Digital Filter Design

The following outlines the design of the Digital Filter inside the Arduino R3

Requirements

1. Filter must pass voltages above a certain threshold, and attenuate voltages below a certain threshold

Public Methods

- ***float Filter(int data)***
  - Function:
    - Public method for filtering the digital data. Calls the private filtering method.
    - Ensures that the data is properly filtered, even if errors occur.
  - Arguments:
    - data (int): the digital data passed which needs to be filtered
  - Returns:
    - float value of filtered data

Private Methods

- ***Pair _TryFilter(int data)***
  - Function:
    - Attempts to filter the given data using a specific voltage threshold.
  - Arguments:
    - data (int): The raw EMG data
  - Returns:
    - Returns a boolean indicating if filtering was successful, and the filtered value.

Testing

To test the filter algorithm, we will copy the algorithm shown below, fill in the setup function properly, and use a c++ random number generator to test the filtering. We will use the `Serial.print()` method

to print the random number, and then reuse the method to print the output of the filter given that value. The `loop()` will look something like this:

```
float signal = float(rand() % 1023);    // Random number between 0 and .09
float filteredSignal = Filter(signal);  // Filter the signal

// Print the Filtered Signal value
Serial.print("Random Signal = ");
Serial.println(signal);

// Print the Filtered Signal value
Serial.print("Filtered Signal = ");
Serial.println(filteredSignal);
Serial.println();

delay (100);    // Delay 100 ms
```

Future Improvements

For future improvements to the Digital Filter, a convolutional filter could be implemented. This would allow us to use values at different power levels while still attenuating noise.

A convolutional filter requires an array of input signals to convolve upon, given that the signal is not continuous. This would require a queue of data, inherently slowing down function of the hand. Depending on the size of the queue (which impacts the quality of the filter), an input signal can take more time to be processed into the Control Algorithm. For our purposes, we wanted fast control of the fingers, but an implementation of a complex filter could allow more accurate movements.

Algorithm

```
/* Constants declared in header */
const float SIGNAL_THRESHOLD = 6.1;   // Voltage threshold from testing

struct Pair {
        bool success;
        float data;
    Pair(bool s, float d) : success(s), data(d) {}
};

.
.
.

void setup() {

    .
    .
    .

}

.
.
.

void loop() {

    .
    .
    .

    /* Read EMG signal */
        float filteredSignal = Filter(rawSignal);


        .
        .
        .

}
```

```
float Filter(float data) {
        /*
        - Function:
                - Public method for filtering the digital data. Calls the private filtering meth
                - Ensures that the data is properly filtered, even if errors occur.
        - Arguments:
                - data (float): the raw EMG data in V that needs to be filteed
        - Returns:
                - float value of filtered data
        */

        Pair filterPair = _TryFilter(data);

        if (!filterPair.success) {
                Serial.println('Filter Failure')
                return 0.0;
        }

        return filterPair.data;

}

Pair _TryFilter(float data) {
        /*
        - Function:
                - Attempts to filter the given data using a specific voltage threshold.
        - Arguments:
                - data (float): The raw EMG data in Volts
        - Returns:
                - Returns a boolean indicating if filtering was successful, and the filtered val
        */

        Pair tryFilter;

        if(data < SIGNAL_THRESHOLD) {
                tryFilter.data = 0.0;
                tryFilter.success = true;
        }
        else {
                tryFilter.data = data;
                tryFilter.success = true;
        }
```

```
        return tryFilter;

}
```