Chapter 0 – Automation and Control – The big picture
Revised: 17 September 2018

The multi-faceted world of control and automation systems

Often for a student it is hard to understand at the beginning just what the field of Control Systems encompasses. Students have studied electricity, electronics, dynamics, aerodynamics, thermodynamics, heat transfer, strength of materials, etc. Controls is none of that. So if you’ve already studied everything, what’s left? Well, to illustrate the importance of Control Systems, let’s cast a quick glance into the world of heavier-than-air flight. Two things led to the success of the Wright brothers in conquering the air. Needed was a light source of power. Flying around with a steam boiler and steam engine was just not going to work. The Wrights designed and built their own very light (by the standards of the day) gasoline engine that delivered 12 HP at a weight of just 180 lb. (See http://www.wright-brothers.org/Information_Desk/Just_the_Facts/Engines_&_Props/1903_Engine.htm for a detailed account of this engine.) The second barrier to surmount was control of the aircraft, that is the ability to steer it right and left. This was not well understood. Much of the thinking of the day was that a plane would be steered like a ship, i.e. by yawing it with a rudder. This turned out not to be the case. Unlike ships, airplanes can roll, and rolling to turn was much quicker in response than yawing to turn. It is not to be overlooked that the Wrights were bicycle builders and mechanics, and that one steers a bicycle not by turning the handlebars but rather, primarily, by rolling into and out of turns. The understanding of roll as a means to steer made controlled flight possible.

Control Systems are indeed hard to pin down. Two people saying those words can mean two completely different things. The problem is that the field of Controls is much broader nowadays than it was, say, 60 years ago, when we had simple, one-loop controllers, looking at, monitoring, and controlling a single process variable—like steam pressure, tank level, exhaust-gas temperature. The control function would often be implemented using mechanical or pneumatic hardware, or electrical hardware, though not electronics. There was no electronics in the early days of Controls. What happened in the 1960s is that the computer came on the scene. Married to the activity of control and automation, it greatly expanded the possibilities of what could be done with a single, isolated control loop. Many different things could be automated and controlled, and it became possible to change the behavior of the controller by re-programming it.

As computers became faster and cheaper, they spread and made many functions, that previously had been purely mechanical or hydraulic, smarter. A good example of this is the ABS (anti-lock braking system) for a vehicle. There was no such thing, or a very crude system, prior to the advent of today’s ubiquitous systems. Now they are required in vehicles. Computer hardware got cheap and faster, so this system was developed that requires quick thinking on the part of a monitoring and control computer. Indeed, many now characterize today’s automobile as more a rolling computer than primarily a mechanical device. And on the proximate time horizon, driverless cars, airplanes, and ships are on the planning boards. That is an idea hard to accept by many, but in vehicle crashes, most often the culprit is human error.

An example: skidding out on an icy road. The problem is that a human can only react so fast to encountering black ice. In the world of accident reconstruction, it takes about a second for a person even to recognize that he/she is in trouble, then it takes a half-second for him/her to respond. This is
too slow on black ice. He/She gets into a yawing oscillation. He/She reacts, but not fast enough. The car has gone too far by the time he/she reacts, so the reaction is exaggerated. The next oscillation in the opposite direction is even greater because of the slowness of the driver’s response. Eventually, after a few yaw-gyrations, the car has gone too far and spins completely around. An anti-skid system, on the other hand, recognizes by wheel slippage the danger...within milliseconds. It reacts also within milliseconds to correct the situation, so the car corrects before it gets out of control and does not skid out.

Thus, the vehicle’s sensors coupled with a fast computer saves the day. One huge advantage of electronic control, then, is simply the speed at which it can react.

Before diving into any particular niche of Controls, we are going to have a big overlook of the field, so as not to lose sight of the forest for the trees. There are a lot of trees in Controls, and it is easy to lose your way while you are focusing on the details of a particular part of it. There are two big, general areas in Controls: automation and then feedback control.

Automation is also sometimes called sequential control, because it is mostly the automation of a set series of steps that always must be undertaken to initiate a process sequentially. An example of a set procedure is a checklist used by pilots to take off in an airplane. Another example would be the procedure for starting a large diesel engine of a ship or bringing a prime mover in a power plant on-line to start generating power. In all of these cases, if even one step is overlooked or omitted, the consequences can be significant or even fatal. Before the days of automation, these steps would have to be manually performed, in a specific sequence, to ensure the proper start of the device. Nowadays, however, these processes can be automated, i.e. performed automatically by controlling and monitoring components in the system. The human operator can simply observe to ensure that the automated sequence is being executed as it should be.

Feedback control is a narrower topic. It is sometimes called continuous control, because fine adjustments are made in the device being controlled to ensure that some quantity keeps or follows a desired value. The classic example of this is a cruise control system in a car. We set a speed, say 65 mph, at it is the job of the control system to maintain that speed, even when the car encounters an uphill or a downhill grade. It gives more gas when an uphill grade is encountered, and it gives less or even brakes when the car encounters a descending grade. As we shall see, this is all done in a control loop. The device listens or watches (speaking loosely) with a sensor, then reacts accordingly through an actuating device, a component that causes the quantity controlled to change or to stay the same, depending on what the operator wants. So, for example, in a cruise control system, an automatic speedometer measures the car’s speed; then it pushes the accelerator to give more gas if the speed drops off, or it pushes the brake or downshifts if the speed gets too high. If the speed is right where it should be, it does nothing. Feedback control is named such, because the control loop looks at the quantity being controlled and feeds back a signal to do something to make the actual quantity controlled have the value desired.

