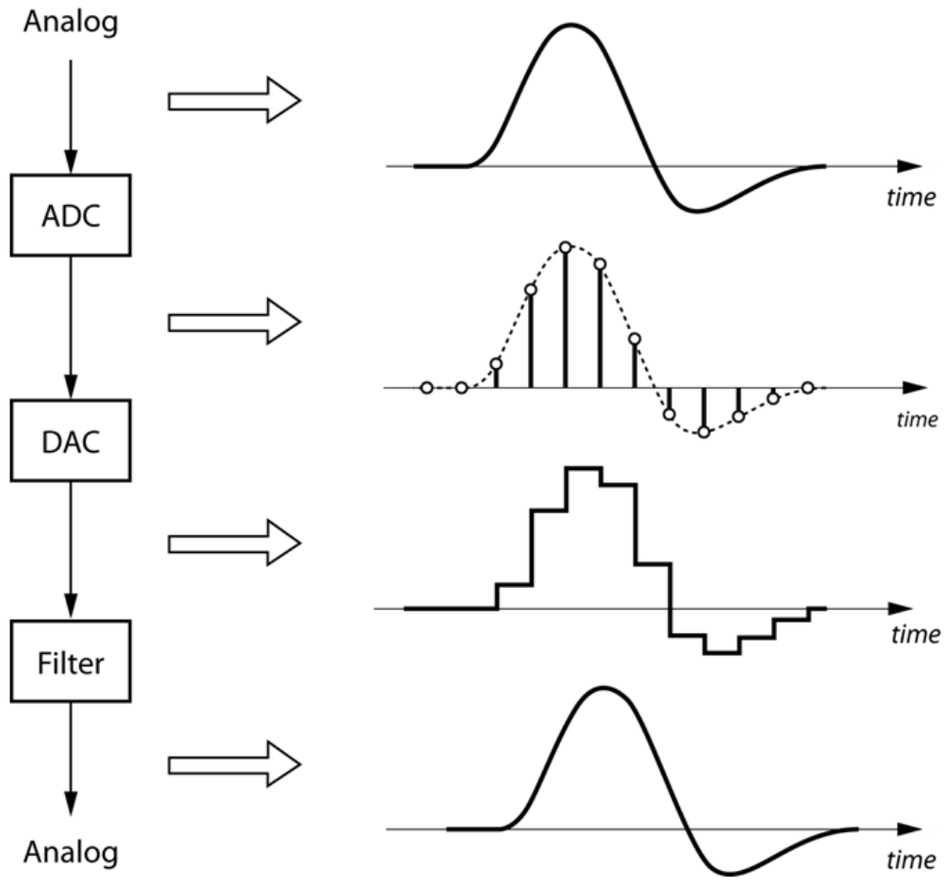
EC-ENG 231 (Spring 2024)

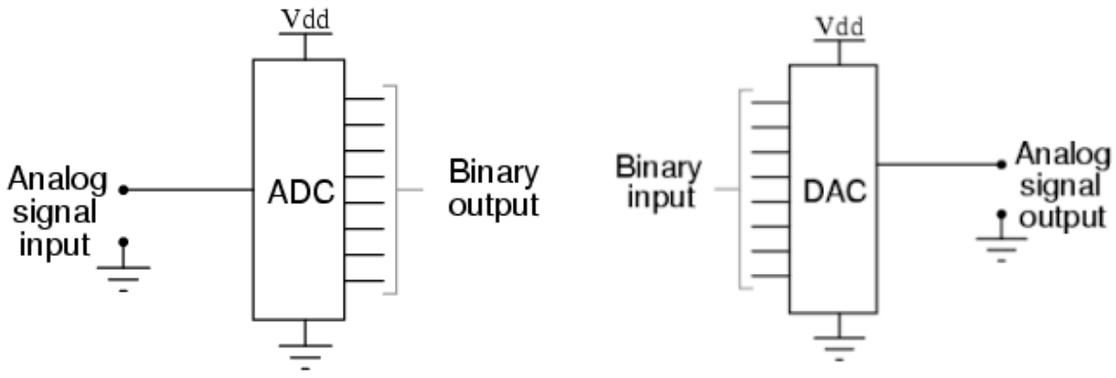
Analog I/O
Sensing

Fatima Anwar
fanwar@umass.edu
UMass Amherst

Analog I/O

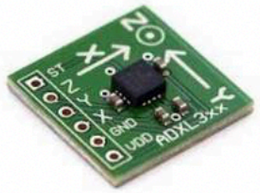
- Input: Analog to Digital Converter (ADC)
  - Periodic or asynchronous sampling
- Output: Digital to Analog Converter (DAC)
  - Periodic or asynchronous update
- Accompanied by analog circuitry
- **What are some of the limitations of analog I/O?**





Typical Sensing Pipeline

Input Device



Physical signal

Sampling

Cleansing / Processing

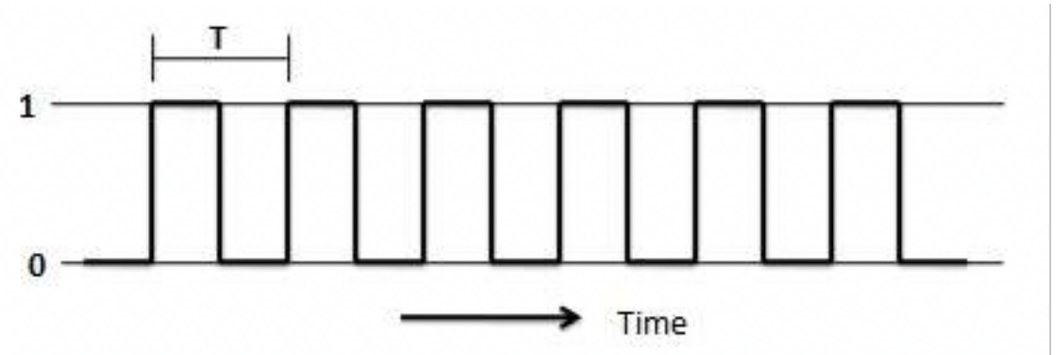
Analog (amplify, filter) → ADC (**sample**, digitize) → Digital (filter, **infer**) → Object, Gesture, Activity, Location, Speed, etc.

Inferencing

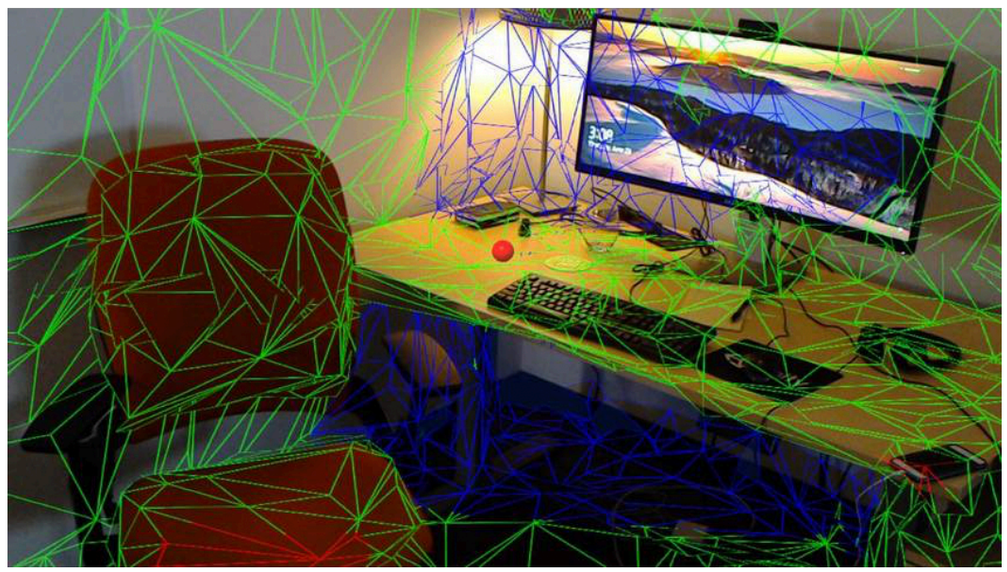
Courtesy: ECE M119 UCLA

Why Sample physical signal?

