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

Private Methods

- ***void _UpdateState(motor currMotor, float sensorReading, MotorState newState)***
  - Function:
    - Updates the state of a motor and sets its values accordingle
  - Arguments:
    - currMotor (motor): motor struct of the current motor being iterated
    - sensorReading (float): current-sensor reading of this specific motor
    - newState (MotorState): the MotorState to set the motor to
- ***void _UpdateTurnState(motor& currMotor, float filteredSignal)***
  - Function:
    - Updates the currMotor to the TURN state
    - Writes a rotation value between 90 and 180 based on filteredSignal level
  - Arguments:
    - currMotor (motor): the current motor we are iterating through
    - filteredSignal (float): the EMG signal from the MyoWare sensor
- ***void _UpdateHoldState(motor& currMotor, float filteredSignal)***
  - Function:

        - Updates the currMotor to the HOLD state
        - Writes a rotation of 90 to the servo, keeping it still
    - Arguments:
        - currMotor (motor): the current motor we are iterating through
        - filteredSignal (float): the EMG signal from the MyoWare sensor
- ***void _UpdateReleaseState(motor& currMotor, float filteredSignal)***
    - Function:
        - Updates the currMotor to the RELEASE state
        - Writes a rotation value of 80 to the motor while decrementing the totalRotation
    - Arguments:
        - currMotor (motor): the current motor we are iterating through
        - filteredSignal (float): the EMG signal from the MyoWare sensor

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


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

ControlMotors(filteredSignal);
```

Future Improvements

After switching to the Continuous-Rotation Servo Motors, we lost the ability to track the displacement of the finger over time. This means that the RELEASE state is guessing how far to release. Implementing a Limit Switch would allow us to sense when the finger returns to its neutral position. However, without this component, the best we can do is guess. Additionally, future developments will need separate batteries for each motor-current-sensor loop. When a single motor stalls, the current of every motor is affected due to the limited power capable by the battery; therefore, the CURRENT_THRESHOLD level is only accurate for the first stall. After several attempts at designing a dynamic current threshold dependent on the number of stalled motors, we elected to truncate the independent finger feature and settle with dependent movement.

Algorithm

```
struct motor {
  Servo servo;
  int overdrawn = 0.0;
  float totalRotation = 0.0;
  MotorState state = RELEASE;
  float sensorReading = 0.0;
};

enum MotorState {
  TURN,
  RELEASE,
  HOLD
};

/* Constants declared in header */
const int CURRENT_PINS[5] = {A1, A2, A3, A4, A5};
const int NUM_MOTORS = 5;
const int MOTOR_PINS[5] = {3, 5, 6, 9, 11};
const float SIGNAL_THRESHOLD = 6.1;   // Example voltage threshold level in range (0 : 1023)
const float CURRENT_THRESHOLD = 460;   // Example voltage threshold level in range (0 : 1023)
const motor MOTORS[5];
const float RELEASE_STEP = 20.0; // constant for how much the totalRotation will decrement each

.
.
.

void setup() {

    .
    .
    .

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

}
```

```
.
.
.

void loop() {

    .
    .
    .

    /* Read EMG signal and filter it */

    /* Read current sensor pins */

    ControlMotors(filteredSignal);

}

void ControlMotors(float filteredSignal) {
  /*
  - Function:
      - Full control algorithm for motors
  - Arguments:
      - filteredSignal (float): Filtered EMG data
  */

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

    // Iterate through each current sensor pin
    for (int i = 0; i < NUM_MOTORS; i++) {
      if (MOTORS[i].sensorReading < CURRENT_THRESHOLD) { // Check if that motor's current readi
        MOTORS[i].state = HOLD; // Set motorState to HOLD
      }
      else {
        MOTORS[i].state = TURN; // Set motorState to TURN
      }
      _UpdateState(MOTORS[i], filteredSignal);  // Update the state
    }
  }
  else {
    for (int i = 0; i < NUM_MOTORS; i++) {
      MOTORS[i].state = RELEASE;  // Set the motor state to RELEASE
      _UpdateState(MOTORS[i], 0.0); // Update the motor states to RELEASE
    }
  }
}

void _UpdateState(motor& currMotor, float filteredSignal) {
  /*
    - Function:
      - Updates the state of a motor and sets its values accordingle
    - Arguments:
      - currMotor (motor): motor struct of the current motor being iterated
      - filteredSignal (float): EMG reading from MyoWare
  */

  if (currMotor.overdrawn > 0 && currMotor.overdrawn < 15) { // No matter the new state, keep H(
    currMotor.state = HOLD;
  }

  switch(currMotor.state) {
    case (TURN):
      _UpdateTurnState(currMotor, filteredSignal);
      break;
    case (HOLD):
      _UpdateHoldState(currMotor, filteredSignal);
      break;
    case (RELEASE):
      _UpdateReleaseState(currMotor, filteredSignal);
      break;
  }
}

void _UpdateTurnState(motor& currMotor, float filteredSignal) {
  /*
    - Function:
      - Updates the currMotor to the TURN state
    - Arguments:
      - currMotor (motor): the current motor we are iterating through
      - filteredSignal (float): the EMG signal from the MyoWare sensor
  */

  // Set to turn state
  currMotor.overdrawn = 0;  // Reset overdrawn counter
  float rotation = constrain(map(filteredSignal, SIGNAL_THRESHOLD, 205, 90, 180), 90, 180);    /

  currMotor.servo.write(rotation);  // Send rotation signal to the servo

  currMotor.totalRotation += rotation;  // Log that rotation (used for returning to original pos
}

void _UpdateReleaseState(motor& currMotor, float filteredSignal) {
  /*
    - Function:
      - Updates the currMotor to the RELEASE state
    - Arguments:
      - currMotor (motor): the current motor we are iterating through
      - filteredSignal (float): the EMG signal from the MyoWare sensor
  */

  currMotor.overdrawn = 0;  // Reset overdrawn counter

  // Check if the motor has moved at all yet
  if (currMotor.totalRotation > 0) {  // If it has moved at all

    currMotor.servo.write(80);  // slowly reverse motor

    currMotor.totalRotation -= RELEASE_STEP;  // Decrement the totalRotation

    if (currMotor.totalRotation <= 0) { // once the motor has gotten to its return state
      currMotor.totalRotation = 0;  // Reset totalRotation count
      currMotor.servo.write(90);  // stop movement
    }
  }
  else {
    currMotor.servo.write(90);  // already at original position, stop
  }
}

void _UpdateHoldState(motor& currMotor, float filteredSignal) {
  /*
    - Function:
      - Updates the currMotor to the HOLD state
    - Arguments:
      - currMotor (motor): the current motor we are iterating through
      - filteredSignal (float): the EMG signal from the MyoWare sensor
  */
  // TODO: Need logic here to keep it stopped when stalled but not stop others

  currMotor.overdrawn++;  // Increment overdrawn current
  currMotor.servo.write(90);  // Write no movement to the motor
}
```