

ECE 215/S – Principles of Electronic Cyber Warfare

Block 2: Signal Processing

Lesson 22: ADC / Arduino Lab

Department of Electrical and Computer Engineering
United States Air Force Academy

Fall 2026

Lab Goal and Deliverable

Lesson 21 gave you the map $v_{\text{out}} = Kv_{\text{in}} + B$; today you point it at a sensor that is **not linear**. Use an **Arduino Due** and an infrared distance sensor to build a voltage-to-distance conversion — and then prove how accurate it is.

- Work in **pairs**. Your partner must **not** be from your Project 1 group
- **Demo your working sensor** to your instructor, and answer the **Gradescope questions individually** — together worth **1 project point**
- You will build the whole chain yourself: transducer, ADC, calibration, and the code that interprets the result

Target accuracy: within ± 0.5 cm anywhere from 2 to 15 cm.

What We Have Learned, and How We Will Use It

Concept	Objective	Where it shows up today
Frequency domain and bandwidth	2.1	Why a reading every 2 s is enough
Filters	2.2, 2.4	Why no anti-alias filter is needed
ADC sampling and resolution	2.5	The Due's converter, and ΔV
Signal conditioning	2.6	The K and B you fit, and the range used

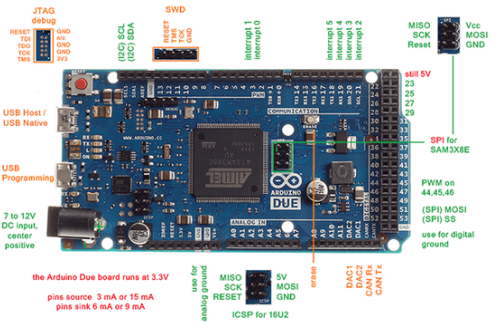
- **Sampling and filtering (2.1, 2.2, 2.4).** The distance holds still while you read it, so one sample every 2 s is plenty — Nyquist binds when the *signal* changes. The reading-to-reading scatter you average out over two or three prints is measurement jitter, not signal, which is why this sensor wires straight to `A0`
- **Signal conditioning (2.6).** The sensor puts 0–2.46 V into a 0–3.3 V converter — **763 of the 1024 levels**, Lesson 21's Pitfall 3 in the wild. And $v_{out} = Kv_{in} + B$ is the piece you will use most today: here it runs in *software*, behind the converter

Part 1: Embedded Systems — MCU, FPGA, ASIC

	Microcontroller	FPGA	ASIC
Flexibility	Reprogram software	Reconfigure hardware	Fixed at fabrication
Processing	Sequential	Parallel	Parallel, optimized
Dev. cost	Low	Moderate	Very high
Unit cost	Low	High	Low at volume
Time to market	Fast	Fast	Very slow
Power	Very efficient	Moderate	Most efficient
Security	Software, variable	Implementation-dep.	Highest, hardwired

- We use a **microcontroller** — cheap, fast to develop, power-efficient
- **Security is the row worth noting.** An ASIC provides a hardware root of trust; an FPGA's reconfigurability is both a feature and an attack surface

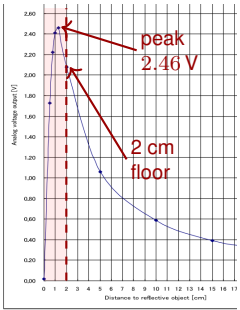
The Arduino Due



- The microcontroller is the single large chip in the middle — everything else on the board exists to make wiring and programming convenient
- Analog inputs feed an on-board **ADC**, 10-bit over 0–3.3 V by default
- Programmed over USB from the **Arduino IDE**

Use the port labeled **USB Programming** — the middle of the three connectors on that edge, between the native-USB jack and the barrel jack. It both powers and programs the board; the other mini-USB jack cannot program it.

The Distance Sensor

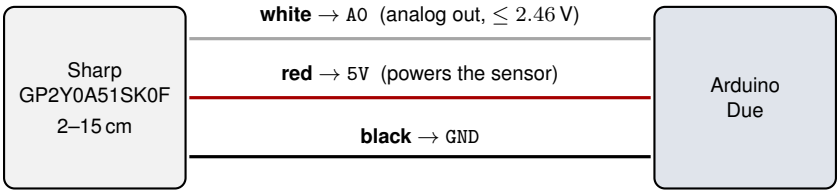


Sharp GP2Y0A51SK0F: volts out vs distance

- Measures obstacles from **2 cm to 15 cm**
- Outputs an **analog voltage** — a transducer, exactly as in Objective 2.6
- **The transfer curve is not linear.** It falls fast near 2 cm and flattens out toward 15 cm
- It peaks at $\mathbf{2.46\,V}$ near 1.2 cm. Below $\sim$1.5 cm it **turns around**: one voltage, two distances. That is why 2 cm is the floor

The nonlinearity is the interesting part. A straight-line fit will not meet spec.

Part 2: Wiring — Powered at 5 V, Read at 3.3 V



The sensor **runs** on 5 V. Its **output** never exceeds about 2.46 V, which is safely under the Due's 3.3 V analog limit — that is the only reason this is safe.

