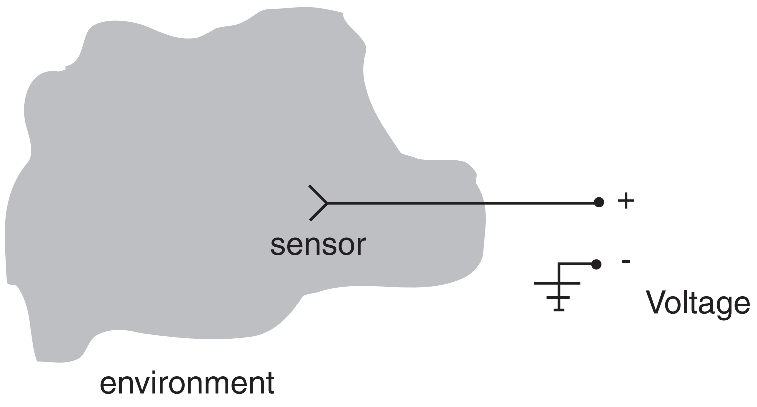
Objectives

Upon completion of this module, you should be able to:

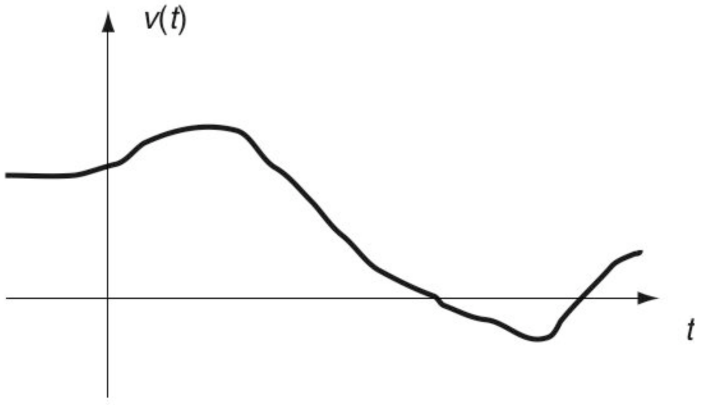
- understand uniform quantizers, including dynamic range and sources of error,
- represent numbers in two's complement binary form,
- assign binary symbols to quantized signal values, and
- scale a signal to fit within a specified range.

What are signals?



As we saw in the **"Volts and Amps"** module, sensors typically convert observed data into time-varying voltages. In other words the sensors act as *transducers* that convert environmental data into a corresponding current flow that can then be observed and measured with electronics as a changing voltage. The resulting data are functions of time with units in volts and are known as 'signals.'

What does such a voltage signal look like? Signals are commonly observed and graphed as functions of time:



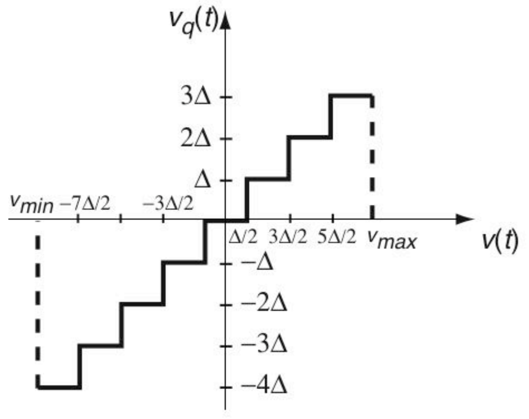
Such a function is known as a *continuous-time signal*. These signals are defined for all values of *t* and take on a continuous range of voltage values.

Problem: Digital computers cannot store this type of data.

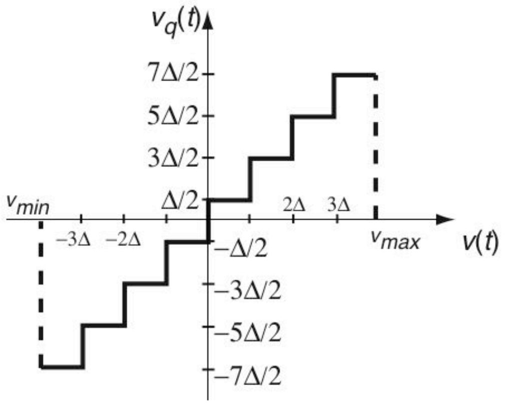
Numerical quantities represented by computers have finite precision, as they must be represented by a small number of bits. We have to *quantize* the values in $v(t)$ into the finite precision values that can be stored in however many bits are allowed per number.

