Microcontroller-Based Measurement for Control Systems

1.0 Learning Outcomes....1
2.0 Microcontroller Measurement Inputs....2
3.0 Digital Inputs for Digital Measurement....3
4.0 Analogue Inputs for Analogue Measurement....3
5.0 Analogue Sensors: Datasheets and Filtering....5
6.0 Summary....7
Recommended Citation....7

1.0 Learning Outcomes

After studying this ACE-CORE guide, you should be able to demonstrate the Learning Outcomes listed in Table 1. On completion of the guide, revisit Table 1 and confirm whether each Learning Outcome has been achieved.

Table 1: *Learning Outcomes from the Microcontroller-Based Measurement Guide*

ID	Learning Outcome	Achieved
1.	Explain how the measurement block fits within a control system, and how a microcontroller implements it in practice.	☐
2.	Distinguish between digital inputs (logic levels and digitally encoded signals) and analogue inputs (continuous voltages converted by an ADC).	☐
3.	Interpret a digital measurement from a timing-based sensor, such as an ultrasonic distance sensor.	☐
4.	Explain how the 10-bit ADC of the Arduino Uno works, including reference voltage, resolution and quantisation, and calculate its resolution.	☐
5.	Use a sensor datasheet to convert ADC values into engineering units and to identify operating limits.	☐
6.	Explain the purpose of a low-pass filter (LPF) in measurement, and how it is implemented on a microcontroller as a difference equation.	☐

2.0 Microcontroller Measurement Inputs

A control system is commonly represented as a block diagram made up of a controller, an actuator, a process and a measurement device (sensor) in the feedback path, as shown in Figure 1. The Introduction to the ATmega328P Microcontroller guide focused on the output side, where the microcontroller drives an actuator. This guide focuses on the measurement side: how the signal from a sensor enters the microcontroller and is turned into a meaningful measurement, such as a distance in metres or a temperature in degrees Celsius.

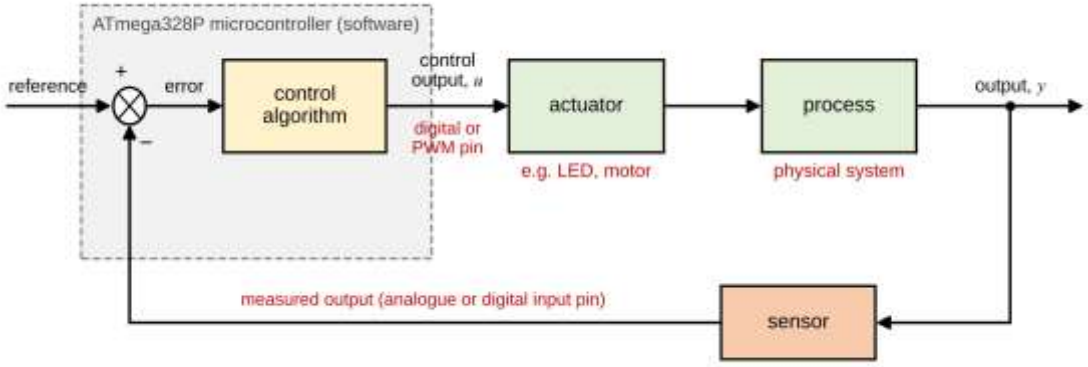


Figure 1: *How a Microcontroller Implements a Closed-Loop Control System*

Sensor signals reach the Arduino Uno R3 in one of two ways:

- through the **digital input pins**, as logic levels or timed pulses, or
- through the analogue input pins (A0 - A5), as a continuous voltage that is converted into a number by the analogue-to-digital converter (ADC).

For now, the comparison with the reference to form the error is not considered, as a complete closed-loop control system is not yet being built.

In either case, the raw signal at the pin is not yet a measurement. As Figure 2 shows, a sequence of steps is needed to turn it into one: the signal is sampled or timed, converted into a number, scaled into engineering units using the sensor datasheet, and often filtered to reduce noise. The rest of this guide follows this measurement chain.

