Control Algorithm Design

The Control Algorithm is a looped process which handles the control of the DC Servo Motors.

Requirements

For each motor:

1. Power DC Motor during muscle contraction
2. Read DC Motor's current using current sensor input
3. Stop DC Motor during overdraw
4. Control 3 motor states:

- Turn
- Hold
- Release

Motor States

The follow section outlines the 3 motor states which describe the motor.

Turn State

During the ***turn*** state, the motor should be in constant power and rotation. The motor will wind the finger up to simulate gripping

```
myServo.write(rotation speed between 90 and 180)
```

Hold State

During the ***hold*** state, the motor should stop rotating, but not unwind. The motor will simply hold its position

Because there are natural current spikes which fall above our current threshold, we need to implement a check that the current is "overdrawing" consecutively in time

```
myServo.write(90)
```

Release State

During the ***release*** state, the motor should stop rotating and unwind. The motor should simulate releasing its grip

```
myServo.write(rotation speed between 0 and 90)
```

Public Methods

- ***void ControlMotors(float filteredSignal, float sensorReadings[])***
  - Function:
    - Full control algorithm for motors
  - Arguments:
    - filteredSignal (float): Filtered EMG data
    - sensorReadings (floats): Current sensor readings

Testing

To test the Control Motor algorithm, we will copy the algorithm show below, fill in the setup function proprerly, and use a random number generator to simulate an input signal for the EMG signal. We will use the `Serial.print()` method to print the random signal so we can ensure the motor is acting as intended. We will add print statements in the `ControlMotor()` method to print the state and the angle being written to ensure the function acts properly. We will also use print statements to log the current sensor readings to ensure the current levels fall below threshold (which also allows us to calibrate the threshold level). Throughout testing, we will sporadically use our hands to manually stop the motor, forcing overcurrent. We will ensure the proper current senses overdraw and turns to the hold state. The `loop` method will look something like this:

```
float filteredSignal = float(rand() % 10) / 100;   // Random number between 0 and .09

float currentReadings[5];

for (int i = 0; i < 5; i++) {
    currentReadings[i] = analogRead(CURRENT_PINS[i]);
}

ControlMotors(filteredSignal, currentReadings);
```

Algorithm

```
/* Constants declared in header */
const int CURRENT_PINS[5] = {A1, A2, A3, A4, A5};
const int MOTOR_PINS[5] = {9, 10, 11, 12, 13};
const float CURRENT_THRESHOLD = 2;   # Example current threshold level in range (0 : 1023)
const float SIGNAL_THRESHOLD = 0.03;   # Example voltage threshold level in range (0 : 1023)
bool isOverdrawn[5] = {false, false, false, false, false};  // array of bools representing if tl
const Servo MOTORS[5];
float totalRotation[5] = {0.0, 0.0, 0.0, 0.0, 0.0}; // array of total angle rotated by each moto
const float RELEASE_STEP = 10.0; // constant for how much the totalRotation will decrement each

.
.
.

void setup() {

    .
    .
    .

    // Iterate through each motor
    for (int i = 0; i < 5; i++) {
        MOTORS[i].attach(MOTOR_PINS[i]); // Attach motors to their output pins
    }

}

.
.
.

void loop() {

    .
    .
    .

    /* Read EMG signal and filter it */

    /* Read current sensor pins */
    ControlMotors(filteredSignal, sensorReadings);

}

void ControlMotors(float filteredSignal, float sensorReadings[]) {
    /*
    - Function:
        - Full control algorithm for motors
    - Arguments:
        - filteredSignal (float): Filtered EMG data
        - sensorReadings (floats): Current sensor readings
    */


    // Check that the EMG signal is powering the motors
    if (filteredSignal > SIGNAL_THRESHOLD) {

        // Iterate through each current sensor pin
        for (int i = 0; i < 5; i++) {

            if (sensorReadings[i] > CURRENT_THRESHOLD) {    // If overdrawing current

                if (isOverdrawn[i]) {   // If it is consecutively overdrawn
                    // Set to hold state
                    MOTORS[i].write(90);
                    continue;  // This should break out of line 133 loop, but remain in for-loo
                }

                isOverdrawn[i] = true;  // Record that this motor has overdrawn current

                // Continue rotating
                float rotation = map(filteredSignal, SIGNAL_THRESHOLD, 0.5, 90, 180);    // Map
                MOTORS[i].write(rotation);
                totalRotation[i] += rotation;
            }

            else {
                // Set to turn state
                isOverdrawn[i] = false;
                float rotation = map(filteredSignal, SIGNAL_THRESHOLD, 0.5, 90, 180);    // Map
                MOTORS[i].write(rotation);
                totalRotation[i] += rotation;
            }
        }
    }
    else {
        // Release state
        for (int i = 0; i < 5; i++) {
            isOverdrawn[i] = false;

            // Check if the motor has moved at all yet
            if (totalRotation[i] > 0) {
                MOTORS[i].write(80);  // slowly reverse motor
                totalRotation[i] -= RELEASE_STEP;  // arbitrary value (requires testing)

                if (totalRotation[i] <= 0) { // once the motor has gotten to its return state
                    totalRotation[i] = 0;
                    MOTORS[i].write(90);  // stop movement
                }
            }
            else {
                MOTORS[i].write(90);  // already at original position, stop
            }
        }
    }

}
```