- We sense physical process that is continuous in both time and space
- Impossible to acquire physical data continuously
- Continuous signal cannot be stored digitally



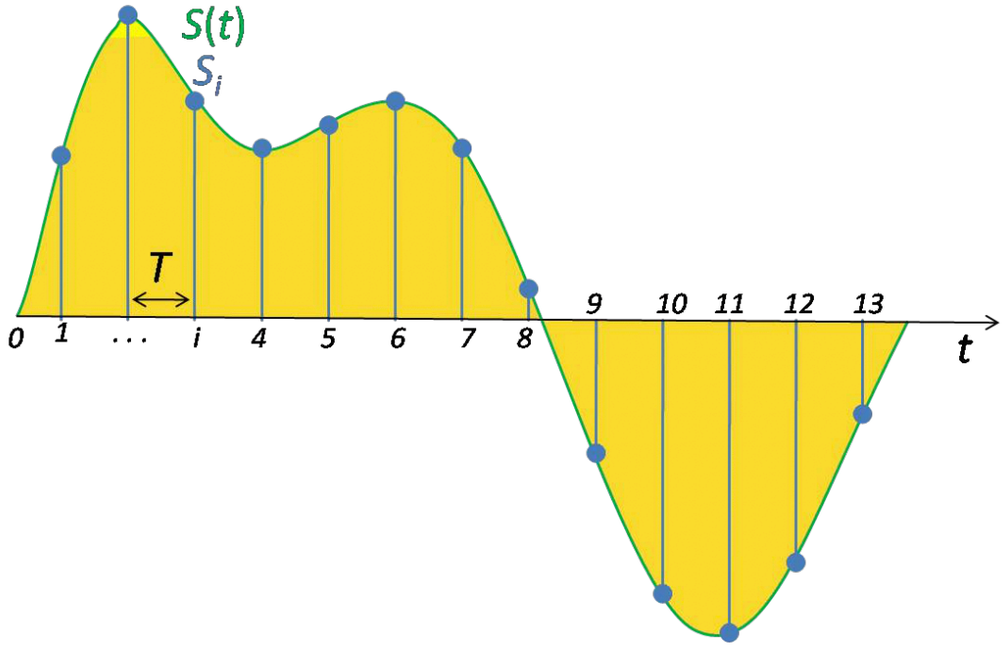
- Sampling is Instantaneous snapshot of continuous signal
  - Time - discrete clock
  - Space - bounded sensor resolution
- Sensors sample the world in a discrete manner

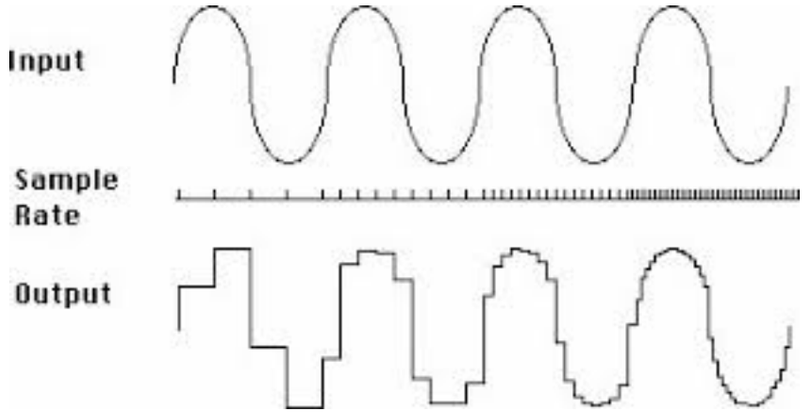


Courtesy: ECE M119 UCLA

Sampling Rate

- Discrete sampling loses information
  - Does high sampling rate solve problem?
  - But how high?
- High resolution ADCs consume power and storage
- Input signal properties govern sampling rate,
  - Higher sampling rate for audio
  - Lower rate for temperature, humidity



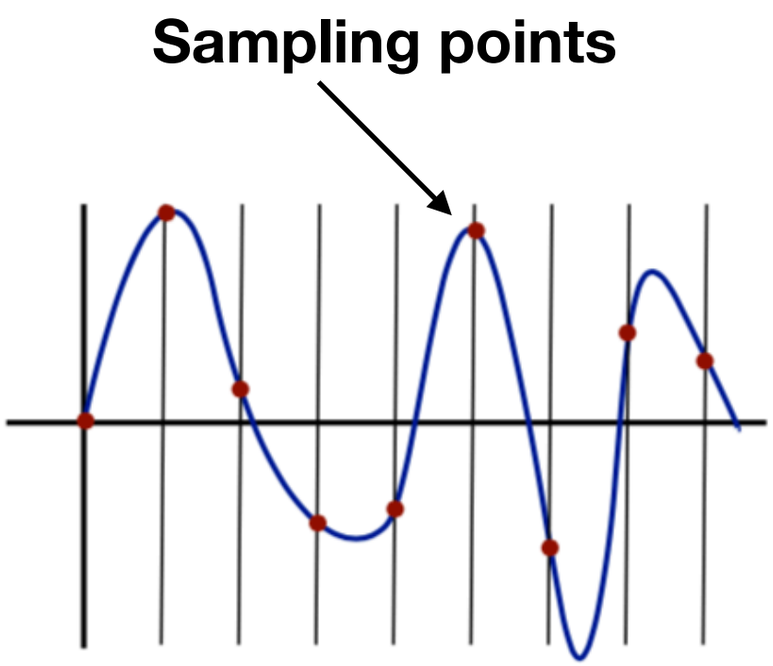


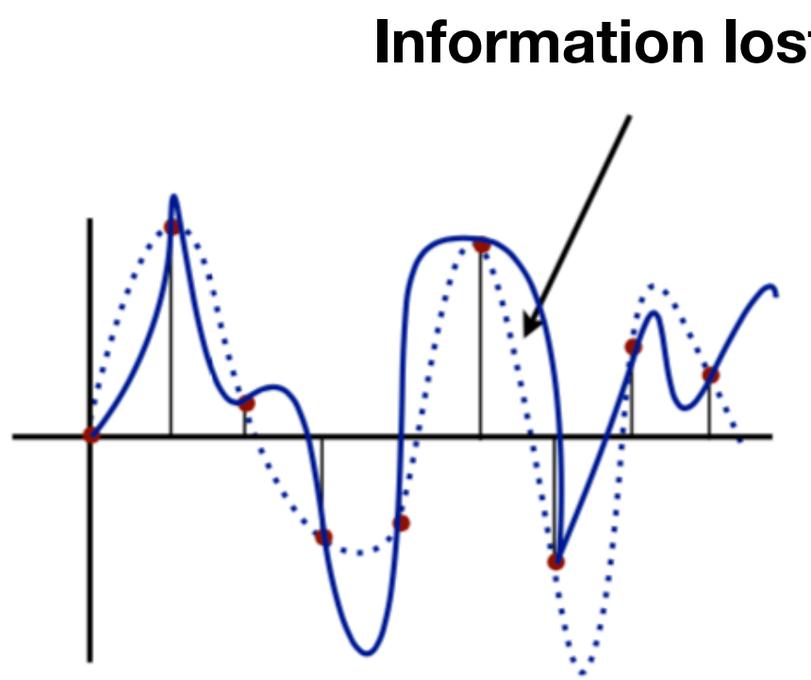
Courtesy: ECE M119 UCLA

How do we determine how high should the sampling rate be?