Quantization

Assume that the observed voltage is known to range from a minimum of v_{min} to a maximum of v_{max}. A *quantizer* is a system that divides this continuous range of values into a finite number of discrete values. The most common quantizers are uniform quantizers that divide the range of values into equal-sized intervals. Here are two typical uniform quantizers for a system with 3 bits of precision.


$v_q(t)$
3Δ
2Δ
Δ
v_{min} $-7\Delta/2$ $-3\Delta/2$
$\Delta/2$ $3\Delta/2$ $5\Delta/2$ v_{max}
$v(t)$
$-\Delta$
-2Δ
-3Δ
-4Δ

midstep quantizer



midrise quantizer

For N bits of precision a quantizer has 2^N output levels. Thus, these 3-bit quantizers have 8 levels. Midstep quantizers are generally preferred over midrise quantizers as they are less sensitive to noise at low signal levels. Let Δ denote the size of the quantization levels in these uniform quantizers. Note that for any input value of $v(t)$ between $-\Delta/2$ and $\Delta/2$ the output of the midstep quantizer is the same, while the midrise quantizer will change output levels whenever the input crosses zero. For a small signal that is approximately zero but is varying slightly because of noise or similar observational variations, the midrise quantizer's toggling between $-\Delta/2$ and $\Delta/2$ can actually magnify the effect of these small distortions. The midstep quantizer is slightly asymmetric with more negative output levels than positive, but that difference becomes insignificant as the number of bits increases.

> Question: The LEGO Mindstorms NXT sensors use 10 bits to encode data. How many levels can be represented with uniform quantizers at that precision? Also, for most of the NXT sensors, $v_{max} - v_{min} = 4.3\text{V}$. In these cases what is Δ?

Now, for the example quantizers above we can represent each value of $v_q(t)$ with 3 bits in the NXT processor. The error in this approximation can be as large as $\Delta/2$, but that decreases as the number of bits increases.

> Question: Given v_{max}, v_{min}, and N bits of precision, what is Δ?

We can choose v_{max} and v_{min} to correspond to the range of voltages allowed by the NXT. That voltage range and the number of bits determine the dynamic range of the data. We'll define the dynamic range as the ratio of the largest value that can be represented to the smallest nonnegative value. Consequently, Dynamic Range = v_{max}/Δ.

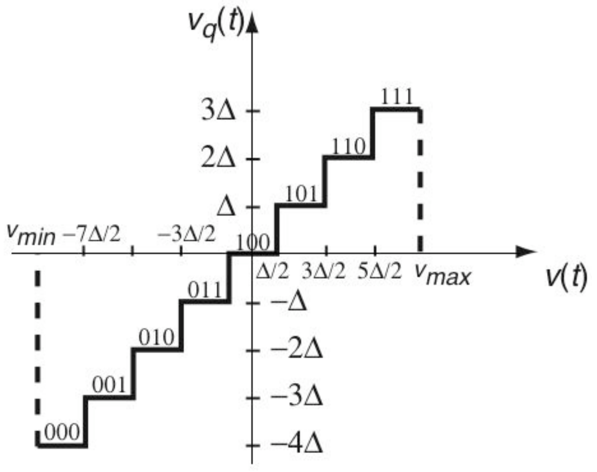
For the NXT and its sensors, we typically have that v_{min}= 0 with v_{max}= 4.3V. In this case $\Delta = (v_{max} - v_{min})/2^N = 2^{-N}v_{max}$. Thus, Dynamic Range = $v_{max}/(2^{-N}v_{max}) = 2^N$.

In many other cases, $v_{min} = -v_{max}$. Here, $\Delta = (v_{max} - v_{min})/2^N = 2v_{max}/2^N = 2^{-N+1}\mathrm{v}_{\mathrm{max}}$. Therefore, Dynamic Range = $v_{max}/2^{-N+1}v_{max} = 2^{N-1}$.

