INDIAN INSTITUTE OF TECHNOLOGY PATNA

EC3101: Microcontroller & Embedded Systems Lab

EXPERIMENT NO: 07

Implement FIR and IIR

Submitted By	Shaurya Aggarwal
Roll No.	2301EE56
Group No.	2 (Thursday)

Aim:

To implement and compare 2 fundamental types of digital filters:
the Finite Impulse Response (FIR) filter and,
the Infinite Impulse Response (IIR) filter,
using an Arduino platform for signal processing.

Software used:

Arduino IDE

Theory

Digital filters are algorithms used to modify or enhance a signal. They are broadly categorized into two main types:

1. IIR (Infinite Impulse Response) Filter

An IIR filter calculates its current output using both current and past input values, as well as past output values. This use of feedback means that a single impulse at the input can theoretically produce an output that lasts forever, hence the name "Infinite Impulse Response". The general equation for an IIR filter is:

$$y[n] = b_0x[n] + b_1x[n-1] + ... - a_1y[n-1] - a_2y[n-2] - ...$$

Here, coefficients manage the feedback from previous outputs.

2. FIR (Finite Impulse Response) Filter

An FIR filter, in contrast, calculates its output using only current and past input values. It does not use any feedback from previous outputs. As a result, its response to a single impulse will eventually settle to zero after a finite number of samples[6]. The equation for an FIR filter is:

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + ... + b_{N-1}x[n-(N-1)]$$

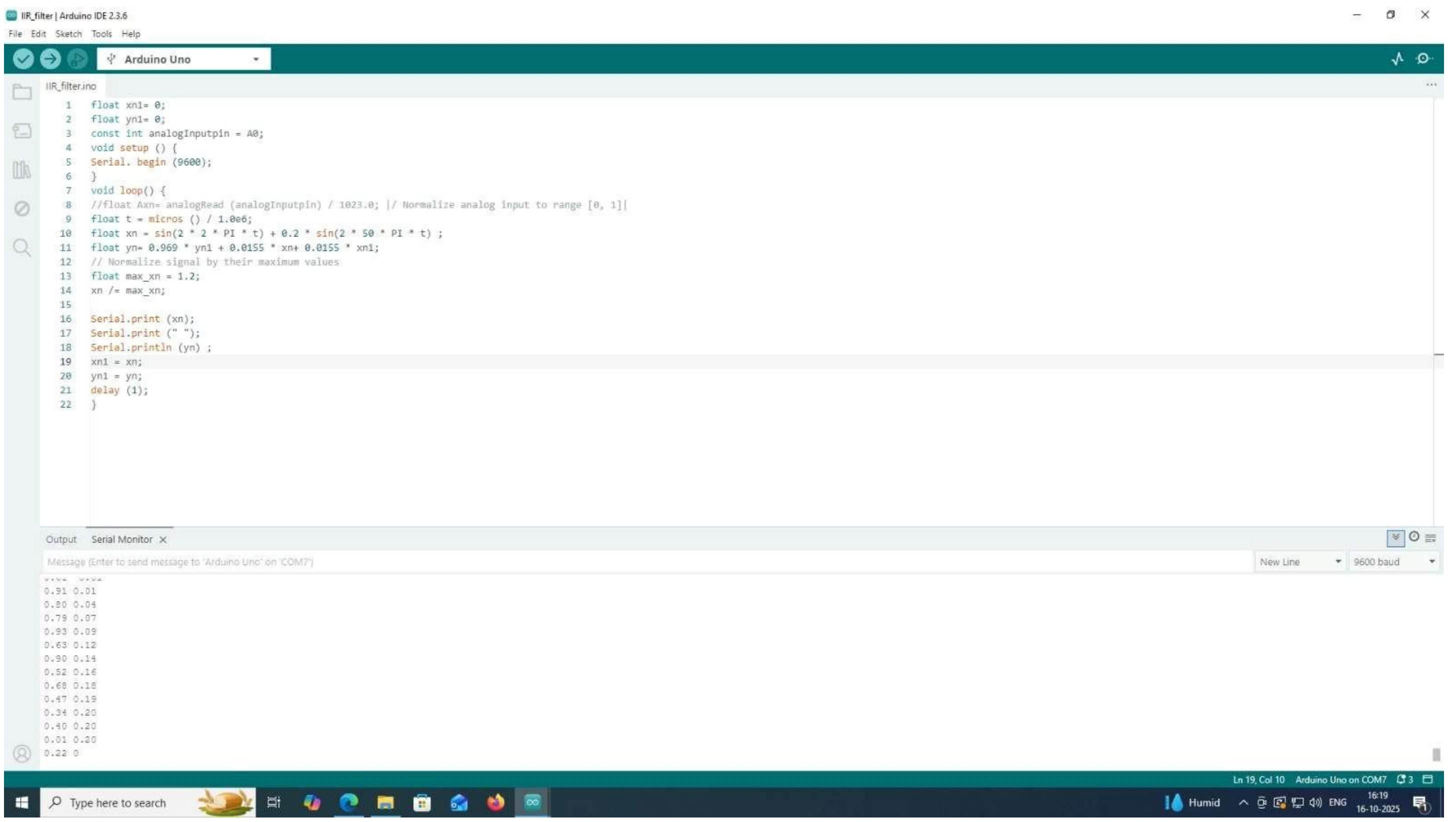
Where N is the number of taps or the order of the filter.

