Voltage, A2D, Piece Wise Linear, Noise
Topics

- What form are real-world signals?
- How can a computer read an analog signal?
- How can we approximate functions?
Signals in the “Real World”: Voltage
Real world analog signals are often changes in voltage:

- Ex: Microphone encodes sound into voltage levels

Audio: Zoomed in

Audio: Zoomed out
Voltage Ranges

These are all DC voltage (Direct Current)

Out of the wall comes AC Voltage (Alternating Current)

5.0V: Some circuits (Arduino)
3.3V: Many circuits (BeagleBone)
1.8V: BeagleBone A2D ref V
0V: Ground
Many electronics components run on, manage, and work with voltages.

**Voltage Regulator:**
Converts input voltage to stable output voltage.

May fluctuate a little

4 to 8V
3.3V

Gnd Input
Gnd Output

**Potentiometer:**
Turning the knob adjusts the output voltage on $V_{out}$.

$V_{out}$

Gnd

**Light Sensor:**
The more light, the lower the voltage on $V_{out}$.

Gnd
1.8V
$V_{out}$
Reading a Voltage

- How can we read a signal into the computer?
  - Real world is analog voltages; computer are digital.
  - We need an analog to digital converter (ADC)
    - Also called an A2D (Analog “to” Digital)
- BeagleBone has a 12 bit A2D:
  - It reads a voltage and gives a number between 0 and $2^{12}-1 (=4095)$
  - It can sample voltages between 0V and 1.8V
    - It is easily damaged by higher voltages!
Quantization & Sampling

- **Quantization:**
  Since it has 4096 samples over 1.8V
  - Resolution of a single bit is:
    \[
    \frac{1.8V}{4096} = 0.00044V = 0.44 \text{ mV}
    \]
  This is pretty good for most applications!

- **Sample Rate:**
  How fast the A2D can read samples
  - Need 44100 Hz (44.1kHz) for CD audio
  - BeagleBone can sample at 1.6MHz (1600kHz)
  - Some applications (reading a POT for volume) may need low sample rates (~10Hz)
BBB A2D Demo for POT

- A2D is enabled by default
  - Done for us by UBoot via /boot/uEnv.txt

- Change to sys file system folder:
  ```
  # cd /sys/bus/iio/devices/iio\:device0
  ```

- Read voltage 0 (for POT):
  ```
  # cat in_voltage0_raw
  ```
Approximating Functions:

Piece Wise Linear
Function Approximations

- Real world functions can be hard to approximate.
  - Some approximations are computationally expensive (high-order polynomials, cubic-spline, ..)
  - Piecewise Linear (PWL)
    Approximate a function with a series of lines.

As you discharge a battery, its voltage drops. (DoD is Depth of Discharge)
Piece Wise Linear

- Pick good points on the function $f(x)$ to capture its shape
  - can be evenly spaced, or
  - can be specially selected points
- Between adjacent points, draw a straight line.
- The approximation $f'(x)$ is the straight lines.
Computing Piecewise Linear

- Given an input value $s$, use points on either side
- Compute $f'(s)$ by solving the point on the line

$$f'(s) = \left( \frac{s-a}{b-a} \right) \cdot (n-m) + m$$
Understanding Piecewise Linear

\[ f'(s) = \left( \frac{s-a}{b-a} \right) \cdot (n-m) + m \]
Piecewise Linear Details

- **Some extra notes:**
  - If a reading is < min or > max data point, clip it to min & max.
  - Enter the points into a program as two arrays:

```c
#define PIECEWISE_NUM_POINTS 11
const float PIECEWISE_DoD[] = { 0.0, 0.1, ... 0.8, 0.9, 1.0};
const float PIECEWISE_V[]  = {12.6, 12.3, ... 11.2, 11.1, 10};
```

- Make sure to use the correct data types for your calculation (possibly floating point).
- Watch for array out of bounds!
Noise

- **Real world data is often 'noisy'**
  - each sample has noise causing it to differ from the correct real-world value.

\[ \text{A2D Sample} = (\text{precise real-world value}) + (\text{noise}) \]
Problem with Noise

- A noisy signal’s fluctuations may be:
  - changes in the real signal
  - noise
- Ex: Turn off phone when battery is empty (3V)

```c
void powerDownIfBatteryDead() {
  if (batteryVoltage < 3.0) {
    powerDown();
  }
}
```

- What happens when noise spike gives you 2.99V reading when battery actually at 3.10V?
Tolerating Noise:
N Samples Past Threshold

• An idea to tolerate some noise:..