Binary Representations

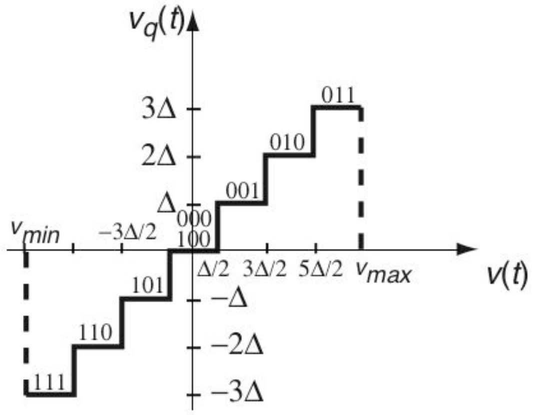
For 3 bits of precision we have 8 unique sets of 3 bits. How should we assign these *symbols* to the different quantizer output levels? Here, we will consider three of the most common binary representations.

Unsigned Binary: We could just label the levels in order from 000 (corresponding to 0_{10}) to 111 (corresponding to 7_{10}). (Note we are using the numerical subscripts to denote the *base* defining the number's representation. A subscript of 10 indicates that the number is in the standard decimal representation.)



However, this ordering does not provide a clear, mathematical relation between the symbols and the values of $v_q(t)$. For instance, we cannot add or subtract these binary values and have a correspondence to mathematical operations on the signal values.

Signed Magnitude Binary: An alternative is to use the first bit in each symbol as a *sign bit* to denote the sign of the voltage being represented. In this case 011 = 3_{10}, but 111 = -3_{10}. Thus, the levels would be labeled in the following manner.



However, with signed magnitude binary 000 = 100 = 0_{10}. Therefore, this binary notation can only support 7 quantization levels instead of the usual 8. Also, arithmetic on the binary representations of the levels, while more straightforward than for unsigned binary, still does not have a simple relationship to operations on the corresponding voltage values.

To add signed magnitude binary numbers:

- Add their magnitudes, if their signs are the same. Give the result the same sign bit.
- If their signs are different, subtract their magnitudes. The relative size of the magnitudes determines the final sign bit. Give the result the same sign bit as the number with the largest magnitude.

Example: 1_{10} + (-3_{10}) = 001 + 111
Subtract the magnitudes: 11 – 01 = 10
11 > 01, so the resulting sign bit is 1 (i.e., the result is negative)
Therefore, 011 + 111 = 110

These operations are fairly awkward, especially when they must be implemented in hardware.

Two's Complement Binary: This binary representation generally provides the best, most elegant solution. All 2^N quantization levels can be used, and arithmetic is straightforward. To find the two's complement binary representation:

- For positive numbers, two's complement is simply the standard, unsigned binary representation with the sign bit set to zero.
- For negative numbers and N bits, the two's complement representation is

$$2^N{}_{10} - |number_{10}|$$

Example: -2_{10} represented with 3 bits
$2^3{}_{10} - 2_{10} = 6_{10}$
Therefore, 110 = -2_{10} in two's complement with 3 bits

Fortunately, there's a much simpler algorithm for arriving at this answer:

two's complement representation = one's complement + 1,
where the one's complement representation for negative values is found by flipping all the bits from the unsigned binary representation of the absolute value of the number.

Example: -2_{10} represented with 3 bits
$2_{10} = 010 \rightarrow$ flip the bits $\rightarrow 101 \rightarrow$ add 1 $\rightarrow 110$

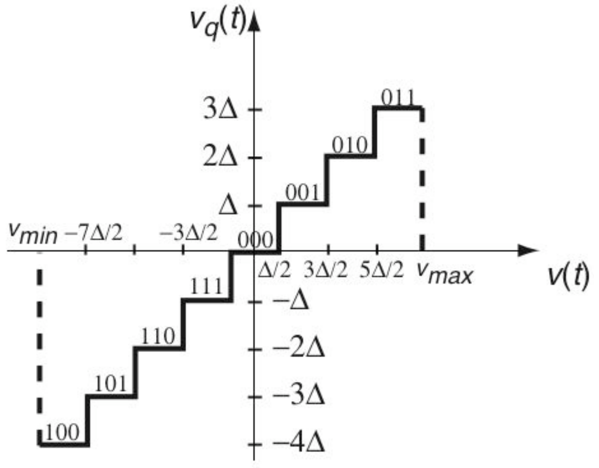
Harder example: -6_{10} with N=8 bits
$6_{10} = 0000\ 0110 \rightarrow$ flip $\rightarrow 1111\ 1001 \rightarrow$ add 1 $\rightarrow 1111\ 1010$

So, for 3 bits the two's complement binary numbers and their decimal equivalents are:

$011 = 3_{10}$
$010 = 2_{10}$
$001 = 1_{10}$
$000 = 0_{10}$
$111 = -1_{10}$
$110 = -2_{10}$
$101 = -3_{10}$
$100 = -4_{10}$

Addition of two's complement numbers follows the usual addition rules. There is no difficulty provided that there is no overflow (i.e., the numbers add to more than 3_{10}) or underflow (i.e., the numbers add to less than -3_{10}). However, those are issues for any finite precision representation. There are only so many values that can be represented with a finite number of bits.