Sequential control and automation systems
Sequential control automates a sequence of steps that have to be performed in a certain order. If you think of a written checklist for starting a piece of equipment, like the large diesel engine on a ship, many steps have to be performed before the engine is started. The steps have to be performed in a certain
order. Often there is a sensor that checks whether a condition exists (a motor spinning, a valve open) before the next step can start. In pure sequential control, input signals are either high or low (voltage > 0, voltage = 0) and output signals are also high or low. Sometimes “high” is indicated by the numeral 1, and “low” by the numeral 0. Thus, sometimes sequential control is called “digital control”, though that can also denote simply “run by a computer”. Sequential control is used heavily in manufacturing. In fact the hardware for sequential control, the programmable logic controller (PLC) was originally developed in the 1960s and 70s for automotive manufacturing. When manufacturing a complex object like an automobile, a long sequence of steps must be performed in order. But from its origin in the automobile industry the use of the PLC has spread to many more diverse manufacturing and operational environments.

**Example – Electric motor start circuit**

Let’s start simple. We want to push a button to start an electric motor. Since the motor draws a lot of current, we will start it by engaging an electrical relay to switch it on. The start button is spring-loaded, so that when you remove your finger, it goes back into the off position, and the relay disengages. We want to start the motor with the button, but we want the motor to stay on, even after the button moves back into the off position. We want the motor to continue running until we push another pushbutton to turn it off.

The action of having the motor continue to run, even after the actuating button returns to the off position, is called *latching*. We want to *latch the motor on*.

**Example – Tank level control**

Consider the following situation:

A tank is placed in a process plant to dispose of liquid accumulated in a process. The tank is emptied by a pump. The in-flow into the tank is controlled elsewhere. Unlike the steady tank level controller discussed in Chapter 3, here the tank level fluctuates between two extreme levels. You do not want the tank to overflow nor do you want the pump to run when the tank is dry. The tank is equipped with two level sensors—high and low. The control logic is:

- When the tank level drops below the low-level sensor the discharge pump is turned off and the red, low-level indicator light is turned on. The pump is latched off.
• When the tank level rises above the low-level sensor, the red, low-level indicator light is turned off.
• When the tank level rises above the high-level sensor, the pump is turned on and the yellow, high-level indicator light is turned on too. The pump is latched on.
• When the tank level falls below the high-level sensor, the yellow, high-level indicator light turns off.

Hardware – PLC
To automate sequences, even simple ones like the two described here, nowadays the specialized computer for automation tasks is a **programmable logic controller** (PLC). PLCs were developed in the 1960s and 70s to replace hard-wired relays and sequencers. This was still in the early days of computer development. PLCs were just specialized industrial computers. One feature they had in the early days is that they were ruggedized for the industrial environment. One other feature is that, like other computer processors, in the course of time the tendency is for them to operate ever faster.

The basic structure of a PLC is shown in the figure below. The PLC typically is in a cube-like box with **pins** for input and output. The PLC shown has 8 inputs and 8 outputs, but an industrial PLC can have many more inputs than outputs. Sometimes inputs and outputs are designated DI:01 or DO:01 instead of just I:01 and O:01 to indicate that the voltage on the pin is **digital**, i.e. **binary**, i.e. either off or on. So if an input pin sees a non-zero voltage, this is a positive input. We also say, “Input pin 1 is high.” We designate a non-zero input or output voltage as 1 and a zero input or output voltage as 0.

![Figure 0.1 – Basic PLC configuration](image)

PLC inputs often are wired to pushbuttons. The outputs are wired to relays that switch on motors or open valves, for example. Outputs can also be wired to indicator lights to show whether a piece of machinery is operating or whether a particular alarm is on.

Larger PLC systems can have separate modules. Figure 0.2 shows a PLC system with a separate I/O module and power supply. The power supply typically takes 110 VAC power and produces DC power to the operation of the PLC. Typical voltages for PLCs to operate are 24 VDC, 10 VDC, 5 VDC.
In this example, the number of digital inputs is large \((m)\), as is the number of digital outputs \((n)\). The I/O module may be located at some distance from the PLC. In fact usually it is located near the equipment being sensed and controlled, so that the various connections to and from the I/O can be made with short wire runs. The PLC will be located in a cabinet near a control station, so that it can communicate with a *Human/Machine Interface* (HMI). This is a computer screen or panel of some type that illustrates to the user state of the machinery being controlled. Figure 0.3 shows the propulsion-control station for the TV State of Maine.
The I/O modules can be connected to the PLC in the control room via a Fieldbus connection. This is a single cable over which data packets travel. The individual inputs and outputs are addressable, which means that the packet of data that travels includes actual data but also the address of the source of the data (if input) or the destination of the data (if output). So, for example, if DI:07 is high, this information is contained in a data packet that also identifies DI:07 as the source of that information. If the PLC wants to set DO:05 high, it puts that intent on the bus along with the address of the pin to set high. Since the Fieldbus carries all of the data back and forth in a single cable, the timing of sending and receiving is coordinated in a rather involved scheme. There are various Fieldbus systems; the data traffic control in each is called its protocol.

In the early days of PLCs only digital inputs and outputs were used. But as PLCs became more sophisticated, they were able to handle analog input and output. This means that instead of sending or receiving just a 0 or a 1, they could send or receive a voltage or current, whose value indicated the level of pressure or temperature sensed by a sensor or the location of an object based upon a displacement sensor (in the case of input) or some command voltage or current (in the case of output). Figure 0.4 shows a PLC with 6 digital inputs and outputs and 2 analog inputs and outputs. One must remember that in the world of Controls, digital refers to an on/off state while analog refers to a range of continuous values.
The language – Ladder logic

The native language for PLCs is *ladder logic*. This is a graphical language that maps inputs to outputs through digital logic. When a program is written, it looks like a ladder, hence the name. Figure 0.4 shows a very simple *ladder-logic diagram*, i.e. PLC program. There is an *input rail*, to which all of the inputs are attached and an *output rail*, to which all of the outputs are attached. Between the two rails is a *rung*. There can be multiple rungs between the rails. Logical flow passes from input to output if it can.

