becoming increasingly common for sensor outputs to be digital rather than analog. One of the choices to be made when looking at digital communication is that of serial vs. parallel. A serial connection will send each bit of a byte down the same line, one at a time. The parallel connection sends all 8 (or possibly more, depending on the design) bits down multiple lines at the same time.

The advantage of a serial connection is the small number of wires (generally 2-4) required for communication. The parallel connection will require more wires (probably 10 or more), meaning a heavier cable and the use of more microcontroller or computer input/output pins. The parallel advantage is that, theoretically, it should be able to move data about 8 times as fast as the serial cable.

As the speeds and capabilities of electronic equipment have increased, the need for parallel connections has dropped off significantly. The most popular method of computer/peripheral communication today is the Universal Serial Bus, or USB. There are three main standards, denoted by 1.0, 2.0, and 3.0. The maximum transmit speed for a 1.0 device is 12 Megabits per second. USB 2.0 increased that to 480 Megabits per second, and USB 3.0 provides for 5 Gigabits per second.

Before the days of USB, printer ports were parallel connections and things like mice, tablets, modems, etc. were serial (a standard known as RS-232 and still very commonly emulated. A USB-to-serial emulating cable can be bought for about $5.

This is useful because many microcontrollers are already equipped to “speak” in the RS-232 standard. One complication that may be observed for individual microcontrollers (as opposed to microcontrollers mounted on a development board with supporting chips in place, such as the Arduino) is the fact that the RS-232 standard may be implemented with signal amplitudes of 5 V, 10 V, 12 V, or even 15 V. It is common to need a signal translation chip, such as the SIPEX SP3222E or 3232E, or similar chips made by other companies. This serves to bump up the microcontroller’s 5 V output to the 15 V a computer may be expecting, and similarly drops an incoming 15 V signal from the computer down to a survivable 3-5 V for the microcontroller.
In addition to these methods of digital communication, there are others that require fewer wires than the full RS-232 and can even achieve higher throughput. If you think carefully about what is required of this few-wire setup, it becomes clear that communication like this is not trivial. Remember that each end (microcontroller or sensor) has only two possible signal levels; it’s not like the sensor could send a “2” which would alert the computer that it is about to send a stream of ones and zeroes. Similarly, the computer can not send back a “3” saying that it is ready to receive that data.

A similar problem emerges when we think about how the data must be sent. If we want to send something like 10001110, how long do we have to leave the line “high” to represent a one? If the sender drops it back down to 0 V to send the first zero, we have to make sure that it doesn’t drop so fast that the receiver actually misses the one. On the other hand, if we leave it high too long, will the receiver think that we’re sending a string of ones? Once it’s been “long enough” and we start to send the first of the three zeroes, how does the receiver know that we intended to send three of them, as opposed to thinking that maybe the sending chip is “slow” and only meant to send one or two?

For many microcontrollers, we have the option of running them at different clock speeds, meaning making them faster or slower (the downside of making them faster is that we will need more power and generate more heat). How can we expect a computer or a sensor to talk to these microcontrollers that could be running at a wide range of speeds?

You can imagine that you would have these same problems trying to communicate to another person far away by raising your right hand for “one” and lowering it for “zero”. What you could do in that case is use both hands. First you position your right hand either up or down, and then raise your left hand to indicate you are sending data. After you see the other person raise his left hand, you drop yours. Then you move your right hand into position for the next data bit, and when it’s there, raise your left hand again. Continuing that way, your receiver would always know if you meant to send one zero or three by counting how many times your left hand went up while your right hand stayed down. Notice that, if you did this enough to become skillful, your left hand would be moving up and down at a regular rate, like a clock.

You can duplicate this idea using only two wires (not including ground and power). In some implementations, the two wires are referred to as DATA and CLOCK. Clock would be the left hand and data would be the right hand. The details of this implementation vary, but one version of this kind of communication is known as I²C, or Inter-Integrated Circuit. This is a half-duplex standard, meaning there is one talker and one listener on the line at a time (like old walkie-talkies). The **Serial Peripheral Interface** is another type of serial protocol, and this one allows for full-duplex communication (like a phone) since, in addition to the clock line, there is a line labeled MOSI (Master Out, Slave In) and one labeled MISO (Master In, Slave Out).

Both of these protocols allow multiple devices to communicate with each other over the same wiring. With a clear understanding of the protocol, you could actually implement one of this directly in software by sending the proper high and low signals to the pins dedicated
to this kind of communication. This is known as **bit-banging** the signal since you are doing things at the most primitive level of highs and lows. Fortunately, people have developed libraries for most microcontrollers (including the Arduino) so we can avoid reinventing the wheel over and over again.

Among devices that use this kind of digital output are barometric pressure sensors, capacitance sensors, accelerometers, humidity sensors, temperature sensors, EEPROM memory chips, digital potentiometers

---

**Bibliography**

*Measurement and Instrumentation Principles*, Alan S. Morris
*Lessons in Industrial Instrumentation*, Tony Kuphaldt
*Handbook of Modern Sensors*, Jacob Fraden
*Building Scientific Apparatus*, John H. Moore, Christopher C. Davis, Michael A. Coplan
*Op Amps for Everybody*, Ron Mancini, Texas Instruments Design Reference
*Analysis of the Sallen-Key Architecture*, Texas Instruments Application Report

---

http://www.asdlib.org/onlineArticles/elabware/Scheeline_ADC/ADC_DAC_ladder.html
http://www.controlguru.com/pages/table.html