• Ex: Power off if 5 consecutive samples are less than 3V:

```c
double batteryVHistory[5];
void powerDownIfBatteryDead() {
    for (int i = 0; i < 5; i++) {
        if (batteryVHistory[i] >= 3.0) {
            return;
        }
    }
    powerDown();
}
```
Tolerating Noise: Hysteresis

• State machine should be stable:
  – Problematic Example:
    Battery-saver when State of Charge < 30%
    ```
    _Bool inLowPower = false;
    void manageLowPowerState() {
      if (batterySoC < 30) {
        inLowPower = true;
      } else {
        inLowPower = false;
      }
    }
    ```
  
• Problem?
  ```
  ..
  ```
Hysteresis Solution

- A solution:

```c
_Bool inLowPower = false;
void manageLowPowerState() {
  // Enter
  if (batterySoC < 30) {
    inLowPower = true;
  }
  // Exit (5% SoC Hysteresis)
  if (batterySoC > 35) {
    inLowPower = false;
  }
}
```
 Noise Filters
Simple Moving Average

• Rather than tolerating noise,..
• Maintain buffer of *previous* N samples
  ```c
  double batteryVFiltered = 0;
  double samples[10];
  int nextIdx = 0;
  void getNewBatetryV() {
    // Sample
    samples[nextIdx] = readA2DVoltage();
    nextIdx = (nextIdx + 1) % 10;

    // Filter
    batteryVFiltered = average(samples, 10);
    //batteryVFiltered = median(samples, 10);
  }

  double average(double *data, int numValues) {...}
  ```
• Note: Must also handle non-full buffer.
Noise Example

Signal and Noise

Signal Value [V]

Time

f-pure(x)
Noise
f(x)
Simple Moving Average Effectiveness

Why is N=10 plots shifted?

Is averaging or median filtering better? When might median be clear winner?
Exponential Smoothing

- Simple moving average equally weights all samples,

**Exponential Smoothing Details**
- Let $s_n$ be the Nth sample from the A2D
  - Let $v_n$ be the Nth filtered value
  - Let $a$ be a weighting value between 0 and 1

**Smoothed Data Points ($v_n$)**

$$v_0 = s_0$$
$$v_n = a \cdot s_n + (1 - a) \cdot v_{(n-1)}$$
Exponential Smoothing Intuition

- $s_n$ is the Nth sample from the A2D
  - $v_n$ is the Nth filtered value
  - $a$ is a weighting value between 0 and 1

- **Smoothed Data Points ($v_n$)**
  - $v_0 = s_0$
  - $v_n = a \cdot s_n + (1 - a) \cdot v_{(n-1)}$

- **Intuition**
  - $a = 1$: 100% weight on instantaneous ‘now’ sample (filtering disabled)
  - $a = 0.1$: Very heavy weight on old data, not much on new data (average over very long time frame)
Exponential Smoothing Effectiveness

Exponential Smoothing

Signal Voltage [V]

Time

f(x)
a=0.5
a=0.3
a=0.2
Summary

- Many sensor generate analog voltage signals.
  - Be careful that signal is in correct voltage range!

- BBB can sample voltages between 0 and 1.8V
  - 12-bit A2D: digital values between 0 and 4095

- Piecewise Linear approximates functions
  - Given a reading (on the X axis),
    use the selected points and straight lines to
    approximate desired value (on the Y axis)

- Noise adds errors to samples
  - Tolerate noise with hysteresis and filter thresholds
  - Filter with simple moving average or exponential smoothing.