On each rung can appear, on the left, *make or break* commands that allow or prevent flow to the right side of the rung. On the right side of each rung is a particular output, which turns on or off depending upon whether it receives flow.

Figure 0.4 – Simple ladder-logic program

The open symbol on the left (written as - || -) is a *make* instruction. If the pin associated with it is high, it closes and allows flow through to the right. In our simple program above, if I:1 were high, then O:1 would be set high. If I:1 were low, then flow would be stopped, and O:1 would be set low.

Figure 0.5 shows the bigger picture. I:1 is connected to a normally open (NO) pushbutton and the output (O:1) is connected to an indicator light. As shown, with the pushbutton not pushed, pin I:1 is low and the make instruction *unmade*. So the light does not illuminate. If the pushbutton is pushed, pin I:1 is set high, so the make instruction is made and the light illuminates so long as the button remains pushed.

Figure 0.5 – Ladder-logic program with physical input and output

Figure 0.6 shows a similar case, except here the input pushbutton is normally closed. Like above, when I:1 is high, the make instruction is active and closes. In this case, then, the light is lit with the button unpushed. When the button is pushed, I:1 goes low, and the make instruction remains unmade and logical flow is blocked there.

Figure 0.6 – Ladder-logic program with NC input
Figure 0.7 shows a case that works exactly as did the case shown in Figure 0.5. I.1 is a break command, which allows flow if the input on the pin to which it refers is low. With an NC pushbutton connected to I:1, the unpursed button lets I:1 be high. This breaks the flow at the -/|- instruction, as it is written with text symbols. Thus flow is stopped with an unpursed button. When the button is pushed, the break instruction executes and opens, disallowing logical flow through it.

The introduction above shows the relationship between what happens with input buttons, how they affect the make and break instructions, and how that is passed on to the output and indicator lights, if that is what is connected to the output relays. In a real, industrial application, however, inputs often come from sensors or switches that measure tank level, pressure, the presence of something, etc. So instead of a button being pushed, a temperature may exceed a certain level, and that makes a temperature switch go from a low output to a high output. Outputs often go to relays that start motors or open valves, for example. There are still manual pushbuttons and switches for inputs, but there are also sensors that send electrical highs and lows to the input pins of a PLC. Likewise, there are still indicator lights attached to the output pins of a PLC, but there are also motor and valve relays, for example, also attached to the PLC output pins. This allows for a complicated sequence of events in industry that allow for tanks to be filled or emptied, motors to be started or stopped, tank contents to be heated or cooled, robot arms to be moved, conveyors to be run, etc.

In the world of automation and control, it is absolutely necessary to be able to distinguish inputs from outputs. Sensors and switches are connected to inputs, because the PLC or control computer needs to keep a constant “eye” on the device or process to know what its state is. The PLC or control computer sends output signals to actuators to affect the process or device being controlled. Thus

- Sensors and switches → inputs
- Outputs → actuators

In developing a ladder-logic program, it is useful to use identifiers that have some physical meaning. For tank level, for example, it’s useful to designate it by preceding the name with “L”, which stands for “level”. For temperature, “T”. So, for example, LLO would mean “low level”. THI would mean “high temperature”. MTR would probably mean “motor”. PMP would stand for “pump”. VLV would stand for “valve”, or VIN would stand for “valve in” and VOUT for “valve out”. If such a consistent scheme is developed and employed, one can develop a ladder-logic diagram that can be devised, read, and checked with ease. Thus the rung in Figure 0.8
says, “If the tank is full and the inlet valve isn’t open, run the pump”. Without knowing anything about the particular process involved, with a consistent set of variable names, one can determine the logic of a particular scenario. This rung refers to no particular input pins (I:1, I:2, etc.) nor output pins (O:1, etc.) but directly to the players involved in the automation studio. When writing ladder logic for a particular application, you should first write it out with these physical variables, then make a table to connect these with particular PLC pins. The point is that that is a separate step from working out the logic. And the logic should be worked out with meaningful variable names, not with I:1, I:2, etc., O:1, etc. That methodology is very prone to error.

The make and break instructions can be used to convert the ladder-logic diagram into something meaningful in your mind. There are some basic constructions that carry meaning that are worthwhile to master. For example Figure 0.9 says “if the pump is not on”.

Figure 0.9 – “If the pump is not on”

The “if” comes from the fact that the uprights in the symbol are ||, not ( ). || implies an input, i.e. a condition that is used to determine the state of an output. Thus, purely as an input, it is an “if”. Logic will pass through the rung at this symbol, only if the thing it references is off. What’s referenced is a PMP, i.e. a pump. Normally a pump is an output (- ( ) -), and we are turning it off or on. But with this reference, we are looking at its state and determining whether it’s on (- | -) or off (- |/ -). The diagonal mark in the symbol means “not”. So logic flows through this symbol only if the pump is not on. Thus, when reading ladder-logic diagram, when one encounters - |/ -, one says that the component referenced under the symbol is off or closed or not-yet-attained.

There are “phrases” in ladder logic that one sees again and again in different applications. In a rung, for example, the structure in Figure 0.10 is a logical AND structure. If both inlet valves are open the pump will start.

Figure 0.10 – Ladder-logic AND “phrase”

Another such “phrase” is seen in Figure 0.11, which represents a logical OR structure. Here the pump will turn on if either inlet valve is open.

Figure 0.11 – Ladder-logic AND
Another common ladder-logic phrase is the **latch**. We have a momentary pushbutton, we want to start something, but we want it to continue running once you remove your finger from the button. So the button starts the motor, but the motor being on keeps it on. This is called **latching**. Figure 0.12 shows this structure.