Implementation and Results

In this experiment, a noisy signal was synthetically generated within the Arduino code. This signal was created by combining a low-frequency sine wave (the desired signal) with a high-frequency sine wave (the noise). Both the IIR and FIR filters were implemented as low-pass filters designed to remove the high-frequency noise.

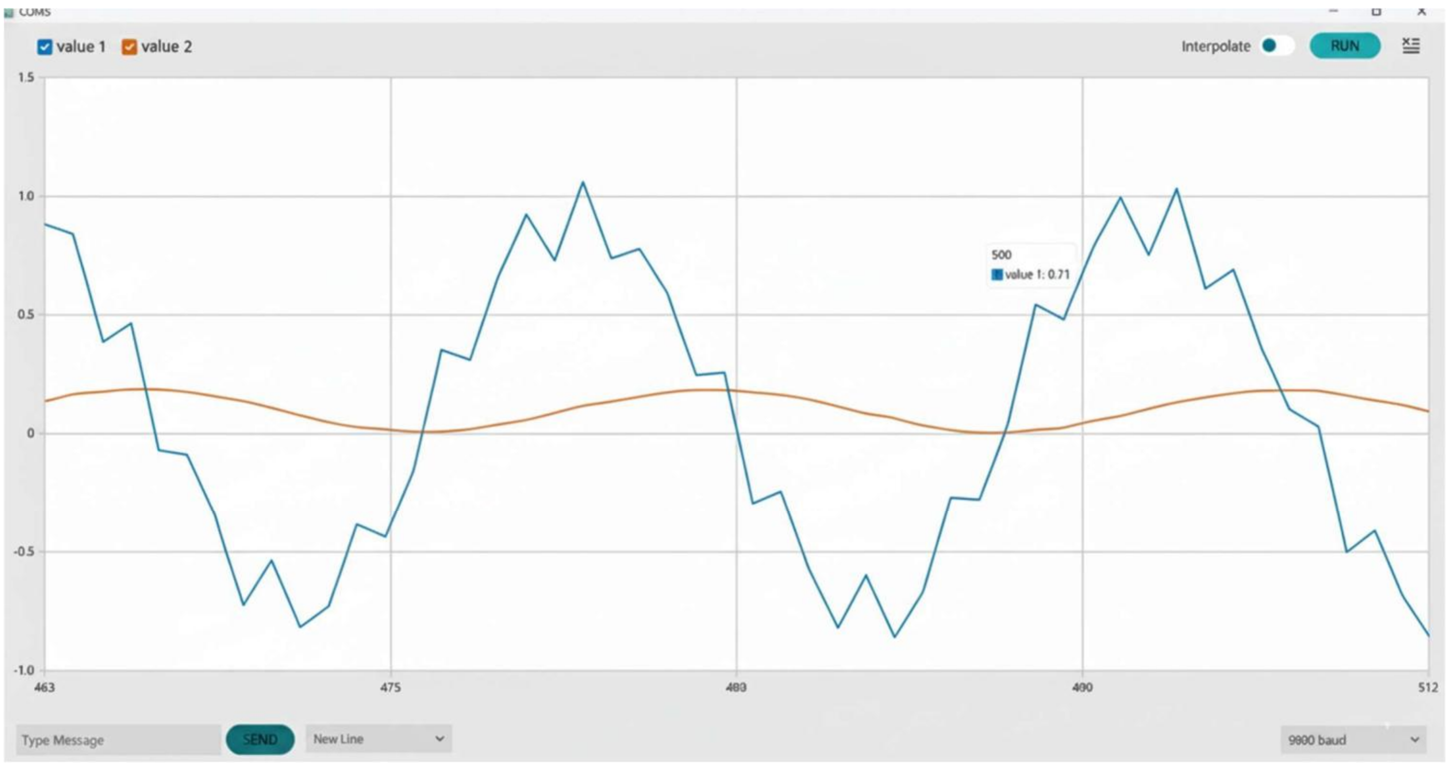
- *IIR Filter Implementation*

The IIR filter was implemented using a difference equation that includes a feedback term (0.969*yn1). The Arduino's serial plotter was used to visualize both the noisy input signal and the filtered output signal. As shown in the plot, the IIR filter successfully attenuated the high-frequency noise, resulting in a much smoother sine wave that closely resembles the original, clean signal.


IIR_filter | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Uno
IIR_filter.ino
Output
Serial Monitor
Message (Enter to send message to 'Arduino Uno' on 'COM7')
New Line
9600 baud
Ln 19, Col 10 Arduino Uno on COM7
Type here to search
Humid
ENG
16:19
16-10-2025