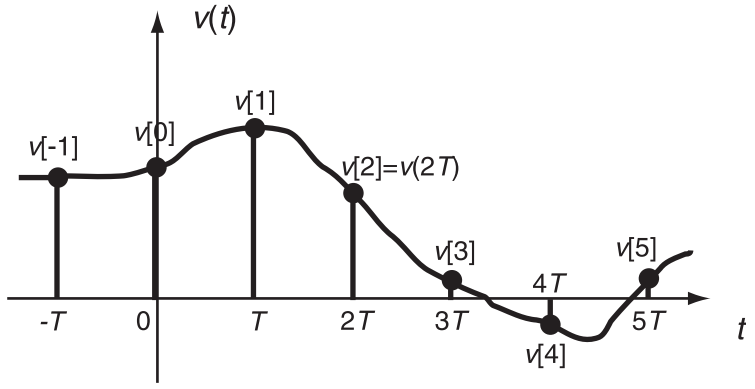
Using two's complement binary, the uniform, midstep 3-bit quantizer has the following representations for each quantization level:



Question: What value of $v_q(t)$ is represented by 110? What range of values of $v(t)$ corresponds to 110?

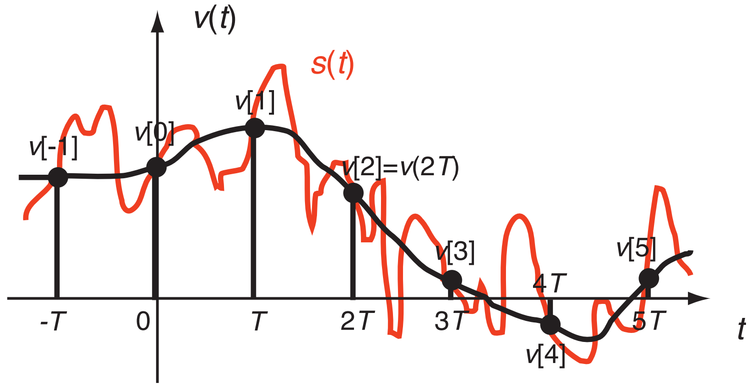
Sampling

We still don't quite have our signal data in a form that a computer can store and process. Currently, we have continuous-time signals that are defined over all values of *t*, while computers are designed to work with sequences of numbers. The solution is to *sample* our signal by only keeping values every *T* seconds.



The function $v[n] = v(nT)$ is known as a *discrete-time signal* and is only defined for integer values of n. We will further distinguish discrete-time signals from continuous-time signals by using square brackets (i.e., [.]) around their time variable. Note that if it has not been quantized, then a discrete-time signal is allowed to take on a continuous range of values.

Going from $v(t)$ to $v[n]$ is a process called *sampling*. As long as $v(t)$ is sampled quickly enough and does not vary too quickly, then it can easily be reconstructed from $v[n]$. However, sampling does introduce some ambiguity into the representation of the signal, as there are always an infinite number of signals that will have those same samples.



Given the samples $v[n]$, most people would choose a signal similar to the smooth, slowly varying $v(t)$ as the continuous-time signal that generated those samples. However, that signal reconstruction is never unique, and $s(t)$ clearly would generate the same $v[n]$ when sampled every T seconds. However, for a signal that varies as quickly as $s(t)$, we can also reasonably assume that samples need to be taken more often to get a reliable reconstruction. In fact, reconstruction systems always pick the lowest energy signal (i.e., the signal with the least variability) that matches the samples.

The two key questions regarding sampled signals are:

- How often do we have to sample a signal so that we can reconstruct it?
- Given the signal has been sampled fast enough, how do we reconstruct it?