![Latch Diagram](image)

**Figure 0.12 - Latch**

For the moment, ignore the STP (stop) button. The STRT (start) button is momentary. When you push it, the motor turns on (logic passes through STP’s -\|/\-). But once MTR is on, the MTR input on the left \(-/\-\) allows the logic to pass through to MTR as an output \(-/-\) and keep it on. We need the STP button, so that when it is pressed, the break associated with STP opens, and MTR turns off.

Thus the very common latch structure consists of

- A rung and a sub-rung; the first element of the rung starts the output
- The sub-rung has a make with the same ID as the output; this is the latch; so we are able to use an output in a decision-making input function
- There must be a way to stop the thing referred by the output, here a way to stop the motor; this belongs just after the sub-rung comes back into the main rung. Setting the break high breaks the logic of the rung, so the motor turns off; this interrupts the latch too; the entire rung/sub-rung returns to a waiting state for another STRT

Thus in a latch the starting agent is on the main rung inside the sub-rung. The latch on the sub-rung refers to the output of the main rung. The component just after the junction of the sub-rung with the main rung is the stopping agent of the system.

**Continuous feedback control systems**

Continuous control systems differ from sequential control in that a particular physical quantity—temperature, pressure, location, speed, etc.—is monitored and compared with a desired value. If the actual value of that quantity is not equal to the desired value, action is taken to move the actual value toward the desired value. Thus continuous control consists of a feedback loop. The actual value of a physical quantity is sensed with a sensor. It is fed back and compared with the desired value and action is taken on this difference. There are two types of control loops: the regulator and the positioner or tracker. Sometimes the positioner is called a fly-by-wire system. A sensor senses a physical quantity, like temperature or pressure, and turns it into an electrical voltage or current. Thus a sensor is a transducer, in that it converts a physical quantity in one domain into a physical quantity in another domain (electrical). The electrical signal, whether it be voltage or current, characterizes the physical system that it represents. In other words, using pressure as a physical quantity to be measured, the greater the pressure, the greater the voltage or current that represents it.
Computers “talk” voltage. That is the syntax of the language that they “speak” and “hear”. And the preference is usually voltage. A physical range of values—for example, 1500-5000 psi—is mapped to a range of voltages—for example, 0-10 volts DC. Thus, for the example given, 1500 psi = 0 VDC and 5000 psi = 10 VDC. When the sensor is bought, installed, and set-up using your Android device, this correlation of values is set up as part of the set-up procedure.

In our realm of continuous control, since the value read in is a voltage that can vary in a continuous fashion (not just off/on) and the value sent out can also vary continuously over a range, continuous control is also referred to as analog control. In the world of Controls, analog means a continuously variable range of voltages, whether input or output. So continuous control is also often referred to as analog control. Thus digital control refers to sequential, on/off control, and analog control refers to continuous, feedback control.

There are two basic types of feedback control loops: 1) the regulator and 2) the positioner. Both of these loops contain the same components arranged in the same way. But the operation of each is a bit different. This is described below. First, however, it is important to distinguish between two broad areas of application of control systems: 1) motion control and 2) process control. Motion control involves controlling acceleration, velocity, or displacement (location). Motion control is used heavily in manufacturing applications and robotics. Fluid-power systems, since they control motion, also fall within this realm when they are automated. Technically, since a cruise-control system governs velocity, it is a motion-control system. Process controls on the other hand are found in any installation that involves the flow of fluids—power plants, refineries, chemical plants, etc.

In general, there are more positioners used in motion control than in process control, and there are more regulators used in process control than in motion control. But this is a very general statement with many, many exceptions. A cruise-control system is a regulator, yet it is used here in a motion-control application. Likewise, valves play a large role in process control, yet often a valve’s position—83% open, for example—is set by a position control loop.

Example – Regulator: cruise control
A cruise-control system on an automobile is a regulator loop. Its purpose is to keep the speed of the car steady, at a desired speed or setpoint in the face of disturbances that tend to change the speed from that desired value. The disturbances are primarily changes in the incline of the terrain. For example, you set the setpoint by getting the car up to a desired speed, whereupon you engage the cruise control. At that moment, the car’s central processing unit (CPU) reads the speed from a speed sensor and stores it. Then the CPU continues to monitor the speed to make sure it stays at the desired level.

If the car then encounters a hill, the speed will start to drop off. This deviation in actual speed from the desired speed will be noticed. A control loop running in the CPU will take action on this deviation in actual speed from desired speed. It will send a signal to open the throttle of the car to add more air and fuel to increase the speed of the car.

Most cars only control the speed one way. That is, they do not brake the vehicle if it starts to travel too fast, for instance when it encounters a descent. But many hybrid and electrical vehicles have what is called regenerative braking. The electric motor that drives the vehicle can also function as a generator. The electric field in which the motor turns can be increased, so that it slows the vehicle but also...
generates electrical energy for storage in the battery. There are also in some newer cars braking control to slow the vehicle by downshifting the transmission to a lower gear.

The characteristic of a regulator vs. a positioner is that the setpoint of the control loop stays constant; disturbances are present that tend to knock the actual value off of its desired level.

**Example – Positioner: fly-by-wire system**

A fly-by-wire system in an airplane is basically a remote-control, force-multiplying system. An older aircraft and small general-aviation aircraft have control surfaces—ailerons for roll, the elevator on the tail for pitch, and the rudder for yaw—mechanically connected to the input yoke or stick and rudder pedals via mechanical cables and/or linkages. With the advent of the jet engine and the increase in size of aircraft immediately after World War II, the force needed to deflect the control surfaces into the airstream increased significantly. The *fly-by-wire* system was developed to overcome this problem.

