Introduction to Biomedical Engineering

Section 4: Basics of High-level programming: Matlab

Lecture 4.3 Closed-loop control of bionic prosthetics

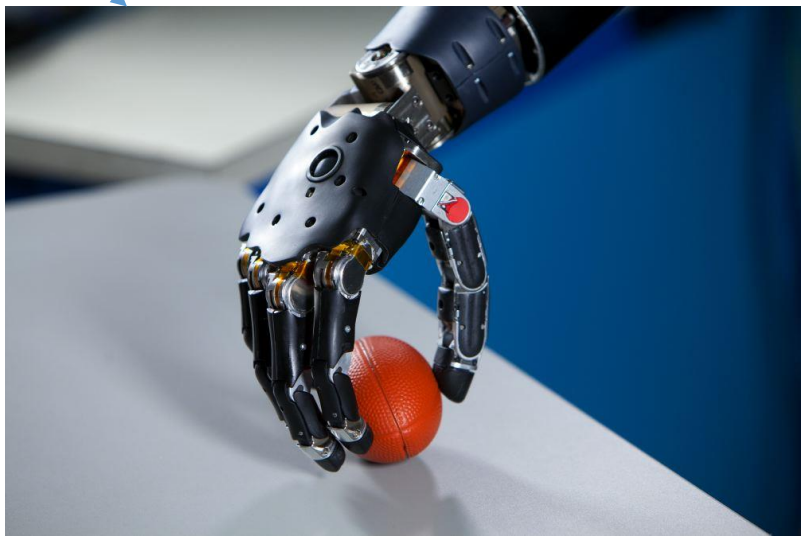
POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