With regards to the first question, the answer is that the sampling rate can be surprisingly low. Formally, the sampling frequency, $1/T$, only has to be more than twice the highest frequency of the signal. If we consider a simple signal such as a sinusoid, this sampling rate is just slightly more than 2 samples per period.

Example: The sampling rate for the music on a standard audio CD is 44.1 kHz, or 44,100 samples per second. Consequently, the highest frequency contained in CD-quality music is approximately 22kHz. (In practice the sampling frequency is always somewhat greater than twice the highest frequency to account for real world limitations in the sampling and reconstruction process.)

Signal reconstruction is also a fairly simple operation achieved primarily through a process known as lowpass filtering. Lowpass filtering removes the faster varying components of the sampled signal while preserving the parts that change more slowly (i.e., that are of 'low frequency'). Although this result may not be intuitive, for a signal that has been sampled appropriately, retention of these low frequency components is all that is required for reconstruction of the original signal.

Scaling

As we've seen, quantizers assume an input signal over the range v_{min} to v_{max}. Best results are achieved when the input signal is *scaled* to match the input range of the quantizer as closely as possible. If the signal's range is too small, then we are not taking full advantage of the quantizer's dynamic range. Consider the extreme case of an input signal that falls entirely between $-\Delta/2$ and $\Delta/2$. A midstep quantizer would completely eliminate this signal, since all of the signal's quantized values would map to zero. If the signal's range is too large, then values larger than v_{max} or smaller than v_{min} would be clipped to fall between v_{min} and v_{max}. If the signal has values falling considerably outside the quantizer's range, then distortions significantly larger than $\Delta/2$ can occur during quantization.

Scaling is also essential when programming the LEGO NXT such that the output hub from one sensor block feeds one of the input hubs of another sensor block. If the output data falls outside the range of the data hub it is connected to, then the input hub will either ignore the data value or change it to a value within its specified range. Quite likely, neither action is what the programmer intended when connecting these blocks.

As an example, consider programming your tribot to follow a dark line on a white background. As part of a line-following algorithm, you might want your tribot to turn left in response to a low light intensity reading (i.e., away from the black line) and to turn right in response to high light intensity levels (i.e., away from the white background).

The NXT's light sensor provides light intensity measures ranging from 0 (completely dark) to 100 (very bright). The steering input for the NXT's Move block accepts inputs from -100 (turn sharply left) to +100 (turn sharply right). Let's scale the light intensity data to match the range expected by the Move block's steering input. (Note: for several reasons this approach is not quite what you should do for a good line following algorithm, but you will learn more about that in the lab covering the light sensor.)

A simple scaling technique will take the input intensity reading, i, and change it to $s = ai + b$ for some constants a and b. Since we have 2 unknowns, we only need two equations to specify the constants. For $x = 0$, we want the steering s to be -100:

$$-100 = a(0) + b$$

For $x = 100$, we want the steering to be 100:

$$100 = a(100) + b$$

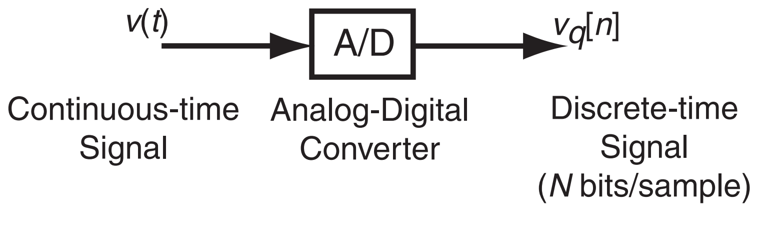
Therefore, $a = 2$ and $b = -100$ to give $s = 2i - 100$.

Questions:

1) What value of s corresponds to the tribot moving in a straight line? What is the corresponding light intensity reading? Where would this reading place the light sensor relative to the black line?
2) What should a and b be if you want to
   - reverse the steering (i.e., dark → turn right and bright → turn left)?
   - use less extreme steering such that $-50 \leq y \leq 50$?

Summary

The complete system for sampling and quantizing a continuous-time signal is commonly represented as an analog-digital, or A/D, converter.



Despite the sampling and quantization, where almost the entire original signal is thrown away, $v(t)$ can still be reconstructed very closely. All of us have had experience with CD quality music that consists of 44,100 samples/second and 16 bits/sample but exhibits very little distortion when compared to classic analog recording techniques. ECE 2025 (Introduction to Signal Processing) covers these topics in much more detail.