Nyquist-Shannon Sampling Theorem

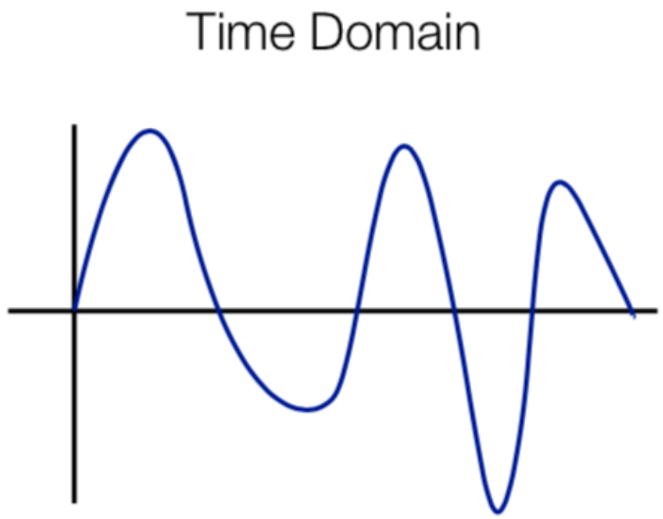
- If a periodic sampling rate is greater than twice the bandwidth of the signal, the original signal can be perfectly reconstructed using sinc interpolation
- Signals have to be band limited for recovery guarantees