OPEN
POLYTECH

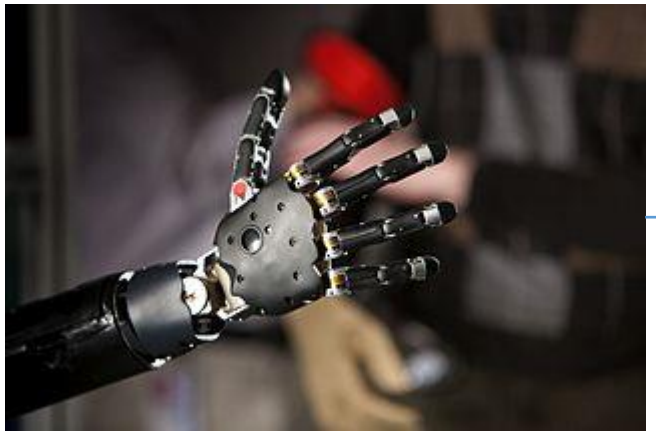
Problem statement

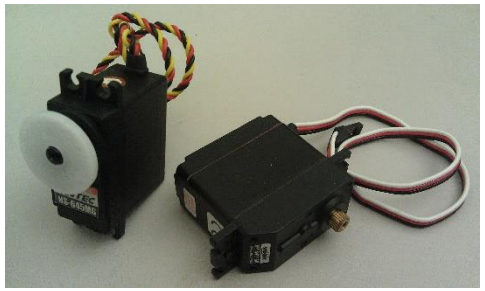
Measurements

Embedded system



Thing to control

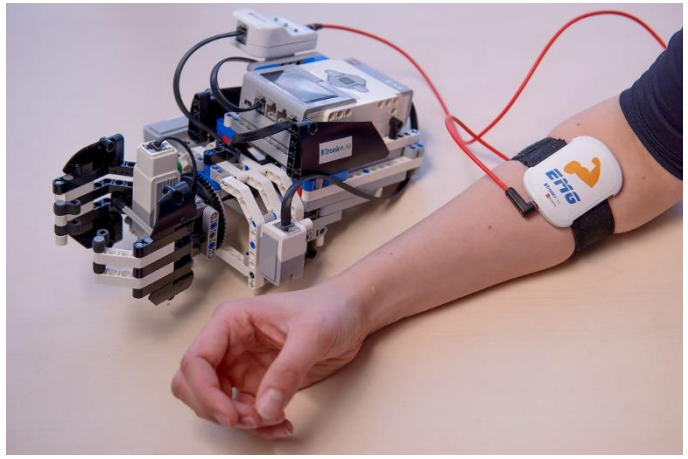


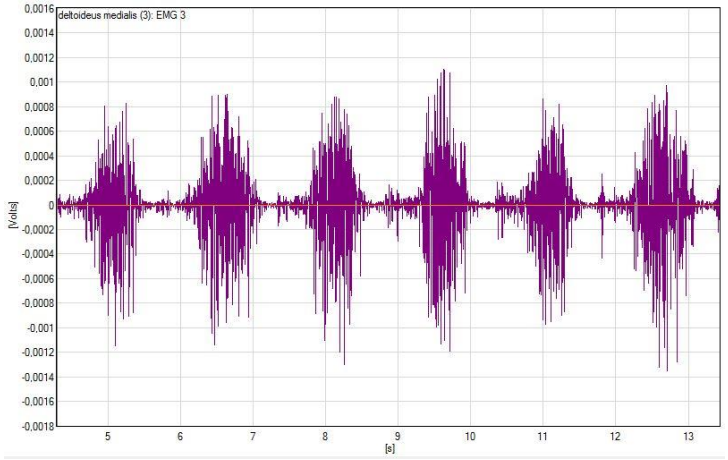


We will control the motor speed

Thing to measure

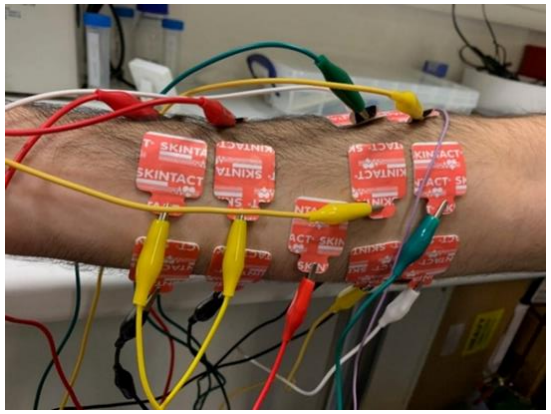
- EMG (Electromyogram) is the surface potential measurements of the muscle activity:
  - Each muscle fibre generates action potential when active
  - More fibres → More activity → stronger force
  - More fibres → More activity on the EMG

