Consider the problem of using voltages to represent the information in a black-and-white image.

Each (x,y) point in the image has an associated intensity: black is the weakest intensity, white the strongest.

An obvious voltage-based representation would be to encode the intensity as a voltage, say 0V for black, 1V for white, and some intermediate voltage for intensities in-between.

First question: how much information is there at each point in the image?

The answer depends on how well we can distinguish intensities or, in our case, voltages.

If we can distinguish arbitrarily small differences, then there's potentially an infinite amount of information in each point of the image.

But, as engineers, we suspect there's a lower-bound on the size of differences we can detect.

To represent the same amount of information that can be represented with N bits, we need to be able to distinguish a total 2^N voltages in the range of 0V to 1V.

For example, for N = 2, we'd need to be able to distinguish between four possible voltages.

That doesn't seem too hard — an inexpensive volt-meter would let us easily distinguish between 0V, 1/3V, 2/3V and 1V.

In theory, N can be arbitrarily large.

In practice, we know it would be quite challenging to make measurements with, say, a precision of 1-millionth of a volt and probably next to impossible if we wanted a precision of 1-billionth of a volt.

Not only would the equipment start to get very expensive and the measurements very time consuming, but we'd discover that phenomenon like thermal noise would confuse what we mean by the instantaneous voltage at a particular time.

So our ability to encode information using voltages will clearly be constrained by our ability to reliably and quickly distinguish the voltage at particular time.

To complete our project of representing a complete image, we'll scan the image in some prescribed raster order — left-to-right, top-to-bottom — converting intensities to voltages as we go.

In this way, we can convert the image into a time-varying sequence of voltages.

This is how the original televisions worked:

the picture was encoded as a voltage waveform that varied between the representation for black and that for white.

Actually the range of voltages was expanded to allow the signal to specify the end of the horizontal scan and the end of an image, the so-called sync signals.

We call this a “continuous waveform” to indicate that it can take on any value in the specified range at a particular point in time.

Now let’s see what happens when we try to build a system to process this signal.

We’ll create a system using two simple processing blocks.

The COPY block reproduces on its output whatever voltage appears on its input.

The output of a COPY block looks the same as the original image.

The INVERTING block produces a voltage of 1-V when the input voltage is V, i.e., white is converted to black and vice-versa.

We get the negative of the input image after passing it through an INVERTING block.

Why have processing blocks?

Using pre-packaged blocks is a common way of building large circuits.

We can assemble a system by connecting the blocks one to another and reason about the behavior of the resulting system without having to understand the internal details of each block.

The pre-packaged functionality offered by the blocks makes them easy to use without having to be an expert analog engineer!

Moreover, we would expect to be able to wire up the blocks in different configurations when building different systems and be able to predict the behavior of each system based on the behavior of each block.

This would allow us to build systems like tinker toys, simply by hooking one block to another.

Even a programmer who doesn’t understand the electrical details could expect to build systems that perform some particular processing task.

The whole idea is that there's a guarantee of predictable behavior:

If the components work and we hook them up obeying whatever the rules are for connecting blocks, we would expect the system to work as intended.

So, let's build a system with our COPY and INVERTING blocks.

Here's an image processing system using a few instances each block.

What do we expect the output image to look like?

Well, the COPY blocks don't change the image and there are an even number of INVERTING blocks, so, in theory, the output image should be identical to the input image.

But in reality, the output image isn't a perfect copy of the input.

It's slightly fuzzy, the intensities are slightly off and it looks like sharp changes in intensity have been smoothed out, creating a blurry reproduction of the original.

What went wrong?

Why doesn't theory match reality?

Perhaps the COPY and INVERTING blocks don't work correctly?

That's almost certainly true, in the sense that they don't precisely obey the mathematical description of their behavior.

Small manufacturing variations and differing environmental conditions will cause each instance of the COPY block to produce not V volts for a V-volt input, but V+epsilon volts, where epsilon represents the amount of error introduced during processing.

Ditto for the INVERTING block.

The difficulty is that in our continuous-value representation of intensity, V+epsilon is a perfectly correct output value, just not for a V-volt input!

In other words, we can't tell the difference between a slightly corrupted signal and a perfectly valid signal for a slightly different image.

More importantly — and this is the real killer — the errors accumulate as the encoded image passes through the system of COPY and INVERTING blocks.

The larger the system, the larger the amount of accumulated processing error.

This doesn't seem so good.

It would be awkward, to say the least, if we had to have rules about how many computations could be performed on encoded information before the results became too corrupted to be usable.

You would be correct if you thought this meant that the theory we used to describe the operation of our system was imperfect.

We'd need a very complicated theory indeed to capture all the possible ways in which the output signal could differ from its expected value.

Those of us who are mathematically minded might complain that "reality is imperfect".

This is going a bit far though.

Reality is what it is and, as engineers, we need to build our systems to operate reliably in the real world.

So perhaps the real problem lies in how we chose to engineer the system.

In fact, all of the above are true! Noise and inaccuracy are inevitable.

We can't reliably reproduce infinite information.

We must design our system to tolerate some amount of error if it is to process information reliably.

Basically, we need to find a way to notice that errors have been introduced by a processing step and restore the correct value before the errors have a chance to accumulate.

How to do that is our next topic.