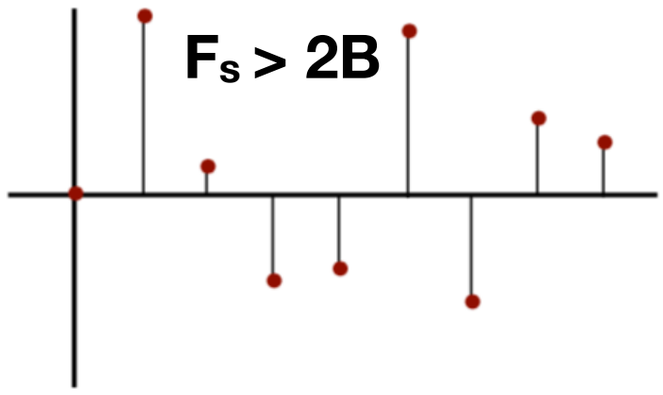


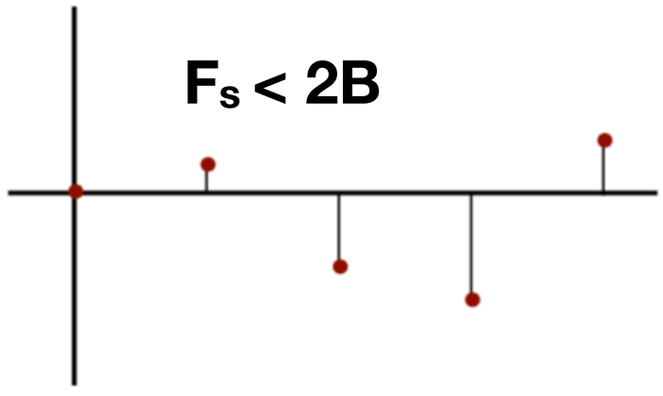


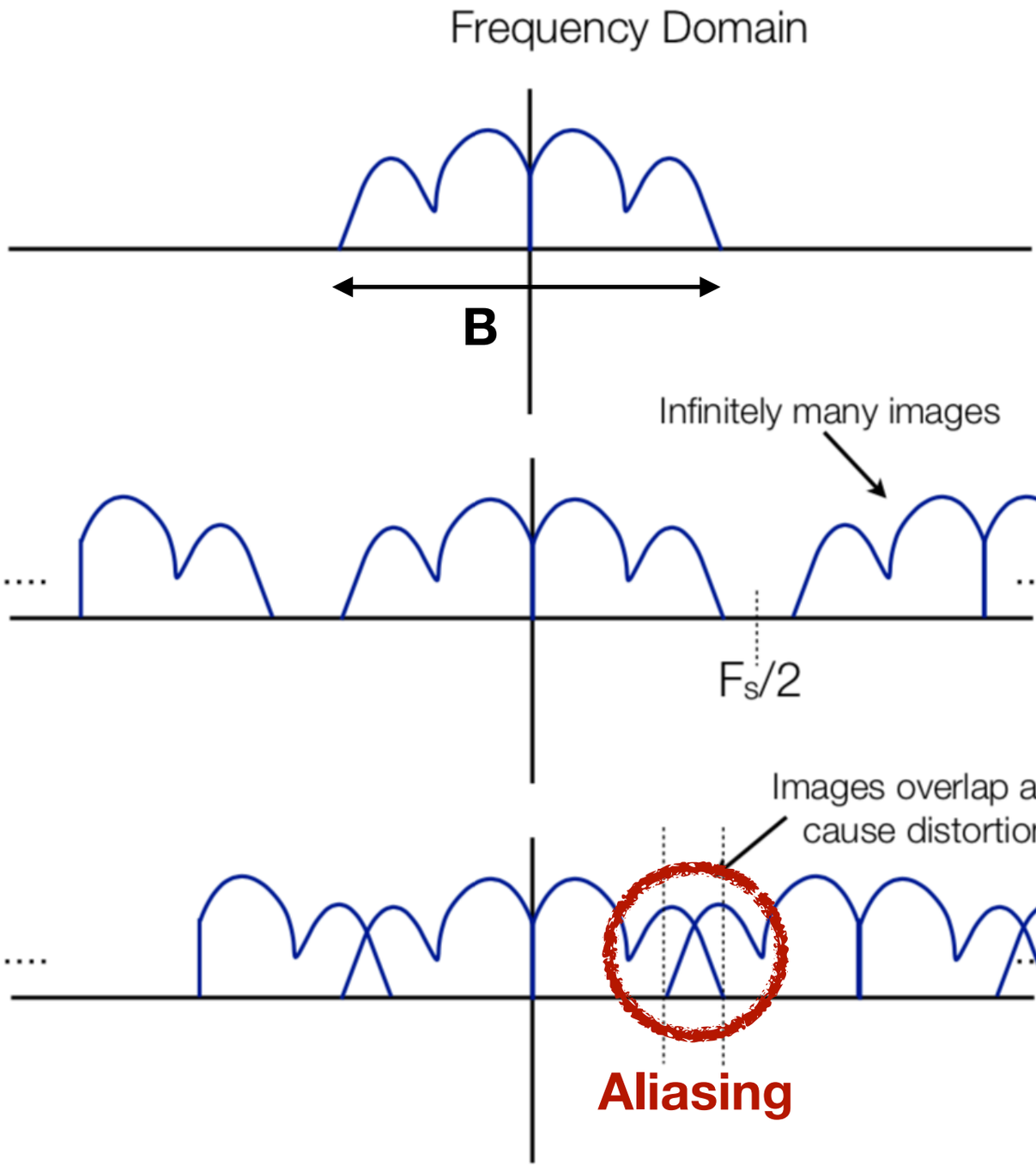
Courtesy: ECE M119 UCLA

Aliasing





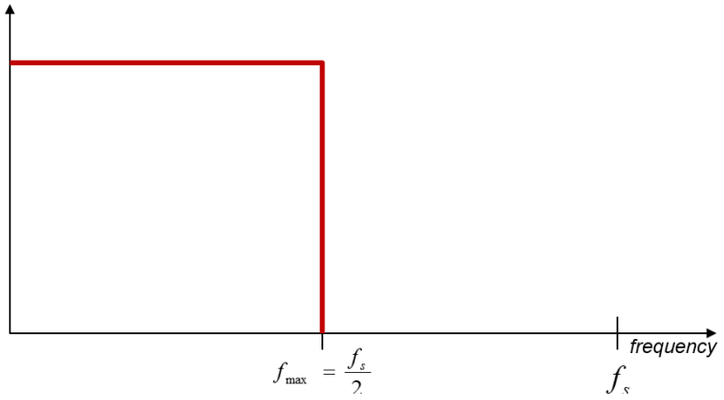


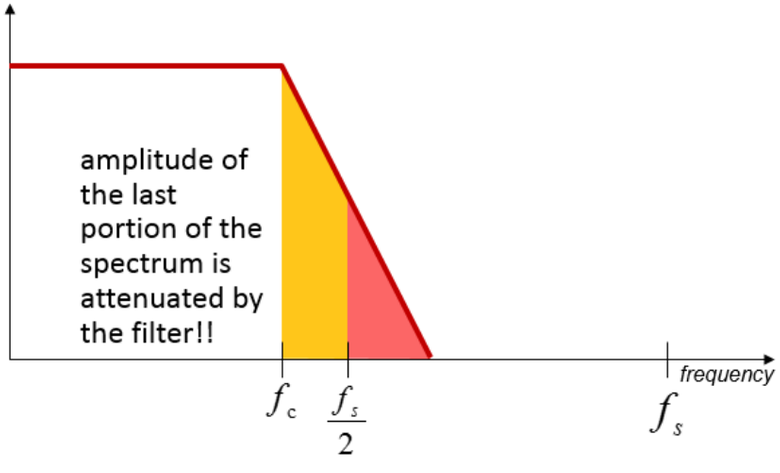


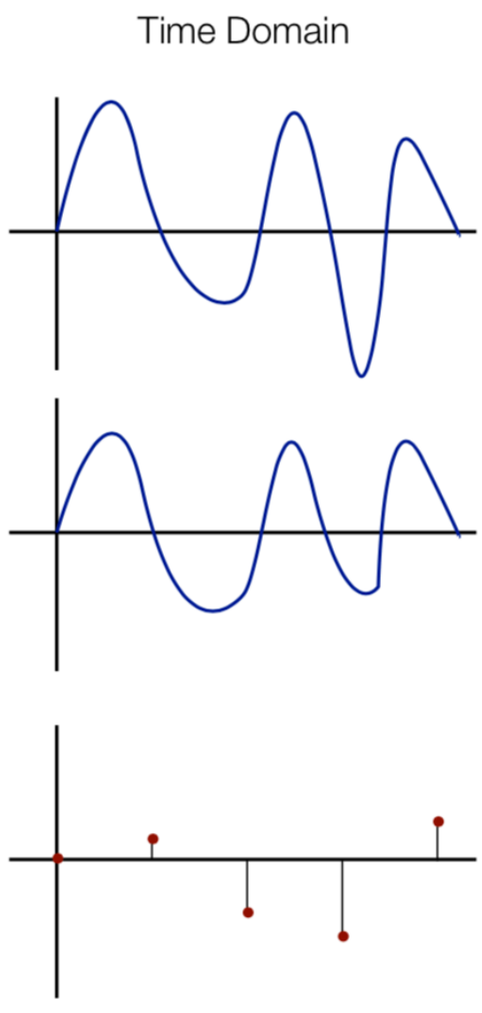
Courtesy: ECE M119 UCLA

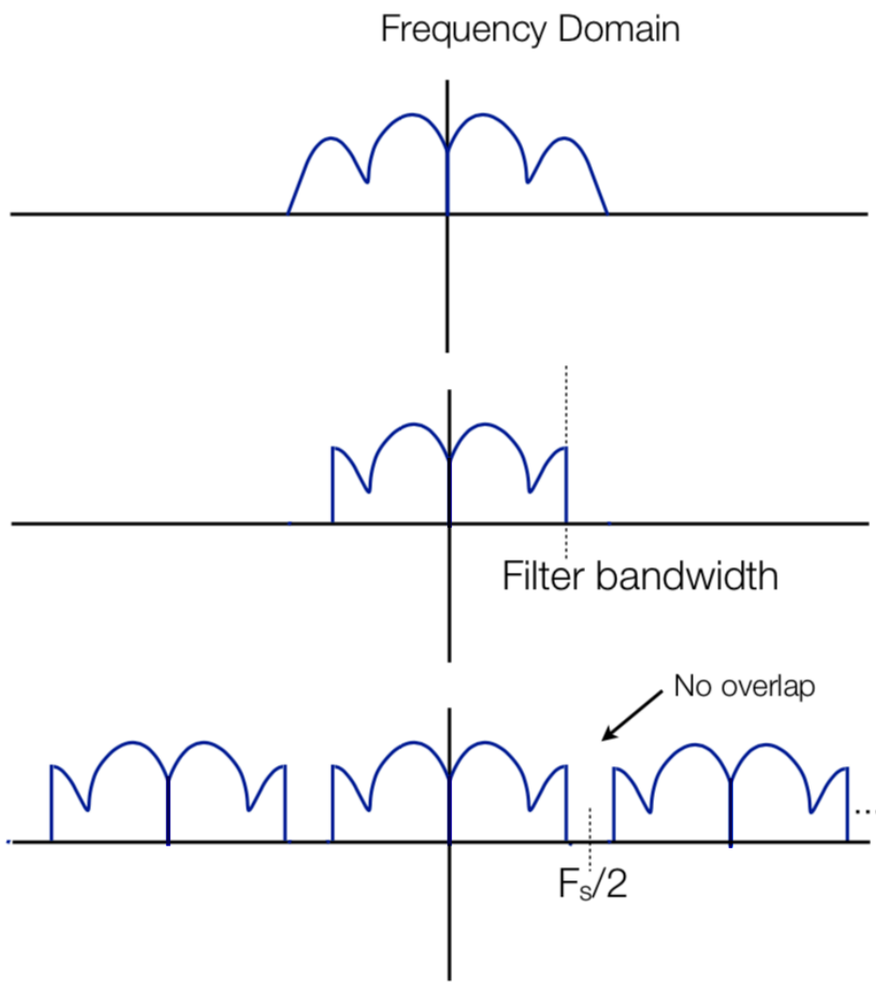
Anti-Alias Filter

- High frequency signals cause aliasing
- Truncate those high frequency signals using a low pass filter



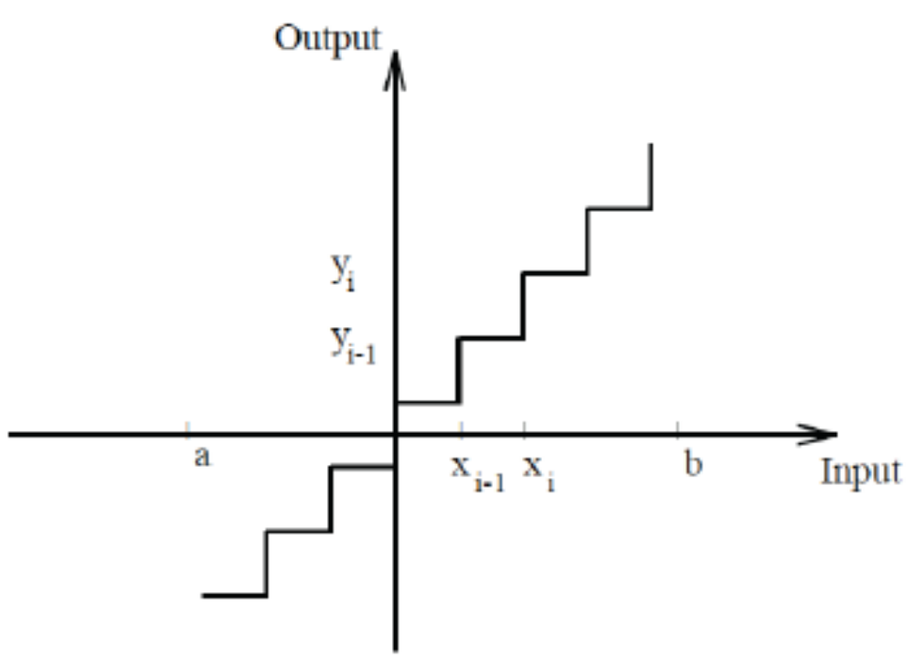




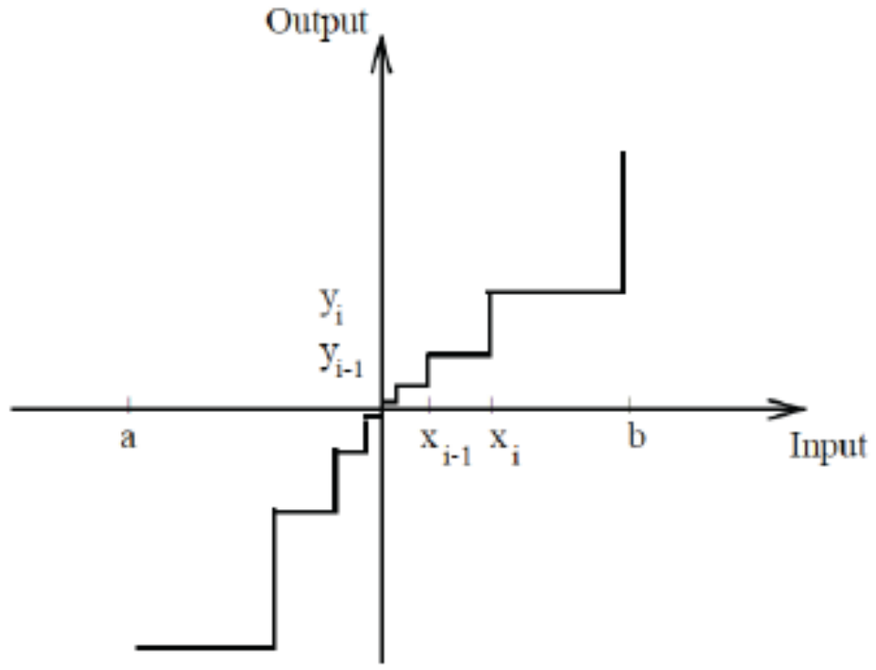


Quantization

- Discretize an amplitude value from continuous domain to a number of ‘bins’
- Most quantization use uniform bin sizes but some do it non-uniformly
  - Non uniform quantizers are called companders
  - Useful for acoustic applications since human hearing is known to be sensitive to log-scale rather than linear-scale loudness