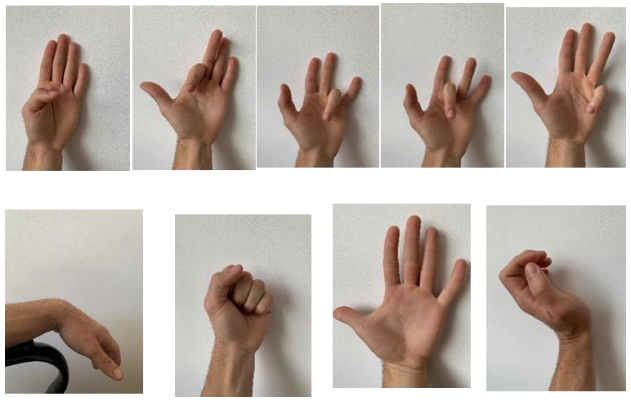




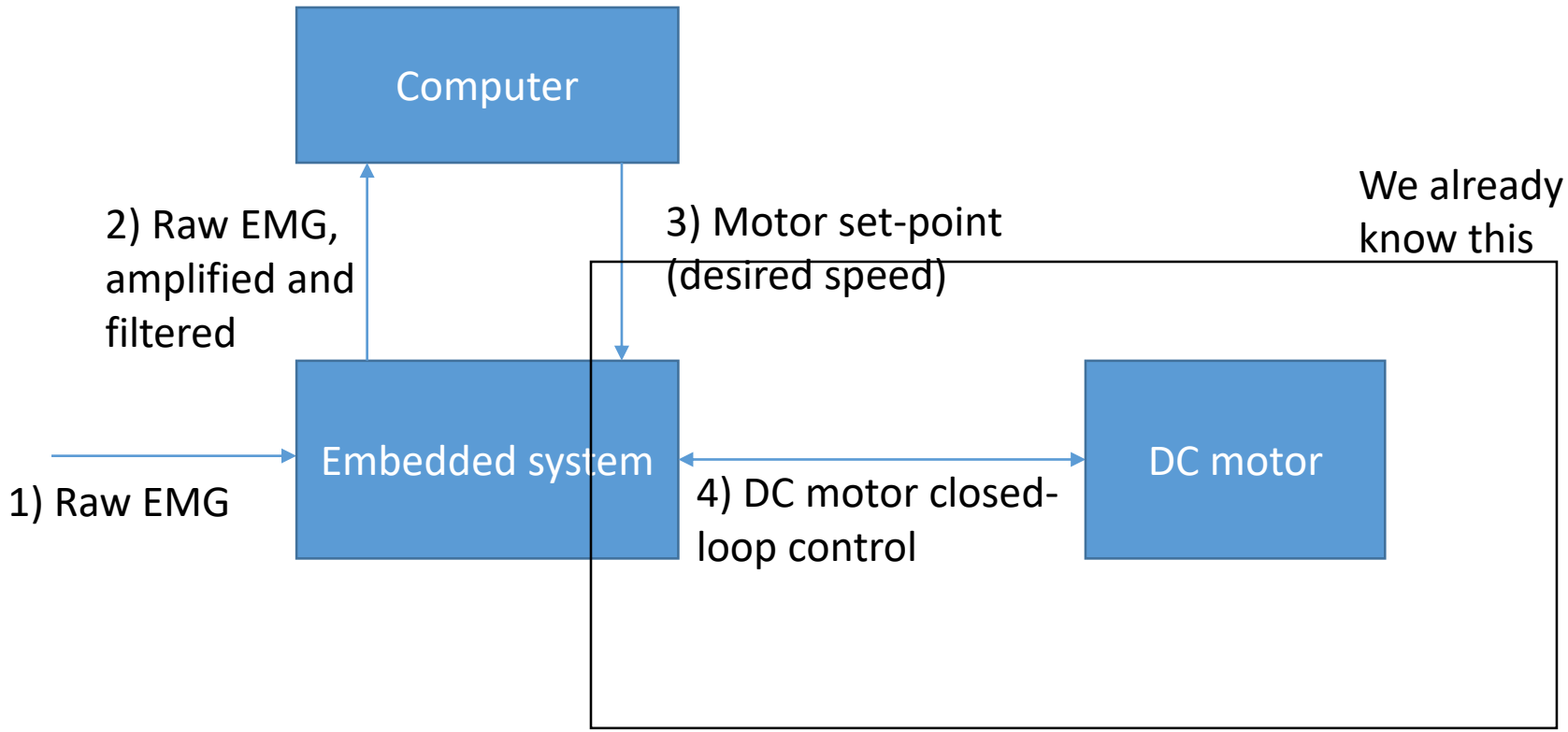
Characteristics of the signal

- Stochastic
- Frequency band 1-100 Hz
- Needs amplification
- Potentials are stronger if they are closer to the electrode → Location of the electrode gives information on the nearby muscle → several electrodes to resolve complex motion where several muscle are active