With a positioner, the input to the loop is the setpoint, which is constantly changing. In a fly-by-wire system, the direct connection between the input devices in the cockpit—the yoke or stick and the rudder pedals—is cut. The yoke/stick and the rudder pedals just become devices that express the pilot’s wish. If the pilot wants the aircraft to climb, he/she pulls back on the yoke or stick. A sensor detects this movement and sends an electrical signal back to the elevator on the tail of the airplane. There a hydraulic actuator applies power to rotate the elevator so that its trailing edge moves upward. This causes the tail of the aircraft to move downward, increasing the airplane’s angle of attack, which causes it to climb. Pushing on the yoke/stick has the opposite effect. The elevator rotates so that its lower surface is deflected into the airstream. This causes the tail to deflect upward, which give the aircraft a descending attitude, and it thus descends.

Characteristic of a positioner loop is that the setpoint of the loop is changing on a regular basis, as the operator drives the loop through various desired values. The deviation between actual and desired values is generated by changing the desired value often. The loop must respond so that the actual value *tracks* the changing desired value. Sometimes a positioner loop is called a *tracker*, because the job of the loop is to have the actual value track the changing desired value. Also with a positioner, there are normally few disturbances acting to change the actual value from its desired value.

**Example – Drive-by-meat system**

Believe it or not, I actually read somewhere the phrase “drive-by-meat” system to describe an ordinary human being driving a vehicle down a road. “Meat” seems to be popular to signify an ordinary human being. What “drive-by-meat” wants to express is that a human (“meat”), when serving as sensor (looking at the speedometer) and responding to that (pressing on the accelerator or brake), is actually a controller+actuator. If a human is flying an airplane, he/she is a fly-by-meat system. Thus you can go back home and brag to your friends and loved ones that you are a drive-by-meat system!

**Example – Human body temperature control**

This is not *control-by-meat* but rather *control-of-meat!* Your body temperature stays at or near 98.6°F, regardless of the temperature of the air, whether it be -40°F or +115°F. How does it do this? Are you so smart that you know how to make this wonder work? No, it is automatically built into human beings. If you become too hot through physical exertion or because the surrounding temperature is high,
evaporative cooling is turned on to lower your temperature. That is, you sweat. What happens if your body temperature falls below your built-in setpoint?

Warm-blooded animals and humans have this temperature-control system and thus can remain active and functioning even in cold weather. Cold-blooded animals become inactive during cold parts of the year, when their body temperature sinks to a level that does not allow them to function any longer.

The control loop
See Chapter 1 of Control Systems Engineering: A Practical Approach.

The language – Block diagrams
A dynamic model of a physical system is expressed graphically in a sketch called a block diagram. A block diagram shows the various components of a system in blocks connected by lines called signals. A block has a single input and a single output. Figure 0.8 shows a block with its input and output signals. In this simple case, the block content is just a constant. The block takes the input signal and multiplies it to produce the output. If the input signal changes, the output signal changes also. It is also said that the block transfers the input to the output, though I have always considered that a strange way of talking. Thus the block content is called the transfer function of the component in the system that it represents.

![Figure 0.8 – Simple block with I/O signals](image1)

A constant is the simplest kind of transfer function that can go in a block. A transfer function can be more complicated. For example, integration is another type of transfer function. Figure 0.9 shows an integrator block. As time progresses, the input $x(t)$ changes in value. The integration of this input quantity over time represents the shaded part under the $x(t)$ curve at any point in time.

![Figure 0.9 – Integrator block](image2)

A differentiation block differentiates the input. Thus its output is the slope of the curve at any point in time. Figure 0.10 shows a differentiation block.
The conventional symbol used for a transfer function is $G(s)$. The “s” comes from the world of Laplace transformations, which are at the core of the simulation operation in dynamic modeling. Nowadays the drudgery of taking Laplace transforms or figuring out inverse Laplace transforms is avoided by the numerical simulation that is part of Simulink, Modelica, and other simulation software. But the “s” is still there as a relic of the Laplace transform solution procedure. In the $s$-domain, integral and differential operations are converted to algebraic operations. Specifically, integration is equivalent to division by $s$. Differentiation is equivalent to multiplication by $s$. So the integration and differentiation operations often appear as shown in Figure 0.11.

![Figure 0.10 – Differentiation block](image)

**Figure 0.10 – Differentiation block**

Actually, Simulink is inconsistent in its notation and writes the content of the differentiation block as $\frac{du}{dt}$ rather than as $s$. As a user of the software you need not get involved in math of Laplace transforms because it is transparent to you. But you should know what is logical to expect when using these two blocks and other blocks that we shall see later. All of the drudgery of the mathematics is removed, done by the software.

More can be found about block diagrams and block-diagram algebra in Appendix A of *Control Systems Engineering: A Practical Approach*.

With dynamic-modelling software—e.g. Simulink and Modelica—one can build block-diagrams models of systems and then simulate how they will respond to various inputs. For example, one could build a block-diagram model of a vehicle and the controller used in a cruise-control system. One could “run” car up and down hills, over roads with various surfaces, try out engines of various powers...all without cutting a single piece of metal or assembling any collection of parts. And, indeed, that is precisely what automotive designers do when conceiving a new vehicle. In the past, before the days of ready-at-hand computer simulation, manufacturers had to produce a number of prototype vehicles before getting the details of the production vehicle fixed. Today they will still produce prototypes for trial before going into production, but the number of prototypes needed to finalize a design is much less than it was before the days of programs like Simulink and Modelica. This holds true for any type of vehicle—aircraft...
and marine vessels too. The dynamic performance characteristics of these vehicles is largely worked out through testing simulations of them before the manufacture of any prototype takes place.