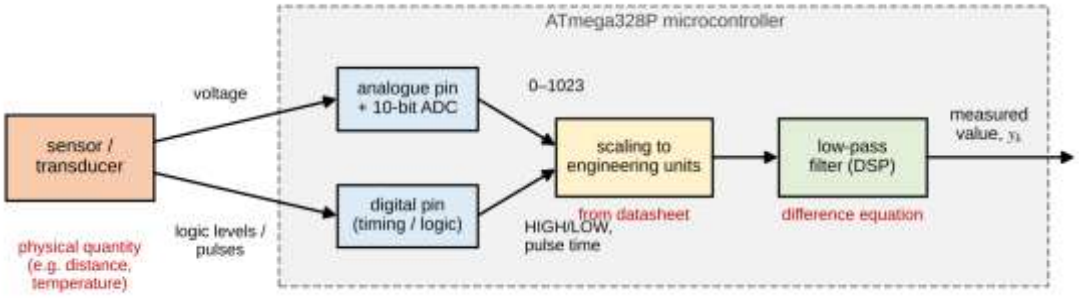


Figure 2: *The Measurement Chain from Sensor to Measured Value*

3.0 Digital Inputs for Digital Measurement

The digital I/O pins of the Arduino Uno were introduced in the Introduction to the ATmega328P Microcontroller guide. Here, they are used as inputs. A digital sensor communicates with the microcontroller in one of two ways:

- **Discrete logic levels:** the sensor gives a binary output, HIGH or LOW, to show whether a condition has been met, for example whether a switch is pressed or motion has been detected. The microcontroller simply reads the state of the pin.
- **Digitally encoded signals:** the sensor measures a physical quantity and sends the result in a structured digital form, such as a pulse whose width represents the value, or a serial data stream. The pin is then part of a timing or communication interface, and the microcontroller must capture and interpret the signal in software to reconstruct the measurement.

Example: Ultrasonic Distance Sensor

An ultrasonic distance sensor (such as the HC-SR04) is a common timing-based digital sensor. The microcontroller sends a short trigger pulse; the sensor then emits a burst of ultrasound and sets its echo pin HIGH until the reflected sound returns. The microcontroller measures the width of the echo pulse, t_{echo}, which is the time taken for the sound to travel to the object and back. The distance is therefore:

$$d = \frac{v\, t_{echo}}{2} \tag{1}$$

where $v \approx 343$ m/s is the speed of sound in air at 20 °C. For example, an echo pulse of 5.83 ms corresponds to $d = (343 \times 0.00583)/2 = 1.0$ m. The accuracy of the measurement depends on how precisely the microcontroller can time the pulse, and on the speed of sound, which changes with air temperature.

4.0 Analogue Inputs for Analogue Measurement

Analogue-to-Digital Conversion

The Arduino Uno measures analogue voltages using its built-in 10-bit analogue-to-digital converter (ADC). Although the voltage at an analogue pin (A0–A5) is continuous, the ADC converts it into a whole number that can be processed in software. By default, the ADC uses the 5 V supply as its reference voltage, V_{ref}, so that input voltages from 0 to 5 V are mapped onto $2^{10} = 1024$ levels. The output, N, is therefore a whole number from 0 to 1023 (note that 0 is one of the levels).

The smallest change in voltage that the ADC can distinguish is its resolution:

$$\Delta V = \frac{V_{ref}}{1024} \tag{2}$$

With a 5 V reference, this gives:

$$\Delta V = \frac{5}{1024} \approx 4.88 \text{ mV}$$

Any change in input voltage smaller than about 4.88 mV cannot be detected. Converting a continuous voltage into one of the 1024 levels is called quantisation, and it is illustrated in Figure 3. Quantisation introduces an unavoidable error of up to one level, because every voltage within a level is represented by the same number. A different reference voltage can be selected (for example, through the AREF pin) to change the range and the resolution.