For now, lets concentrate on controlling one finger with one sensor

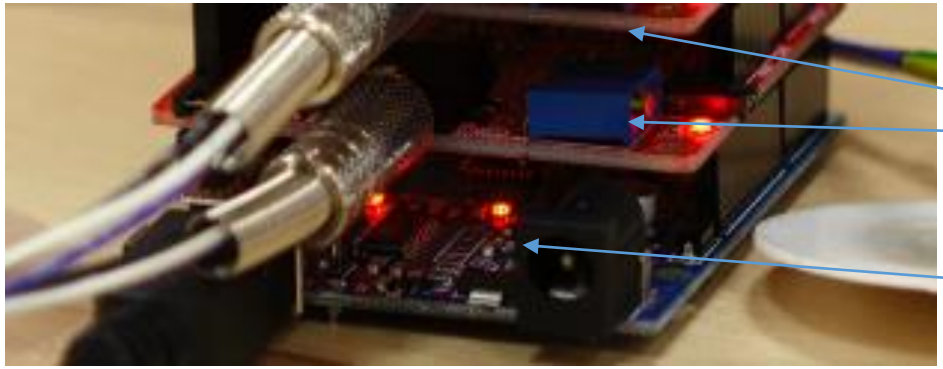


What we need

- Stuff from Lecture 3.3
  - Arduino Uno
  - Motor
  - Rotary encoder
  - Power supply
  - TIP120 transistor (for PWM control of speed) and resistor
- Olimex Arduino EMG shield

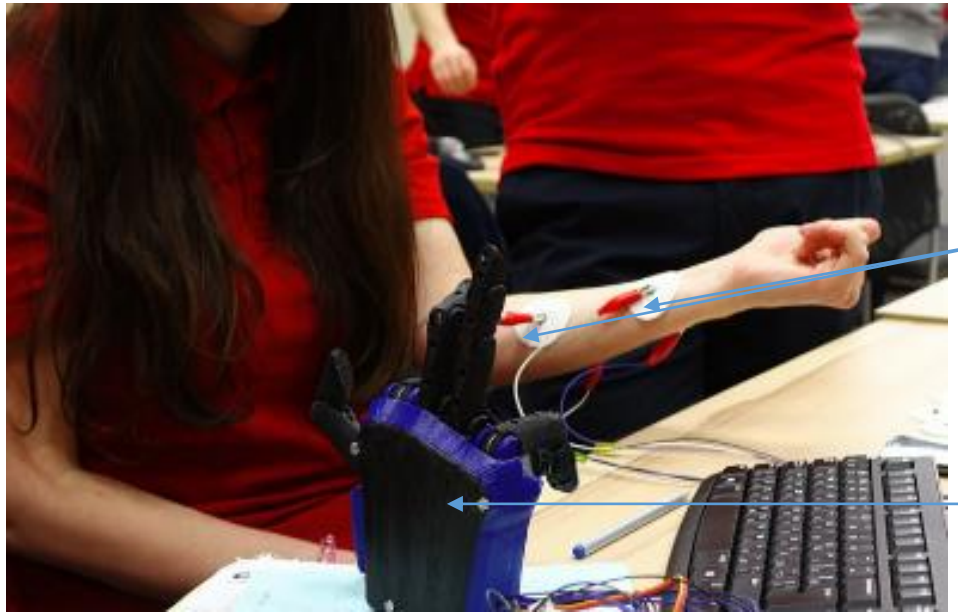
- Sticky electrodes

Full assembly



Olimex shield (you can stuck them channel-by-channel, 2 shown here)

Arduino



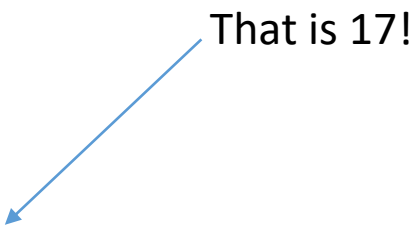
Sticky electrodes

DC motor

Start with acquiring data real-time

- 1. Understand which pin to read from Arduino.
- 2. follow the instructions of the shield manufacturer (they use interruptions and flexible timer to read the data)
  - #include <FlexiTimer2.h>
  - //http://www.arduino.cc/playground/Main/FlexiTimer2
- 3. Design the data structure to cope with the incoming data
- Alternative: ignore all above and load the example included (ShieldEkgEmgDemo.ino)!