**Integrated control systems**

The two basic types of automation and control systems usually are part of a larger automation and control system for the manufacturing or production facility that they control or the vehicle whose movement they control. Regarding vehicles, they have become veritable rolling/flying/floating computers. Modern automobiles have as many as 50 microprocessors onboard! The individual loops and the sequential control tasks form an interconnected environment. What happens in one process or sequence might affect the control of some other process. Hence it is advantageous to have a supervisory level above the individual loops and sequences to coordinate and monitor the overall system. There are various schemes for doing this, some older than others. As time has marched on, the lines between these supervisory systems have been blurred, so that they all look quite similar and perform similar functions in an integrated environment. One thing that has happened, however, is that the sensors and actuators used in automation and control have acquired ever more computing capability locally. We talk of “smart” sensors and actuators. So where dumb field devices used to have to be managed from a central location, smart devices take on more of the control locally. So the “intelligence” of the collective system has gotten to be ever more distributed or local. We tell an actuator to do something, locally it ensures that what was commanded has actually been accomplished, and it sends confirmation back to the supervisory system. Local intelligence in an actuator or sensor also monitors its own health. It runs self-checks and reports a degradation or failure of function back to the supervisory software so that action can be taken for a repair or replacement. You probably already are familiar with this via the check-engine light in your car. Some fault has been detected by a smart device and reported to the supervisory system in the car.

The evolution of automation and control from manual controls to fully integrated controls did not happen in one single bound. In many older plants you find manual controls with islands of automation here and there in the facility. For example, on an older ship, many of the controls will still be manual—operated with levers and hand-wheels—but isolated systems, like the air-conditioning system, might be fully automated. So although some systems are fully automated, the entire galaxy of controls in the engine room and bridge is not. You wind up with a hybrid system but with no overall integrated control system.

**SCADA systems**

SCADA systems are old systems. But they have been modernized. SCADA stands for Supervisory Control and Data Acquisition. The original SCADA systems were mostly remote monitoring systems. For instance back in the 1980s I knew an engineer who was monitoring all of the dams on the Penobscot River from his house, using a SCADA system delivering operating data over phone lines. But SCADA systems have grown up and become much more sophisticated over the years. Today’s SCADA systems integrate the operation of PLCs and controllers, include a sophisticated HMI and allow the operator to monitor and operate the plant remotely. Today’s SCADA systems have grown to be very much like DCSs (see below).
Distributed control systems (DCS)

Distributed control systems evolved to replace centralized control rooms in powerplants, refineries, and other process facilities. Since steam ships where just smaller, mobile steam plants, these also had central, hard-wired control stations for monitoring and operating the vessel’s turbine or diesel engine. Pre-DCS control rooms consisted of hard-wired and plumbed levels and steam gauges to monitor the plant’s performance. As computing hardware and became cheaper and smaller, DCSs were developed as more flexible stations for monitoring and controlling the plant. These have evolved over the last few decades into fully integrated systems with advanced controls and capabilities.

Fieldbus systems

Fieldbus systems refer to the layout of the sensors and actuators and their connection with the HMI, the interface for monitoring and controlling the actuators. “Bus” is the key part of this phrase. It refers to the way that everything is wired.

Prior to computer controls, single-loops were created with wiring directly from the controller to the actuator and wiring back from the sensor to the controller. Actually, in many cases, the controls were pneumatic rather than electrical. There were even pneumatic “computers”, made from linkages, bellows, etc., that would take action based upon the pneumatic signals that were received from the field. “Field” refers simply to the plant that is being automated. With electrical, hard-wired, single loops, the standard came to be a 4-20 mA current loop. Current was used to transmit a signal instead of voltage, because voltage drops during signal transmission over distances, whereas current remains constant. But the big problem with single, hard-wired loops was that there was an enormous amount of wire needed to connect field devices with the gauges and control levers and buttons in the control room. The bus architecture solved this problem.

As was mentioned briefly above in the discussion of PLC I/O modules, with a bus architecture, there is a single cable or a pair of cables that connects everything. Figure 0.12 shows this bus architecture. All components—sensors and actuators—plug into a bus. The individual components are addressable, that is they have their own unique identifier to specify the source of data or the destination for operating commands. The information on the bus is in data packets. This means that the data—signals that indicate temperature, pressure, etc.—is accompanied by an address that tells from what sensor it came. If the data is a command for an actuator—for example, to tell a valve to go to 50% open—the command will be accompanied by the address of the actuator for which it is intended.

Figure 0.12 – Bus architecture
Sometimes the bus is referred to as the *data highway*. Experts in the field talk about *data conflicts*, *data traffic control*, etc. The way data packets flow to and fro on the network is not trivial. There is a *data protocol* to handle the packets and their movement up and down the bus. In the early days of Fieldbus systems, different developers of these systems made up their own schemes from data transmission. There were no standards. Thus a veritable Tower of Babel was built, with manufacturers of components being hard pressed to guarantee compatibility with the various bus protocols of different DCS suppliers. Since then, however, a good deal of work has been done to develop standards that guarantee data compatibility on a bus network. A consortium of DCS vendors joined together to create the *Fieldbus Foundation*, an organization that promoted standards for components, so that they would “speak” a data dialog that was understandable on a Foundation Fieldbus network. As components—actuators and sensors—have gotten “smarter”, the amount of data travelling a network has increased, and this has made the adoption of standards for bus-architecture systems more imperative.

**CAN-bus systems**

CAN stands for Controller Area Network. It was introduced by Bosch in 1986 as a protocol to allow Electronic Control Units (ECUs) in vehicles to communicate via a bus architecture. ECUs are self-contained control units in a vehicle dedicated to a particular task. Typical ECUs in a vehicle include the Engine Control Module, Transmission Control Module, Brake Control Module, Suspension Control Module, etc. For example, a light-control unit turns on the headlights and taillights, the turn signals, the fog lights, the brake lights when the brakes are applied, etc.