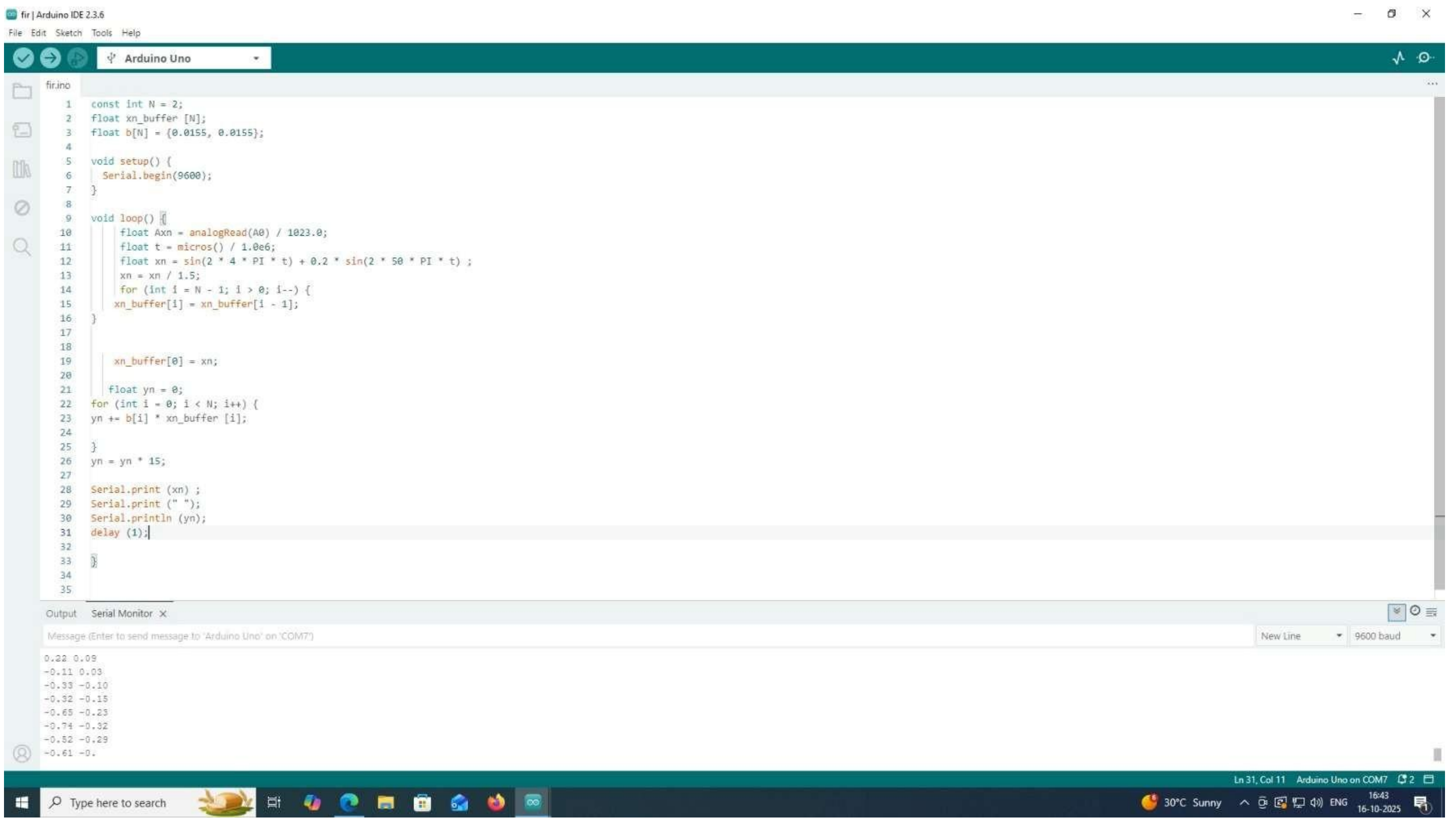
```
float xn1= 0;
float yn1= 0;
const int analogInputpin = A0;
void setup () {
Serial. begin (9600);
}
void loop() {
//float Axn= analogRead (analogInputpin) / 1023.0; |/ Normalize analog input to range [0, 1]|
float t = micros () / 1.0e6;
float xn = sin(2 * 2 * PI * t) + 0.2 * sin(2 * 50 * PI * t) ;
float yn= 0.969 * yn1 + 0.0155 * xn+ 0.0155 * xn1;
// Normalize signal by their maximum values
float max_xn = 1.2;
xn /= max_xn;

Serial.print (xn);
Serial.print (" ");
Serial.println (yn) ;
xn1 = xn;
yn1 = yn;
delay (1);
}
```

```
0.91 0.01
0.80 0.04
0.79 0.07
0.93 0.09
0.63 0.12
0.90 0.14
0.52 0.16
0.68 0.18
0.47 0.19
0.34 0.20
0.40 0.20
0.01 0.20
0.22 0
```