Uniform

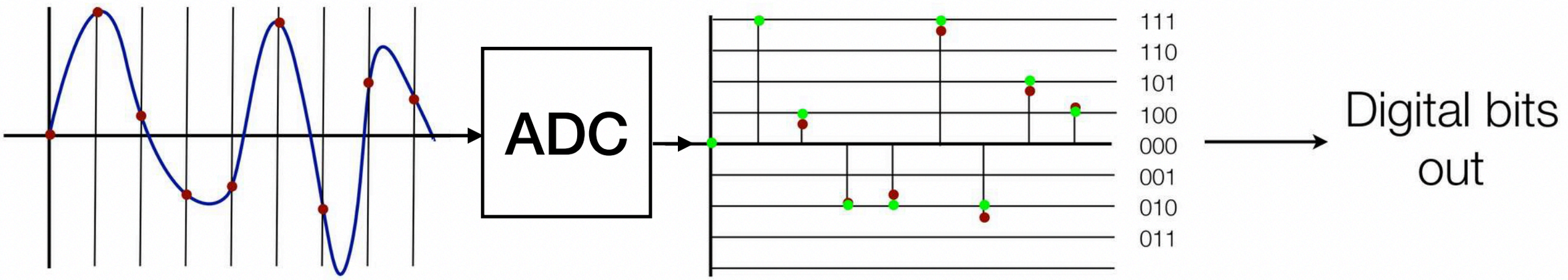


Non Uniform

Analog to Digital Conversion (ADC)

Analog Input:
Continuous Time
Continuous Amplitude Signal

Digital Output:
Discrete Time
Discrete Amplitude Signal



Notes on Sampling Resource Consumption

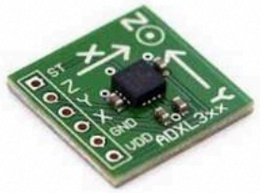
- Each sample acquired leads to processing burden for the embedded system
  - Some systems are energy constrained while others are computing constrained

- Increasing quantization resolution (eg. 8-bit vs. 16-bit) results in better digital representation but increases ADC power consumption and processing burden
  - Resolution higher than about 24 bits per sample is impractical in most cases because of measurement noise

- Some embedded systems are purely smart sensors, which digitize the analog signal and communicate them to a central controller for processing
  - Each additional sample and bit of resolution increases communication cost

Typical Sensing Pipeline

Input Device



Physical signal

Sampling

Cleansing / Processing

Analog (amplify, filter)

ADC (**sample**, digitize)

raw data

Digital (filter, **infer**)

Object
Gesture
Activity
Location
Speed
etc.

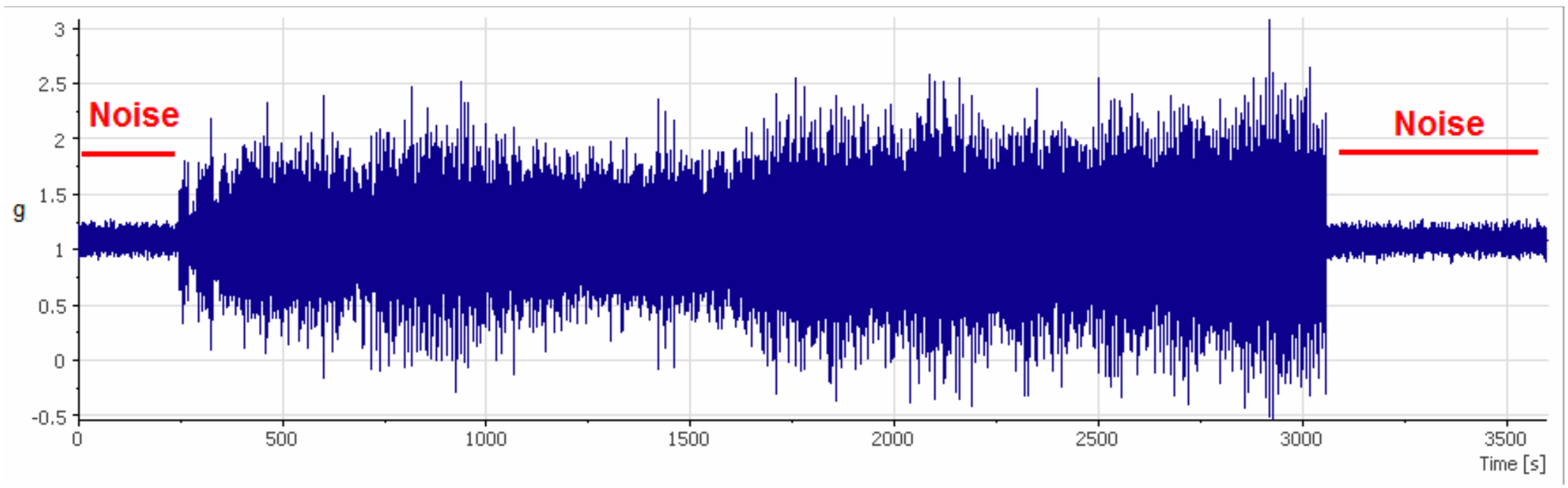
Inferencing

Courtesy: ECE M119 UCLA

Raw data → Sensor Data Cleaning → **Processed data**

Why do we need to clean sensor data?

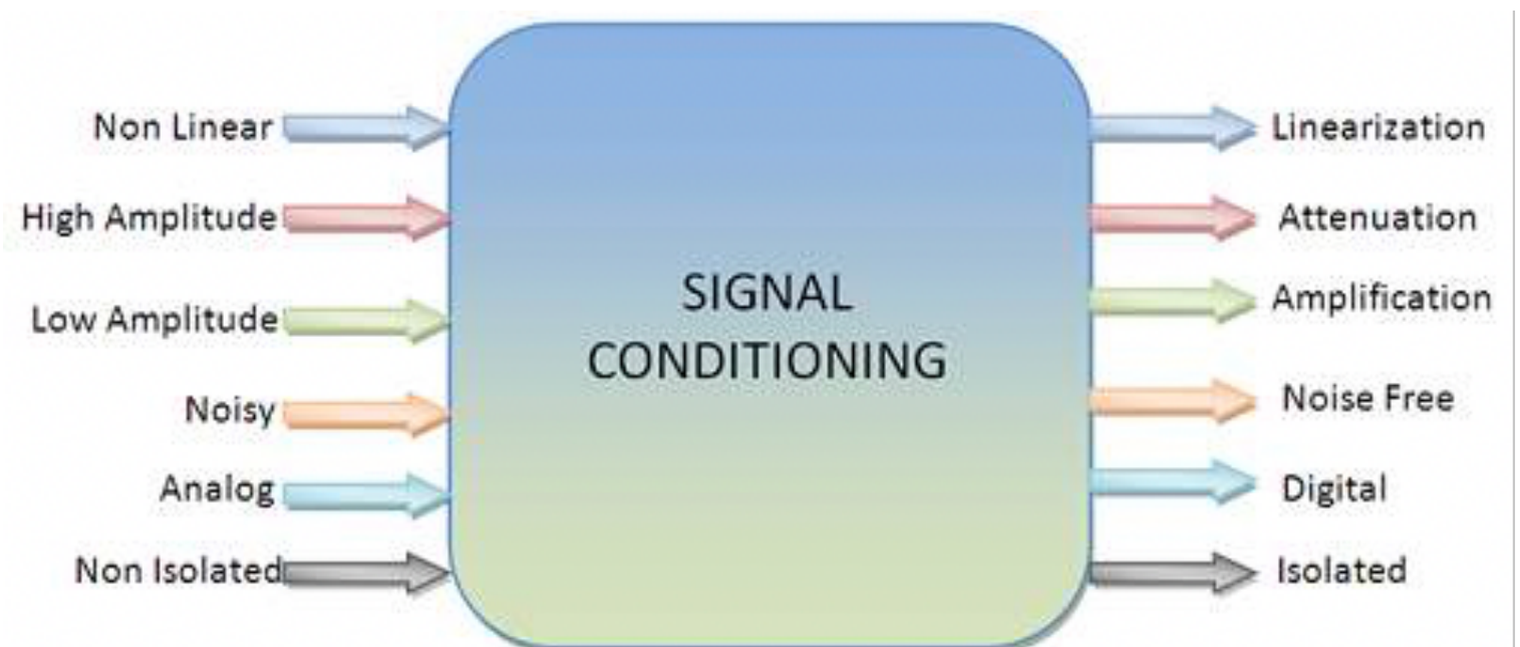
- Sensors read something you want to know, as well as some other things you don’t want to know
  - Example is an accelerometer that gives you orientation but also noise
- Sources of noise,
  - System errors from cheap and poorly calibrated accelerometers
  - Human/environment errors from jitter, temperature, vibrations etc.