![Engine Control Unit](image)

**Figure 0.13 - Engine Control Unit, one of the principal Electronic Control Units in a vehicle**

Figure 0.13 shows an engine control ECU. Pre-CAN ECUs had communicated via point-to-point wiring. This was becoming increasingly complex as the number of ECUs in a vehicle increased. Thus Bosch developed the CAN bus to simplify the wiring in a vehicle and to allow the different ECUs to communicate with each other. You are probably unwittingly familiar with a CAN bus. If something goes wrong with your vehicle, the “check engine” light comes on. You take your car to a shop, and there they plug in a diagnostic tool that reads the fault codes to identify the problem. The mechanic is then accessing the CAN bus and the principal vehicle ECU to read off the faults.
Though the CAN bus was originally developed for automotive use, application of it has spread to many different small-scale systems. Its data protocol is well-developed and defined, and standardized componentry is available off-the-shelf.

**The language – P&IDs**

This is very confusing, because we already have PIDs as part of our study of continuous control systems. “PID control” means proportional/integral/derivative control, the most common algorithm in the world for controllers. “P&ID” means “process and instrument diagram”, the principal graphical rendition of how the parts and pieces of a process control system fit together. Figure 0.14 shows a P&ID for some chemical process.

A P&ID does not show the physical location of equipment; it only shows how the equipment is hooked together—the flow lines and then the instrumentation and connections between the instrumentation components and the process they monitor and control. Standardized symbology has been developed to make the graphics used in P&IDs more consistent than it was in the past.

Even though we do not know what this process is in Figure 0.14, we can determine much about it by knowing what the standardized symbols mean. For instance, pumps are labelled “P-“ followed by an identifying number. The valves are labelled “V-“ with an identifying number. Control valves, often the actuators in a process plant, have the small, half-dome hat on them. The circles with letters in them indicate the sensor and controller associated with the valves. For instance, we see the string LT-LC-I/P.
The “L” signifies that it is part of a level control. “LT” signifies “level transmitter”. “LC” means “level controller”. “I/P” signifies a current-to-pneumatic transducer, indicating that the valve is actuated by pneumatic pressure.

Machinery health-monitoring systems
While not part of a control strategy, much of the equipment in an industrial facility or onboard a ship or airliner is very expensive. Its failure represents a large loss for the owner or operator. Often too the down time is very expensive. If an airliner is sitting on the ground because one of its engines has a mechanical problem, the financial cost of its inactivity is enormous for the airline. So it pays to be on top of failures, even to be ahead of them by fixing things before they fail instead of after they fail. With more and more intelligence pushed down to a local, field level, this has become more readily available as the automation and control world has evolved. Machines and components can be designed to monitor their own condition and to announce an impending failure before it occurs. Often the chief strategy for machinery health monitoring is simply to regularly compare a parameter or a set of parameters to a benchmark condition that was taken when the machine was new and healthy. If vibration levels or temperature levels, for example, trend without big changes, then the machine is healthy. But if a sudden deviation occurs, this can be detected and reported to initiate a response to fix what’s wrong.

In the maintenance world, there are four types or levels of maintenance:

1. Reactive maintenance – One runs a machine or system until it fails. Then one reacts to the failure by making a repair or a replacement of the system or component that failed.
2. Preventive maintenance – One knows about when a part is expected to fail and replaces parts, lubricants, or worn components on a time schedule to preempt the failure.
3. Predictive maintenance – Equipment is outfitted with sensors that monitor the machine state. These sensors trigger a maintenance action when a deviation from normal behavior is detected.
4. Proactive maintenance – This is predictive maintenance on steroids. The mass of data generated by today’s smart sensors is analyzed thoroughly. Often the combination of data from sensors is analyzed statistically to identify impending failure in a more sophisticated way than is done with predictive maintenance.

The Internet of Things (IoT)
See Wikipedia article on IoT. This concept is fairly new. It arises from the idea that “things”—physical components—are nowadays so imbued with “intelligence”—that is, that they have onboard microprocessors—and they are all connected to the Internet or at least a plant’s intranet, that they can pretty much take care of themselves. So the Internet of Things refers to all things being networked and smart enough to take care of themselves with little or no intervention by humans. Humans enjoy the fruits of all this technology, while in the background all of the high-tech things we depend on and that make our lives easier and better are collaborating, announcing impending problems, scheduling repairs, ordering parts for those repairs, etc. A lot of this is just hype, but there is also some truth to it. Machines and systems with smart components can take care of themselves at a level of autonomy that was inconceivable not long before now. Many of your household appliances are endowed with WiFi connectivity that allows you to monitor their condition from your smart phone and even change settings
of these smart devices. And these capabilities will expand in the future with more and more intelligent devices and consumer products connected to the Internet.

In an industrial setting, machines and systems will be networked, and they can be monitored via cell phone. If a fault is detected, action can be taken—either by an intervening human or by the machines themselves—to avoid an unscheduled outage. If there are parallel pumps, for example, and one detects a deviation in trended health monitoring, it can report that, start-up the other pump, and report this event to plant operators, so that they can take action to repair the ailing pump.

Industry 1.0, 2.0, 3.0, 4.0
See Wikipedia article on Industry 4.0. This is a European—primarily a German—concept, closely associated with IoT. The notion here is that today’s cyber-physical world, where computing and connectivity are so integrated with physical systems and components, represents nothing less than a fourth industrial revolution. The first three industrial revolutions were:

1. The original industrial revolution where machines were invented to take over work that had previously been done by human or animal power. The invention of the steam engine was the chief enabling technology for this change.
2. Mass production and assembly-line manufacture. The first exemplar of this was Henry Ford’s development of systematic manufacture for the Model T Ford.
3. The computer revolution, when computers were harnessed to the task of automating manufacturing and system control.
4. Smart, networked components and systems, collectively referred to as the cyber-physical world.

The German government promoted Industry 4.0 as an imperative for government investment, and much money was made available for smart manufacturing. The term Industry 4.0 is thus nowadays in widespread use in Europe.