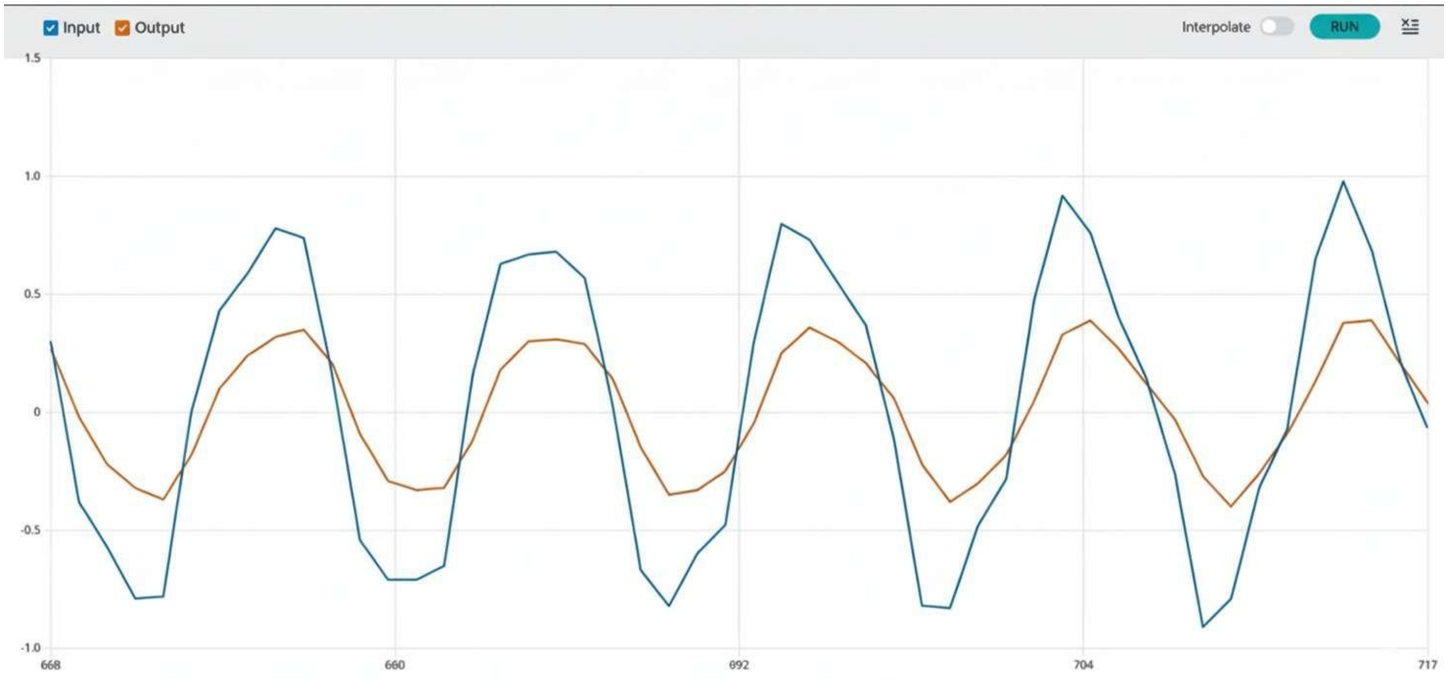
- ***<u>FIR Filter Implementation</u>***

The FIR filter was implemented by storing previous input samples in a buffer and applying a set of filter coefficients (b). This specific implementation acts as a moving average filter. The resulting plot again demonstrates effective filtering. The output signal is significantly smoother than the noisy input, confirming that the FIR filter also successfully removed the unwanted high-frequency components.



```
const int N = 2;
float xn_buffer [N];
float b[N] = {0.0155, 0.0155};

void setup() {
  Serial.begin(9600);
}

void loop() {
      float Axn = analogRead(A0) / 1023.0;
      float t = micros() / 1.0e6;
      float xn = sin(2 * 4 * PI * t) + 0.2 * sin(2 * 50 * PI * t) ;
      xn = xn / 1.5;
      for (int i = N - 1; i > 0; i--) {
     xn_buffer[i] = xn_buffer[i - 1];
}


     xn_buffer[0] = xn;

    float yn = 0;
for (int i = 0; i < N; i++) {
yn += b[i] * xn_buffer [i];

}
yn = yn * 15;

Serial.print (xn) ;
Serial.print (" ");
Serial.println (yn);
delay (1);

}
```

```
0.22 0.09
-0.11 0.03
-0.33 -0.10
-0.32 -0.15
-0.65 -0.23
-0.74 -0.32
-0.52 -0.29
-0.61 -0.
```



Input
Output
Interpolate
RUN
1.5
1.0
0.5
0
-0.5
-1.0
668
680
692
704
717

Conclusion

This experiment successfully demonstrated the practical application of both IIR and FIR digital filters for noise reduction. The key differences between the two were observed:

- FIR filters are inherently stable and can be designed to have a linear phase response, which is crucial for applications where preserving the signal's timing is important. However, they often require a higher order (more calculations) to achieve the same level of filtering as an IIR filter.
- IIR filters are computationally more efficient, requiring fewer coefficients and calculations to achieve a sharp cutoff. The trade-off is the potential for phase distortion and instability if the filter is not designed carefully. Ultimately, the choice between an FIR and IIR filter depends on the specific requirements of the application.