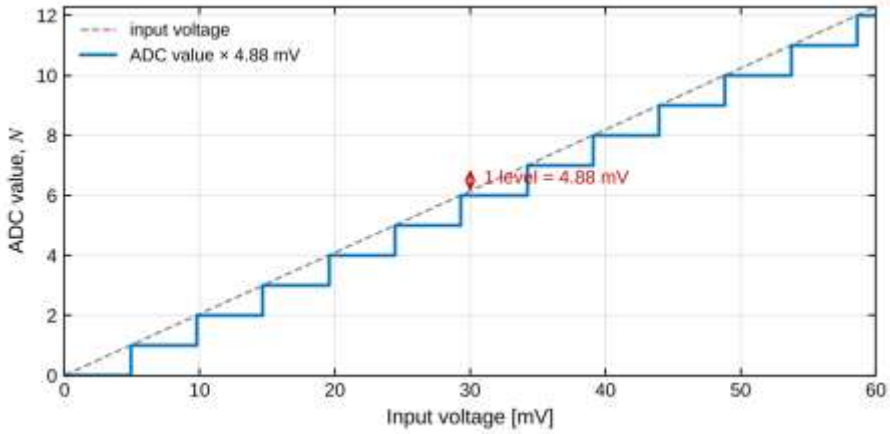


Figure 3: *Quantisation by a 10-Bit ADC with a 5 V Reference (First 60 mV Shown)*

Converting ADC Values into Engineering Units

The ADC value is converted back into a voltage using:

$$V_{in} = N \times \frac{V_{ref}}{1024} \tag{3}$$

The voltage is then converted into the physical quantity using the relationship given in the sensor datasheet (Section 5.0). The resolution of the ADC therefore sets the smallest change in the physical quantity that can be measured. Understanding resolution and quantisation is essential when deciding whether the Arduino Uno can measure a quantity precisely enough for a given application.

Example 1: Analogue temperature measurement

A TMP36 temperature sensor is connected to pin A0. Its datasheet states that the output voltage is 0.5 V at 0 °C and increases by 10 mV for every 1 °C. The ADC reads $N = 155$. Determine the temperature and the temperature resolution of the measurement.

Solution:

From Equation (3), the voltage is:

$$V_{in} = 155 \times \frac{5}{1024} = 0.757 \text{ V}$$

From the datasheet, the temperature is:

$$T = \frac{V_{in} - 0.5}{0.01} = \frac{0.757 - 0.5}{0.01} = 25.7 \text{ °C}$$

One ADC level is 4.88 mV, and the sensor changes by 10 mV per °C, so the temperature resolution is $4.88/10 \approx 0.49$ °C. Temperature changes smaller than about 0.5 °C cannot be detected with this arrangement.

5.0 Analogue Sensors: Datasheets and Filtering

Analogue sensors produce a continuous electrical signal, usually a voltage, that is related to a physical quantity such as position, temperature, pressure or light intensity. Unlike digital sensors, analogue sensors need their signal to be scaled, and often conditioned, before it can be used in a control system.

The Datasheet

Using an analogue sensor correctly depends on its datasheet. The datasheet specifies:

- the transfer characteristic, that is, the relationship between the physical quantity and the output voltage (for example, 10 mV per °C with a 0.5 V offset, as in Example 1)
- the operating range, sensitivity, linearity and tolerance
- the effect of temperature and the response time
- the electrical limits, such as supply voltage, current draw and output impedance

Without this information, it is not possible to scale the ADC value correctly, choose a suitable filter, or calibrate the sensor. The datasheet also defines the safe operating limits of the sensor, which is essential for a safe control system. It is therefore a primary engineering document that underpins correct modelling, hardware interfacing and reliable system performance.

Low-Pass Filtering

Measurements from many analogue sensors contain noise: small, rapid random variations that are not part of the quantity being measured. If the noisy measurement is fed directly to a controller, the controller reacts to the noise as well as to the real changes in the system. A low-pass filter (LPF) reduces the high-frequency noise while keeping the slower changes that represent the real behaviour of the system, as illustrated in Figure 4. The controller then works with physically meaningful information.