Data structure and transmission



- #define NUMCHANNELS 6
- #define HEADERLEN 4
- #define PACKETLEN (NUMCHANNELS * 2 + HEADERLEN + 1)

- #define SAMPFREQ 256 // ADC sampling rate 256
- #define TIMER2VAL (1024/(SAMPFREQ)) // Set 256Hz sampling frequency

Since we are using interruptions

- volatile unsigned char TXBuf[PACKETLEN]; //The transmission packet
- volatile unsigned char TXIndex; //Next byte to write in the transmission packet.
- volatile unsigned char CurrentCh; //Current channel being sampled.
- volatile unsigned char counter = 0; //Additional divider used to generate CAL_SIG
- volatile unsigned int ADC_Value = 0; //ADC current value

Setup

```
void setup() {

 noInterrupts();  // Disable all interrupts before initialization

 // LED1
 pinMode(LED1, OUTPUT);  //Setup LED1 direction
 digitalWrite(LED1,LOW); //Setup LED1 state
 pinMode(CAL_SIG, OUTPUT);

 //Write packet header and footer
 TXBuf[0] = 0xa5;    //Sync 0
 TXBuf[1] = 0x5a;    //Sync 1
 TXBuf[2] = 2;       //Protocol version
 TXBuf[3] = 0;       //Packet counter
 TXBuf[4] = 0x02;    //CH1 High Byte
 TXBuf[5] = 0x00;    //CH1 Low Byte
 TXBuf[6] = 0x02;    //CH2 High Byte
 TXBuf[7] = 0x00;    //CH2 Low Byte
 TXBuf[8] = 0x02;    //CH3 High Byte
 TXBuf[9] = 0x00;    //CH3 Low Byte
 TXBuf[10] = 0x02;   //CH4 High Byte
 TXBuf[11] = 0x00;   //CH4 Low Byte
 TXBuf[12] = 0x02;   //CH5 High Byte
 TXBuf[13] = 0x00;   //CH5 Low Byte
 TXBuf[14] = 0x02;   //CH6 High Byte
 TXBuf[15] = 0x00;   //CH6 Low Byte
 TXBuf[2 * NUMCHANNELS + HEADERLEN] =  0x01;  // Switches state

 // Timer2
 // Timer2 is used to setup the analag channels sampling frequency and packet update.
 // Whenever interrupt occures, the current read packet is sent to the PC
 // In addition the CAL_SIG is generated as well, so Timer1 is not required in this case!
 FlexiTimer2::set(TIMER2VAL, Timer2_Overflow_ISR);
 FlexiTimer2::start();

 // Serial Port
 Serial.begin(57600);
 //Set speed to 57600 bps

 // MCU sleep mode = idle.
 //outb(MCUCR,(inp(MCUCR) | (1<<SE)) & (~(1<<SM0) | ~(1<<SM1) | ~(1<<SM2)));

 interrupts();  // Enable all interrupts after initialization has been completed
}
```

Malarkey

Data structure, note from this:

- 6 channels
- 2 bytes per channel
- Position of the bytes are important, especially when you are receiving all this stuff

Set up timer interruption with required frequency → Now the function Timer2_Overflow_ISR will trigger 256 times a second

Sending the data to a computer via serial interface (remember the settings!)

What happens every interruption?

```
/*******************************************************/
/*  Function name: Timer2_Overflow_ISR                 */
/*  Parameters                                         */
/*    Input   :  No                                    */
/*    Output  :  No                                    */
/*    Action: Determines ADC sampling frequency.       */
/*******************************************************/
void Timer2_Overflow_ISR()
{
  // Toggle LED1 with ADC sampling frequency /2
  Toggle_LED1();

  //Read the 6 ADC inputs and store current values in Packet
  for(CurrentCh=0;CurrentCh<6;CurrentCh++){
    ADC_Value = analogRead(CurrentCh);
    TXBuf[((2*CurrentCh) + HEADERLEN)] = ((unsigned char)((ADC_Value & 0xFF00) >> 8));  // Write High Byte
    TXBuf[((2*CurrentCh) + HEADERLEN + 1)] = ((unsigned char)(ADC_Value & 0x00FF)); // Write Low Byte
  }

  // Send Packet
  for(TXIndex=0;TXIndex<17;TXIndex++){
    Serial.write(TXBuf[TXIndex]);
  }

  // Increment the packet counter
  TXBuf[3]++;

  // Generate the CAL_SIGnal
  counter++;    // increment the devider counter
  if(counter == 12){  // 250/12/2 = 10.4Hz ->Toggle frequency
    counter = 0;
    toggle_GAL_SIG(); // Generate CAL signal with frequ ~10Hz
  }
}
```