Homework problems
Problem 0.1 – AND and OR rungs
Set up a ladder-logic program with two switch (latched) inputs, which are connected to pins I:1 and I:2. Have two outputs too, O:1 and O:2, which are indicator lights. Let O:1 turn on if both switches are on (the AND function). Have O:2 turn on if either switch is on or if both are on (the OR function).

Problem 0.2 – XOR
Create a ladder-logic program that works exactly as described in the previous example. But add a third output XOR, which turns on only if one and only one of the inputs is high. If both inputs are high, XOR remains unilluminated.

Problem 0.3 – Latching a momentary pushbutton
Create a ladder diagram that uses a NO, momentary pushbutton as input to start a motor, but latch the motor on, so that it runs until a second NO, momentary pushbutton is pushed.

Problem 0.4 – Discrete tank level controller, version 1
Consider the following situation:
A tank is placed in a process plant to dispose of liquid accumulated in a process. The tank is emptied by a pump. The in-flow into the tank is controlled elsewhere. Unlike the steady tank level controller discussed in Chapter 3, here the tank level fluctuates between two extreme levels. You do not want the tank to overflow nor do you want the pump to run when the tank is dry. The tank is equipped with two level sensors—high and low. The control logic is:

- When the tank level drops below the low-level sensor the discharge pump is turned off and the red, low-level indicator light is turned on. The pump is latched off.
- When the tank level rises above the low-level sensor, the red, low-level indicator light is turned off.
- When the tank level rises above the high-level sensor, the pump is turned on and the yellow, high-level indicator light is turned on too. The pump is latched on.
- When the tank level falls below the high-level sensor, the yellow, high-level indicator light turns off.

Let the inputs and outputs accord with the following table.

<table>
<thead>
<tr>
<th>High-level sensor</th>
<th>O:1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yellow light</td>
<td>O:1</td>
</tr>
<tr>
<td>Low-level sensor</td>
<td>O:2</td>
</tr>
<tr>
<td>Red light</td>
<td>O:3</td>
</tr>
<tr>
<td>Pump</td>
<td>O:4</td>
</tr>
<tr>
<td>Green light</td>
<td>O:4</td>
</tr>
</tbody>
</table>

Also assume that the sensors send out a positive signal when they are uncovered and 0 when they are uncovered.

**Problem 0.5 – Discrete tank level controller, version 2**

Let's look at another type of tank level controller. The tank shown in the figure below has an outflow that is controlled from elsewhere. That is, the tank contains a liquid, a supply to a downstream process, so there must always be this ingredient on hand to mix to make the final product (it could be paint, ketchup, medication, etc.). The tank’s supply of the ingredient is supplied by an inlet, whose valve controls the level in the tank. If the level drops below a certain point, the valve opens. If it rises to a
high level, the valve closes. Since this is a discrete level control system, the valve is either open or closed.

The tank is equipped with a low-level sensor and a high-level sensor. These both go “high” when they are covered with liquid. That is, that is their “on” state. Thus, when the high-level sensor is on, the valve should be closed. When the low-level sensor is uncovered (off), the valve should open. It should latch open until the high-level sensor is again covered.

Can you write the ladder-logic code to effect this operation. Let the inputs and outputs accord with the following table.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>High-level sensor</td>
<td>I:1</td>
</tr>
<tr>
<td>Low-level sensor</td>
<td>I:2</td>
</tr>
<tr>
<td>Valve</td>
<td>O:1</td>
</tr>
<tr>
<td>Green light</td>
<td>O:2</td>
</tr>
<tr>
<td>Red light</td>
<td>O:3</td>
</tr>
</tbody>
</table>

Also assume that the sensors send out a positive signal when they are uncovered and 0 when they are uncovered. Assume too that the valve opens with a positive signal and closes with a 0 signal. Let the lights work as follows: the red light turns on when the valve is closed; the green light is on when the valve is open.

What would the tank level do if the output of the tank were at a steady flow rate?

Convert this system to a heating system for a house. In lieu of the valve actuator, let O:01 be connected to a burner system. If the burner is on, the green light is lit. If the burner is off, the red light is lit.

With heat leaking out of the house on a winter day, what the temperature profile inside the house look like as this system operates to keep the house warm? How would the temperature profile inside the house differ in these two cases: 1) a cold winter day vs. 2) a really cold winter day?
Problem 0.6 – Control loop configuration
Take a blank sheet of paper and draw a positioner loop from memory and a regulator loop from memory. Reconfigure the loop(s) as necessary so that the non-zero input to the loop is on the left and the correct output is on the right. Name the loop components in the boxes and label all signals.

Problem 0.7 – Cruise-control system
Draw a cruise-control loop for an automobile, labelling the specific components and signals used in this system. This is a regulator loop, so draw the loop with the disturbance as the input. A hint: The engine produces a torque, part of which is used to keep the car going at a constant speed on level ground. When the car encounters a hill, part of this torque is deducted to be used to store gravitational potential energy in the car as it rises in elevation, so less is available to keep the car at speed.

Problem 0.8 – Ship thrust control
The T.V. State of Maine has a controllable-pitch propeller to change the thrust exerted by the ship on the water, thus increasing or decreasing speed and also going in reverse. To go faster, the propeller pitch is adjusted to a higher angle to take a bigger bite out of the water with each turn of the propeller. But this loads the engine and causes it to turn more slowly. So there is an inner speed loop around the engine to sense this and increase its speed. Draw the two nested loops here to graphically illustrate the cause and effect of this and the components involved in these two nested loops.

Problem 0.9 – PID response
A PID controller, shown below with the controller gains given, is subject to the input shown over a period of 5 seconds. Determine the 5 intermediate signals indicated, using the blank graphs. Then calculate the overall U output.
Problem 0.9