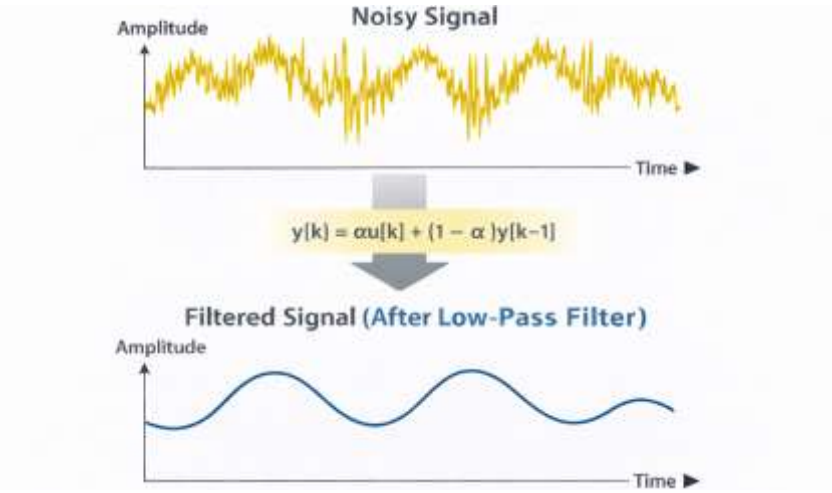


Figure 4: *Effect of a Low-Pass Filter (LPF) on a Noisy Signal*

Historically, LPFs were built as analogue resistor–capacitor (RC) circuits, described by the ordinary differential equation (ODE):

$$\tau \frac{dy(t)}{dt} + y(t) = u(t) \tag{4}$$

where $u(t)$ is the input (noisy) signal, $y(t)$ is the output (filtered) signal and $\tau = RC$ is the time constant. The corresponding transfer function is:

$$H(s) = \frac{1}{\tau s + 1} \tag{5}$$

A larger time constant gives more smoothing, but also a slower response to genuine changes. In a microcontroller, the same behaviour is reproduced in software using a discrete-time version of the filter, which updates once every sampling interval, T_s:

$$H(z) = \frac{\alpha}{1 - (1 - \alpha)z^{-1}} \tag{6}$$

where α is a number between 0 and 1 that depends on τ and T_s. A common choice is $\alpha = T_s/(\tau + T_s)$, so a longer time constant gives a smaller α and more smoothing. Written as a difference equation, which can be implemented directly in code, the filter is:

$$y[k] = \alpha\, u[k] + (1 - \alpha)\, y[k-1] \tag{7}$$

where $u[k]$ is the newest measurement, $y[k]$ is the new filtered value and $y[k-1]$ is the previous filtered value. Each new output is a weighted average of the newest measurement and the previous output. The filter needs only one stored value, $y[k-1]$, which makes it ideal for a microcontroller.

In practice, a digital LPF is usually developed in three stages:

1. design the filter in continuous time and check it in simulation, for example in MATLAB and Simulink
2. discretise the continuous-time model, which introduces the sampling interval T_s
3. implement the resulting difference equation on the microcontroller, and compare the measured data with the simulation

These stages, and the mathematics of discretisation, are covered in the Engineer stage of ACE-CORE.

6.0 Summary

This guide moved from the block diagram to the practical implementation of measurement using a microcontroller. It showed how digital and analogue signals enter the Arduino Uno, and how they are converted, scaled and filtered to become meaningful measurements. The key points are:

- A voltage or pulse at a pin is not yet a measurement; it becomes one only after conversion, scaling and interpretation.
- Digital sensors provide either logic levels or digitally encoded signals, such as the timed echo pulse of an ultrasonic sensor.
- The resolution of the 10-bit ADC (about 4.88 mV with a 5 V reference) and quantisation limit how precisely a quantity can be measured.
- The datasheet provides the relationship between the physical quantity and the electrical signal, and the limits within which the sensor must be used.
- Measurement noise is unavoidable for many analogue sensors, and filtering ensures that the controller responds to the real behaviour of the system.
- An analogue RC filter can be replaced by a discrete-time difference equation running on the microcontroller.

Reliable measurement is the foundation on which closed-loop control is built, and it is integrated into complete control systems in the guides that follow.

Recommended Citation

If you have used this guide in your teaching, learning or research, please cite it using the reference below, adapting the format as required by your preferred referencing style.

Pickering, J. E. (2026). Microcontroller-Based Measurement for Control Systems. ACE-Lab. Teaching Notes: ACE-Lab Learning Guide, Version 1.0. https://zenodo.org/records/23337635