Malarkey

For each channel

Read the value

Convert to bytes

Send the structure over serial

Some maintenance/malarkey stuff to be in sync

That’s it! Now we need to

- Read the data into MATLAB in real time
- Analyse the data somehow → get the motor speed
- Send the control signal back to Arduino (motor speed)
- The rest you already know!

MLAB side of things: setting up

```
ard= serial('COM1','BaudRate',57600);

Fs=256; % set on ard code

Twindow =1; % number of seconds to have on screen at once
plotsize=Twindow*Fs;
chn_num=2;

time=(0:plotsize-1)/Fs;
data=zeros(chn_num,plotsize);

packetsize=17;
numread=20; %max 30 as 512 bytes inbuffer
```

```
%% Graph Stuff
plotGraph1 = plot(time,data(1,:),'-',...
    'LineWidth',2,...
    'MarkerFaceColor','w',...
    'MarkerSize',2);

title('EMG','FontSize',20);
xlabel('Time, seconds','FontSize',15);
ylabel('Voltage, V','FontSize',15);

drawnow
```

Open serial (baud should match with arduino)

Remember sampling frequency, we need it to compute time

Data array, where you store all data, we will use cycling buffer only storing last second (in this case 256 values), make everything zero in the beginning. 2 channels because why not?

This is the structure size, need to match what you send, right?

We will set up the graph now, only updating its data in future

Matlab side of things: reading data

```
%% Reading Data

iSample =1;

while ishandle(plotGraph1)

    if ard.BytesAvailable >= numread*packetsize

        for iRead = 1:numread

            [A,count] = fread(ard,packetsize,'uint8');

            data(1,iSample)=double(swapbytes(typecast(uint8(A(5:6)), 'uint16')));

            iSample = iSample +1;
            if iSample > plotsize
                iSample =1;
            end

        end


        try
          set(plotGraph1,'YData',data(1,:));
        catch
        end

        drawnow

        packetsleft=floor(ard.BytesAvailable/packetsize);

        if packetsleft > 20
            fprintf(2,'Update rate is too slow!: %d \n',packetsleft);
        end

    end

end

fclose(ard);
```

- iSample is a counter
- Do forever (if the graph is still there)
- If there is enough data to read in the buffer (we will read chunks of 20, cause why not? It is trade-off between speed and processing)
- Read the data into big data matrix
- Convert data from 2-byte to double prescision
- Increment counter, if more than 1 sec, start over again (circular buffer)
- Update the graph (there is try-catch to google)
- This is to check that you can process faster than you read (chunks of 20 works on my computer)
- This would occur if you close the graph window

That is it!

- Now you have a system to read and plot real-time EMG!!!
- Some possibilities to consider:
  - Read second channel

```
data(2,iSample)=double(swapbytes(typecast(uint8(A(7:8)), 'uint16')));
```