Courtesy: ECE M119 UCLA

1. Signal Conditioning

- Prepare analog data before it gets digitized
  - Filtering: suppress certain signal components using filters e.g. low pass filter to eliminate sudden changes
  - Amplification/Attenuation: Scale signal amplitude e.g. thermocouple signals have very small voltage levels that must be amplified before digitizing it
  - Excitation: Sensors need external power to operate e.g. thermistor
  - Linearization: correlate output with measurement e.g. software correction applied to piezo pressure sensor

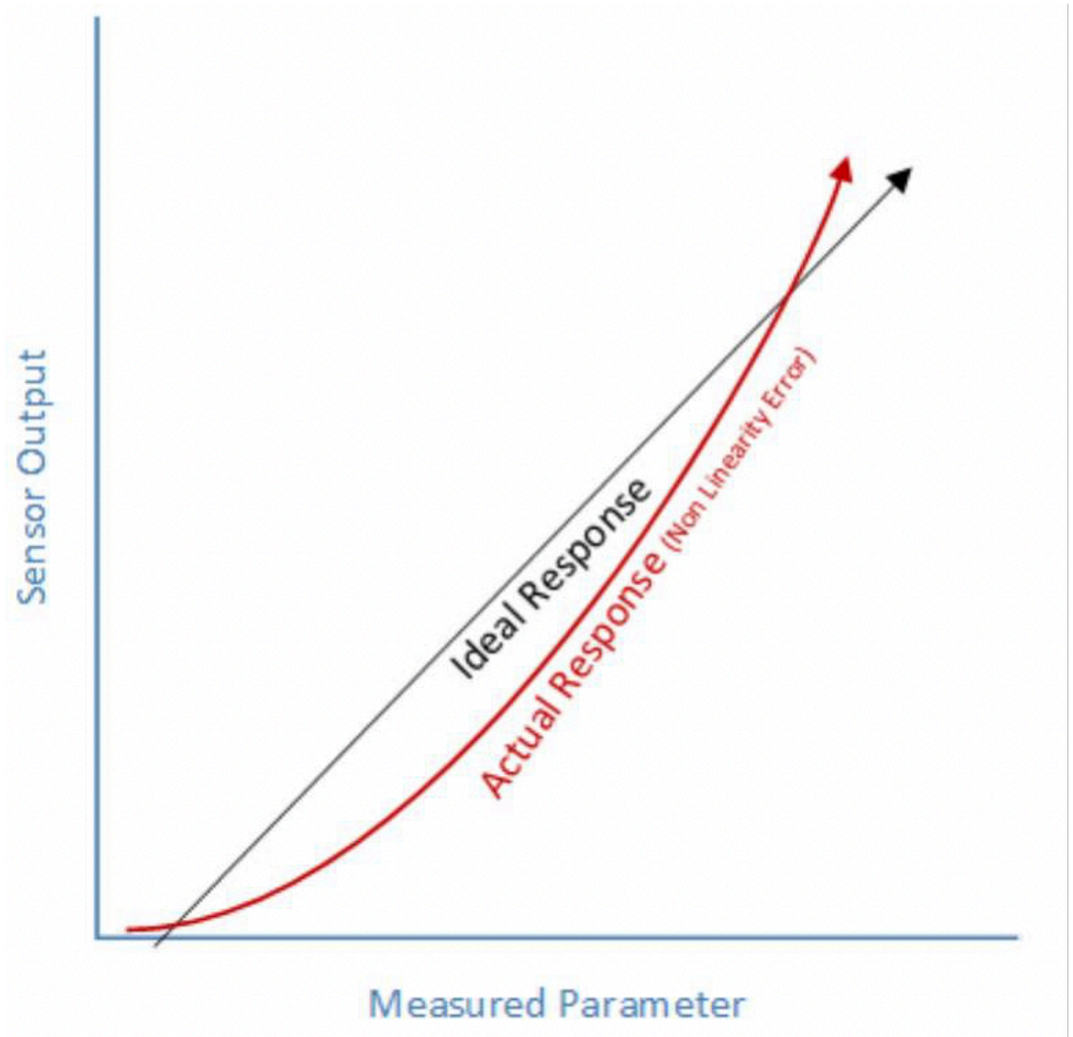


Courtesy: ECE M119 UCLA

A robot is positioned exactly 6” from the goal but its ultrasonic sensor reports 6.3”. How can we fix this problem?

2. Sensor Calibration

- Manufacturing variabilities
- Environmental dynamics such as heat, shock, humidity etc, and Aging
- Coupled components such as ADC variability
- For example,
  - Light sensors affected by spectral reflections and other optical phenomena
  - Inertial sensors almost always have “non-zero offset”
  - Temperature sensors subject to thermal gradients



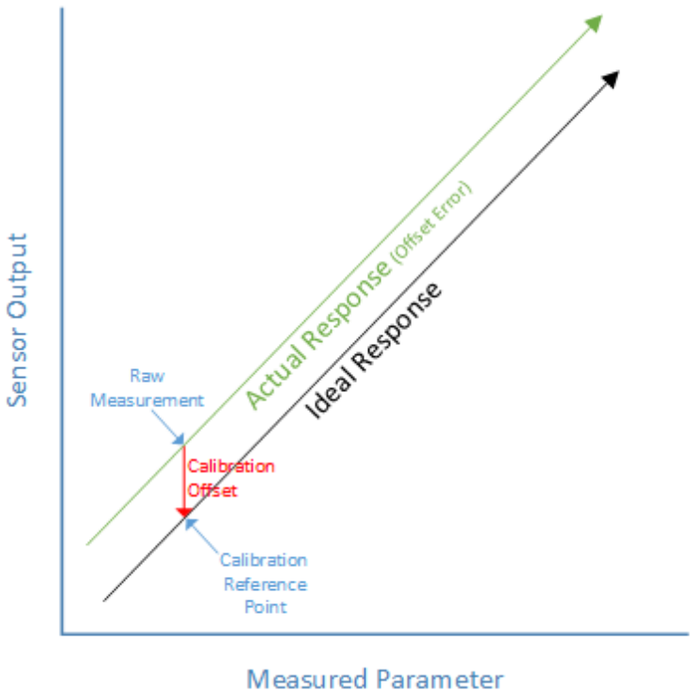
Courtesy: ECE M119 UCLA

2. Sensor Calibration

Calibration establishes correct mapping between a signal (light) and its sensed physical property (light intensity)

A robot needs to position itself exactly 6” from a goal but the ultrasonic sensor report 6.3”so there is an offset = -0.3”,