Never put the red 5 V lead on an analog pin. Due pins are not 5 V tolerant.

Part 3: Run It, and Find ΔV

1. Install the Arduino IDE, open the starter sketch, and select **your board and COM port** in the dropdown
2. Upload, then open the **Serial Monitor** at **9600 baud**. A reading prints every **2 seconds**: a raw count, a voltage, and `d = ?`
3. Derive the Due's ADC parameters from Objective 2.5 — **write ΔV down, Gradescope asks for it**:

$$\text{levels} = 2^b = 2^{10} = 1024 \qquad \Delta V = \frac{V_{\max} - V_{\min}}{2^b} = \frac{3.3\,\text{V} - 0\,\text{V}}{1024} = 3.22\ \text{mV}$$

The raw count *is* the $QL = \lfloor (V_{in} - V_{\min})/\Delta V \rfloor$ from the reading. The sketch runs it backward.

Part 4: Build the Calibration Table

The sensor gives you volts. You want centimeters. Nothing knows how to get from one to the other until you measure it.

1. Zero the ruler at the front face of the sensor and stand a **flat target** on it
2. Record the voltage at **at least 8 fit distances** spanning 2–15 cm — e.g. 2, 3, 4, 5, 7, 9, 11, 13, 15
3. Record **at least 3 check distances** you will *not* use in the fit — e.g. 2.5, 6, 8, 12
4. Wait for two or three prints at each distance and take a typical value. Keep the target **square** to the sensor

Decide which points are check points *before* you fit. That is what makes them evidence.

Part 5: Linearize First, Then Fit K and B

- A straight line through raw V vs d misses by **centimeters** — the wrong *shape*, not a tuning problem
- Add a $\mathbf{1/V}$ column. Plot d against $1/V$: **that** is nearly a straight line

$$d = K \cdot \tfrac{1}{V} + B$$

- **Fit it as in Lesson 21**: two equations from your **2 cm** and **15 cm** rows, subtract to eliminate B, solve for K. An **Excel trendline** fits the same line through all nine fit points at once and averages out your noise — expect it to beat the two-point fit

Then plug the constants into the sketch:

```
// ====== PLUG IN (Part 5) ======
const float CAL_K = 0.0;
const float CAL_B = 0.0;
// ==============================
```

. . . and finish `distanceCm()` so it returns `CAL_K / volts + CAL_B`.

`d = ?` becomes a distance as soon as `CAL_K` is no longer zero.

Same algebra, different job. Lesson 21's K was a dimensionless gain *ahead* of the ADC; here K carries **cm·V** and B **cm**, converting *behind* it. Sanity: $K \approx 5$–8 cm·V, B between -2 and 0 cm.

Parts 6–7: Validate, Then Demo

A fit that matches the data you built it from is close to a tautology. **How would you test your conversion?** On distances it has never seen.

1. Put the target at each **check** distance and record what the sketch reports, and the error
2. Spec: $|\text{error}| \leq 0.5$ cm at **every** check point. Record your **maximum** error
3. All errors in the *same* direction means a biased fit. Scattered errors mean measurement noise — re-take the worst reading
4. **Demo** the working sensor to your instructor, then answer Gradescope **individually**

Never tune K and B to hit the check points.
That turns your evidence back into data.

Authorized Resources

- The provided Arduino kit, your laptop, and a **ruler**
- The Arduino website, the sensor specification sheet, and the starter sketch
- **Excel, MATLAB, or similar** for the curve fit
- **GenAI is authorized to Level 3 only** — as a feedback tool on work you generated yourself
  - This is **stricter** than the course default
- You may **not** use materials from previous semesters

Document any collaboration or GenAI use in your submission.

Submission Checklist

Before you leave:

- ☐ Sensor wired white→`A0`, red→`5V`, black→`GND`, checked before power
- ☐ Starter sketch running and streaming raw counts and volts at 9600 baud
- ☐ ΔV for the Due's ADC computed from 2^b and the 0–3.3 V range
- ☐ Calibration table filled in — **at least 8 fit distances** across 2–15 cm and **at least 3 check distances** held out
- ☐ d plotted against $1/V$ and a line fit to it
- ☐ K and B recorded, sanity-checked, and typed into the sketch's PLUG-IN block
- ☐ `distanceCm()` completed and the Serial Monitor printing centimeters
- ☐ Every check distance validated, errors recorded, max $|\text{error}| \leq 0.5$ cm
- ☐ Working sensor demonstrated to your instructor
- ☐ Gradescope questions answered **individually**
- ☐ Documentation statement — collaboration and any GenAI use
- ☐ Both partners leave with a copy of the calibrated sketch

Wrap-Up

- You built the full chain: **transducer** → **ADC** → **calibration** → **interpretation**
- The habit worth keeping: **hold points out of the fit**, then let them tell you whether the fit is any good
- Your `distanceCm()` becomes the **range channel** of the fire-control system you build in **Project 2** — keep the calibrated sketch

A biased seeker declares in-range early.
Your calibration is Project 2's accuracy.

- **Lesson 23 — Objective 2.7: Digital Logic.** What the computer does with those bits once it has them
- **Quiz 6** (Objectives 2.5–2.6) is due at Lesson 24