  - Real-time RMS calculation:
    - Setup

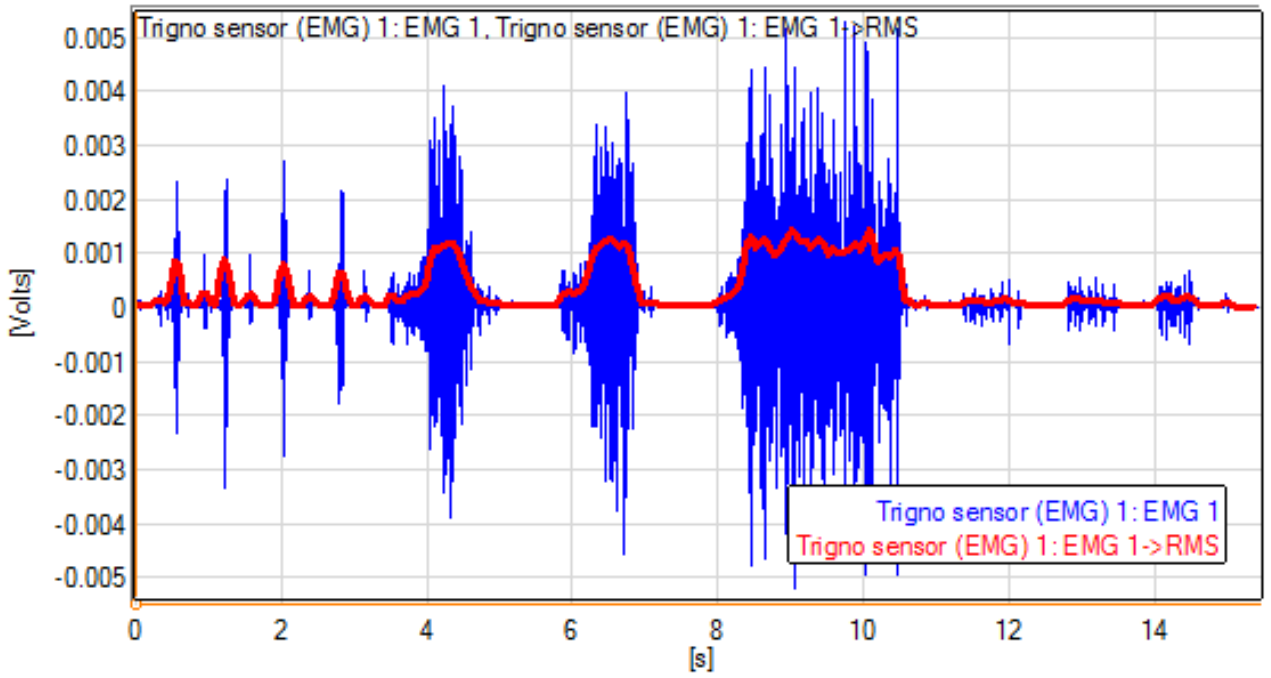
```
[bh,ah] = butter(3,30/(Fs/2),'high');
```

    - Inside the cycle:

```
if iSample > numread
        data(2,iSample) = std(filtfilt(bh,ah,data(1,iSample-numread:iSample)));
        data(2,iSample) = mean(data(2,iSample-numread:iSample));
else
        data(2,iSample) = std(filtfilt(bh,ah,data(1,[end-numread+iSample:end,1:iSample])));
        data(2,iSample) = mean(data(2,[end-numread+iSample:end,1:iSample]));
end
```

RMS computation

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}}$$



Measure of the spread (or power of the spread, If mean =0, RMS = STDEV)

Now, we have powerful vehicle to apply all fancy algorithms real-time to decide the motor actions

- Simple motor action:

RMS = SPEED

What is left to do

- 1) Communicate RMS back to Arduino
- 2) Copy-paste code for motor control
- 3) Make set-point value for speed = RMS of EMG (which you can read form serial)
- During this process you want to set maximal speed to maximal RMS, so makes sense to scale:
  - Desired_Motor_Speed = RMS*max_Speed/max_RMS

Summary

- Closed-loop speed controlled motor (finger), operated real-time by EMG muscle activity!

- Things to expand on:
  - Several fingers controlled by several sensors
  - Delivering second control loop via sending information about the motor speed to user real time
    - right now you can see it, so it is visual (do not try to crush someone's hand)
    - what about sense of touch?
  - What about position control (instead of velocity)?

Things to ask yourself

- Can you comfortably replace the motor to a linear actuator?
- Can you make the system to sense additional parameters (pressure when touch, temperature, etc)?
- Can you make the Matlab code which expands on these and incorporate more ideas?

If the answer is YES

- Then I have achieved my goal

Thank you for your attention!

Some graphic material used in the course was taken from publicly available online resources that do not contain references to the authors and any restrictions on material reproduction.

POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

OPEN
POLYTECH