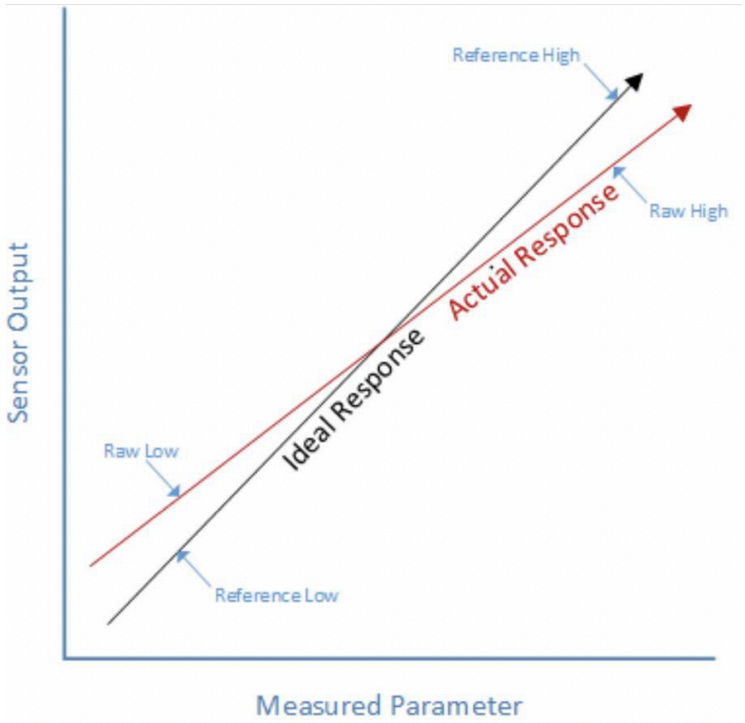
Correct = Raw - offset



One-point

Take into account both Offset and rotation error,

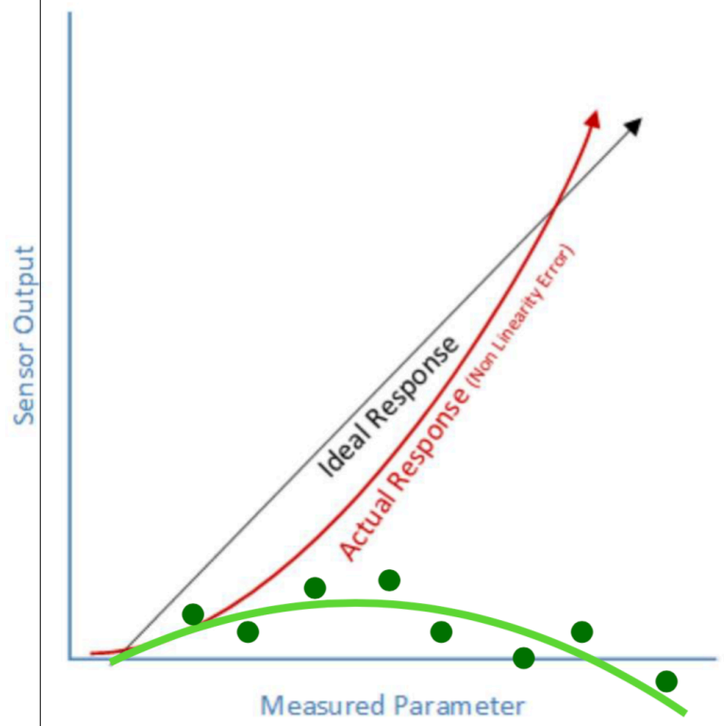
(Raw-RawLow)/(RawHigh-RawLow) = (Correct-RefLow)/(RefHigh-RefLow)



Two-point

For non linear responses. Use curve fitting to find offset function,

Correct = f(Raw)



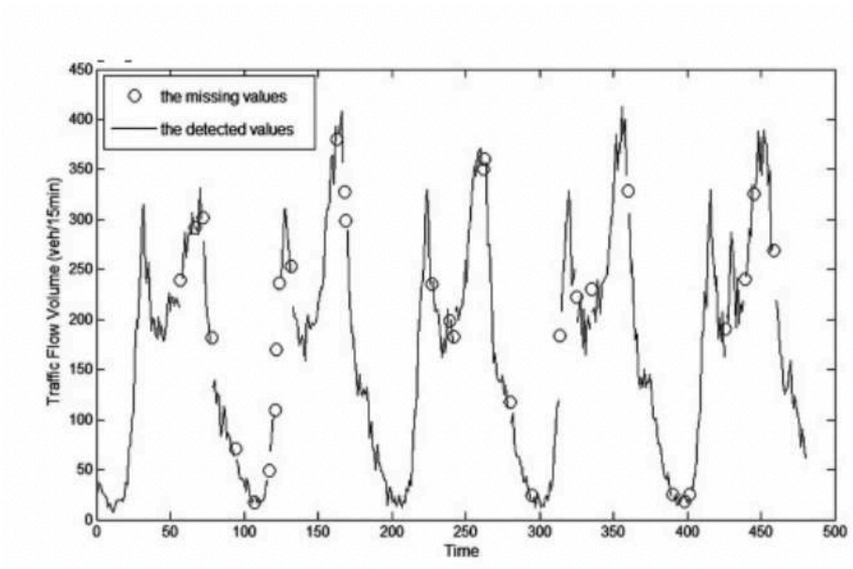
Multi-point

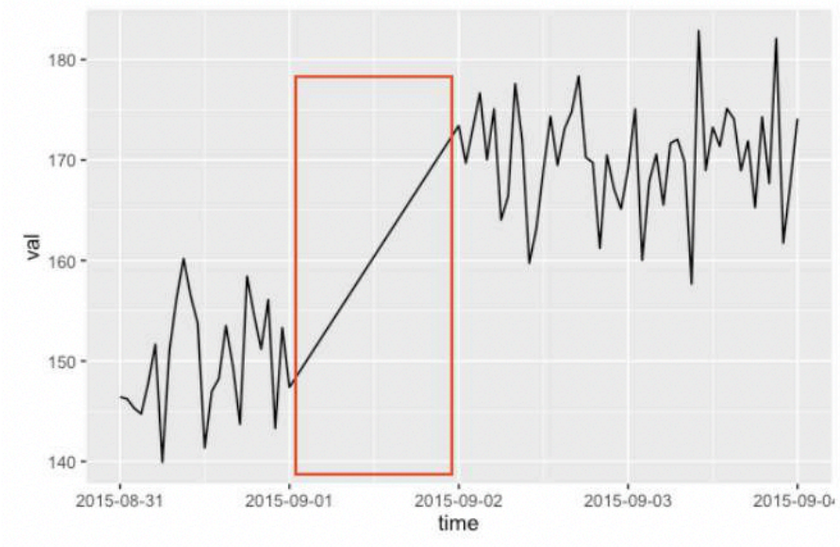
Courtesy: ECE M119 UCLA

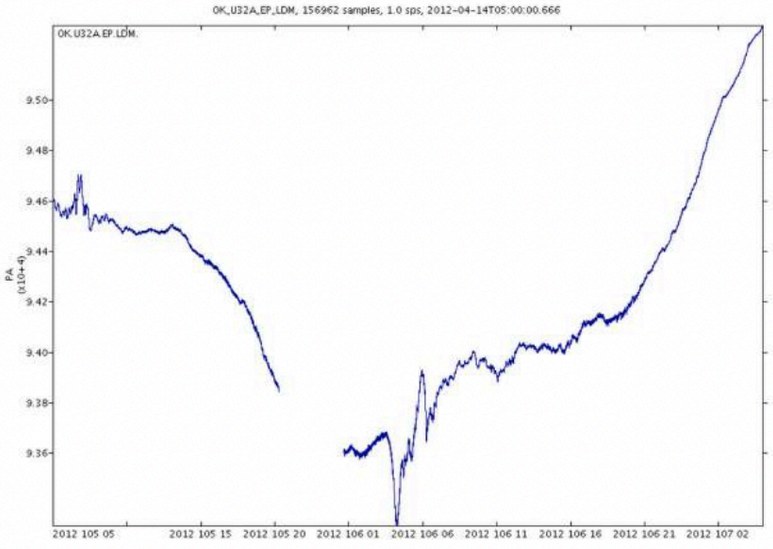
3. Missing Sensor Data

Why data gets missing,

1) Defect, 2) Disconnection, 3) Displacement, 4) Corrupted storage







Courtesy: ECE M119 UCLA

Programming for Analog Sensing

Beaglebone Analog Inputs

- With the digital ports, we measure one of two values (high or low)
- With the analog ports we measure a range of values
- Analog to Digital Converters (ADC) are used to sample analog signals and digitize them
  - Beaglebone Black provides seven analog ports labeled AIN0 through AIN6
  - Its ADC uses **12 bits** with a sampling rate of **200 KSPS** (Kilo Samples Per Second)
  - ADC samples the analog signal when the “start of conversion” signal is high
  - Uses 12 clock cycles to digitize the sampled input; then an “end of conversion” signal is enabled high indicating that the digital data is ready for the program to consume
  - A new conversion cycle can be initiated after the previous data is read

Beaglebone Analog Inputs

- BBB analog pins have 12-bit ADC input:
  - There are 2^{12}= 4,096 possible discrete representations (numbers) for 12-bit sampling resolution
- BBB analog inputs operate at 0 to 1.8V
  - If the input voltage is exactly 0V, we use the decimal number 0 to represent it
  - If the voltage is exactly 1.8V, we can use the number 4,095 to represent it
  - What voltage does the decimal number 1 represent?
    - (1×1.8)/4096 = 0.000439453125V = 0.44 mV
    - Each decimal number between 0 and 4,095 (4,096 values) represents a step of about 0.44mV
- If a sensor outputs a voltage range of –5 to +5V, or more commonly 0V to +5V, we need to scale that range to be in between 0 and 1.8V to be compatible with BBB ADC
  - This enables us to connect many different types of sensors, such as distance sensors, temperature sensors, light intensity sensors, and so on.

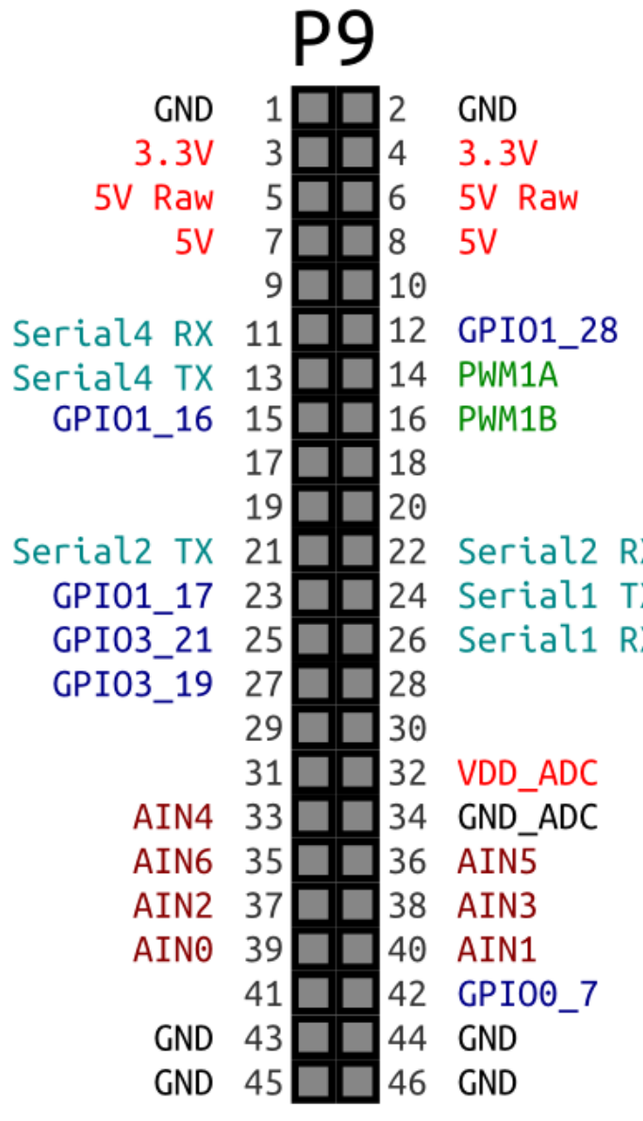
Beaglebone Analog Inputs

- BBB has an interface associated with each AIN input channel
- The BBB interface “in_voltageX_raw” is the raw value of the channel X of the ADC
- For example:
  - “in_voltage**6**_raw” corresponds with AIN**6**
  - “in_voltage**5**_raw” corresponds with AIN**5,** and so on

BeagleBone Pin	**Analog Input**	**interface**
P9_39	AIN0	in_voltage0_raw
P9_40	AIN1	in_voltage1_raw
P9_37	AIN2	in_voltage2_raw
P9_38	AIN3	in_voltage3_raw
P9_33	AIN4	in_voltage4_raw
P9_36	AIN5	in_voltage5_raw
P9_37	AIN6	in_voltage6_raw
P9_32	VDD_ADC	
P9_34	GNDA_ADC	

Beaglebone Analog Inputs

BeagleBone Pin	Analog Input	interface
P9_39	AIN0	in_voltage0_raw
P9_40	AIN1	in_voltage1_raw
P9_37	AIN2	in_voltage2_raw
P9_38	AIN3	in_voltage3_raw
P9_33	AIN4	in_voltage4_raw
P9_36	AIN5	in_voltage5_raw
P9_37	AIN6	in_voltage6_raw
P9_32	VDD_ADC	
P9_34	GNDA_ADC	



Reading an Analog Signal

- Analog inputs are exposed by Industrial I/O subsystem (IIO) through the in_voltageX_raw files
- Go to the IIO (Industrial I/O subsystem) devices directory, and then list the contents of the folder

```
cd /sys/bus/iio/devices/iio:device0
ls
```

```
debian@beaglebone:~$ cd /sys/bus/iio/devices/iio:device0
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ ls
buffer           in_voltage2_raw  in_voltage6_raw  power
dev              in_voltage3_raw  in_voltage7_raw  scan_elements
in_voltage0_raw  in_voltage4_raw  name             subsystem
in_voltage1_raw  in_voltage5_raw  of_node          uevent
debian@beaglebone:/sys/bus/iio/devices/iio:device0$
```

Reading an Analog Signal - Example

- If a sensor is connected to P9_40
  - “in_voltage**1**_raw” should be read for continuous voltage data that is converted to sensor value
  - To read a single ADC from a particular channel, the interface can be used as follows:

```
cat /sys/bus/iio/devices/iio\:device0/in_voltage1_raw
```

(noise voltage values if you haven't connected a sensor yet)

C Program for Analog Sensing

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

// Read the digital value from the ADC interface.
const char AIN_DEV[] = "/sys/bus/iio/devices/iio:device0/in_voltage1_raw";
// function to convert Celcius to Farenheit
double CtoF(double c){
  return (c * 9.0 / 5.0) + 32.0;
}
// Function to extract temperature from TMP36 data
double temperature(int value){
  // Convert the digital value to millivolts.
  double millivolts = (value / 4096.0) * 1800;
  // Convert the millivolts to Celsius temperature.
  // Celcius conversion from the TMP36 Datasheet
  double temperature = (millivolts - 500.0) / 10.0;
  return temperature;
}
int main(){
  FILE *f;
  int a2dReading = 0;
  while (1){
    f = fopen(AIN_DEV, "r");
    if (f==NULL) {
        printf("ERROR: Unable to open voltage input file.\n");
        exit(-1);
    }
    // Get analog to digital reading
    fscanf(f, "%d", &a2dReading);
    fclose(f);
    double celsius = temperature(a2dReading);
    double fahrenheit = CtoF(celsius);
    printf("digital value: %d  celsius: %f  fahrenheit: %f\n", a2dReading, celsius, fahrenheit);
    sleep(1);
  }
  return 0;
}
```

Read this file for sensor data represented as bits

Convert bits to voltage

Convert voltage to temperature

Open the input channel “in_voltage1_raw”

Read the input channel “in_voltage1_raw”

Analog Sensing output

- Save the code above in the file named “temperature_read.c”, use the following commands to compile and run the code

```
gcc temperature_read.c -o read
```

- Run the executable:

```
./read
```

- You should see the following output

```
debian@beaglebone:~/test$ gcc temperature_read.c -o read
debian@beaglebone:~/test$ ./read
digital value: 1567 celsius: 18.862305  fahrenheit: 65.952148
digital value: 1568 celsius: 18.906250  fahrenheit: 66.031250
digital value: 1566 celsius: 18.818359  fahrenheit: 65.873047
digital value: 1567 celsius: 18.862305  fahrenheit: 65.952148
digital value: 1567 celsius: 18.862305  fahrenheit: 65.952148
digital value: 1567 celsius: 18.862305  fahrenheit: 65.952148
digital value: 1566 celsius: 18.818359  fahrenheit: 65.873047
digital value: 1567 celsius: 18.862305  fahrenheit: 65.952148
```

Warning: The analog inputs of the BBB operate at 1.8V. Since the TMP36 has a theoretical maximum output of 3.3V, there is a potential for the BBB to be damaged if the voltage in millivolts exceeds 1.8V. This will only happen on a TMP36 if the temperature exceeds 130 degrees C (266 degrees F).

Reading

- Textbook: Chapter # 9: Interfacing with the Physical Environment
- Read relevant topics
- Try programming exercises

Lab Assignment # 5 - Due Saturday March 30th

- Configure one pin as GPIO input and capture rising edge interrupt on it. Configure another pin as PWM, generate pwm on it, and supply this pwm to the GPIO pin.
- Interface temperature sensor to P9_40.
- In a multithreaded C application,
  - In one thread, timestamp and read sensor data at the PWM rising edges, and store the data in a shared array.
  - In the second thread, read the shared array, print it on terminal, and also write the data to a file. (**Data reads and writes should be in order*) Plot the data.