SOFTWARE FOR ANALYZING EMG SIGNALS EMGLab

Viktor A. Nedialkov

Department of Radio Communications and Video Technologies, Faculty of Telecommunications,
Technical University of Sofia, 8 Kl. Ohridski Blvd, Sofia 1000, Bulgaria

Abstract

A software for analyzing EMG signals in real-time has been developed. The purpose of the software is to analyze the behavior of the human gait and help develop new algorithms for incorporation in active prosthetics.

The software is intended to work with the portable EMG multichannel recorder developed previously.

1. INTRODUCTION

In the process of development of an active EMG prosthesis a multichannel EMG processing device has been built. The device is intended to collect EMG signals from several muscles, amplify, digitize and send the data to PC via Bluetooth or USB link.

Below a general block scheme of the design is shown.

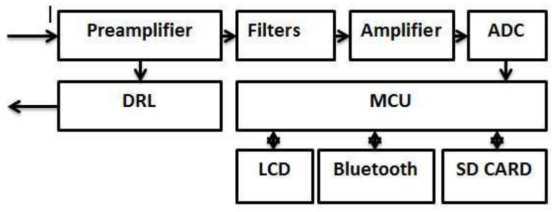

Fig. 1. (Block scheme of the device)

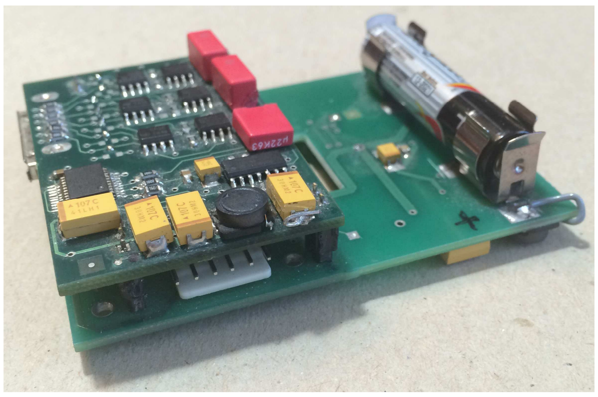

Fig. 2. The device prototype

A digital signal with Fs (discretization frequency) of 2000Hz per channel is produced by the device.

The raw EMG signal must be transformed to a signal more suitable for analysis of the behaviour of the muscles. An algorithm has been developed to detect muscle contraction with a possibility to assess the level of contraction in real-time [1].

The algorithm is based on the Pan-Tompkins QRS detection algorithm. It has four stages in signal processing. Bandpass filtering to remove unwanted noise. Differentiation stage to emphasize the high frequency signal. Integration stage to make the result positive and further emphasize the large differences. The moving average smooth the multiple peaks from the squaring operation.

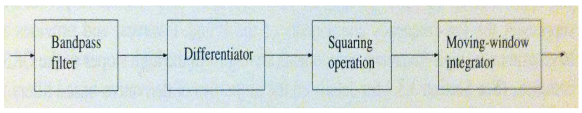

Fig. 3

For the development of the software Microsoft Visual Studio 2010 is used with development language Visual C++. The reason for that is we have already written similar code which will reduce the time for development.

The software has the following main stages

- Communication
- Signal processing
- Visualization

2. COMMUNICATION

Communication is achieved via virtual serial port. The data from the device is send after every discretisation period, which is set at 2000 Hz. The communication speed is 115200bps. Data is sent packed in the following protocol AAh, MSB1, LSB1, MSB2, LSB2.

Where AAh is a sync byte to select the start of a packet. The signal data is 12-bit long packed in two bytes.

The communication port is opened with the function

```
m_hCommPort=CreateFile(com_port,GENERIC_RE
AD | GENERIC_WRITE,0,0,OPEN_EXISTING,0,0);
```

which creates a handle to the port. Then we set the setting of the port when we first get the current settings, edit the variables we want and write the edited setting back:

```
::GetCommState(m_hCommPort,&dcb)

dcb.BaudRate=CBR_115200;

dcb.ByteSize=8;

dcb.Parity=0;

dcb.StopBits=ONESTOPBIT;

::SetCommState(m_hCommPort,&dcb)
```

The method chosen for receiving data is data pooling the communication port at a set period. For that we start a timer with a period 2ms with the following:

```
SetTimer(1,2,NULL);
```

Every 2 ms an event is triggered and event handler is called. The event handler calls the communication function. The communication function pools the serial port for available data and transfer the available data to the temporary buffer.

```
do

{

If(!::ReadFile(m_hCommPort,chBuffer,sizeof
(chBuffer),&bytesRead,0))

{

  ErrorCode=GetLastError();

}

for (int i=0;i<bytesRead;i++)

{

 mBuffer.push_back(chBuffer[i]);

 m_buf_lenght++;

}

while (bytesRead>0);
```

After that the function is searching for the sync byte and deletes all data until it finds it. If the first byte in the buffer is a sync byte the function sets a variable that there is available data.

```
double received_data =
(double)(256*mBuffer[1] + mBuffer[2]);

double received_data2=
(double)(256*mBuffer[3] + mBuffer[4]);

packet[0]= received_data;

packet2[0]= received_data2;
```

Communication is terminated with the functions

```
KillTimer(1);

CloseHandle(m_hCommPort);
```

Which closes the timer event and also closes the handle of the virtual COM port.

After a data byte is ready, it is processed by the signal processing stage of the program.

3. SIGNAL PROCESSING

The signal processing stage is divided by the following sub-stages

- Derivative function
- Squaring function
- Moving Average function
- Screen coordinated calculation

The derivative operator is calculated by the following lines:

```
double   CEmgLabView::Derivative(double   *
packet)

{

double derivative=0;

mDerBuffer.push_back(packet[0]-2048);

mDerBufferCounter++;

if (mDerBufferCounter>5)

{

derivative=(double)(2*mDerBuffer[5]+mDerBu
ffer[4]-mDerBuffer[2]-2*mDerBuffer[1]);

derivative=derivative*0.125;

mDerBuffer.erase(mDerBuffer.begin());

mDerBufferCounter--;

return derivative+2048;

}

else

{

 return derivative;

}

}
```

The last received data is passed to the function, corrected for polarity and pushed into buffer, from which the derivative is calculated.

The squaring is a simple square operation of the derivative result.

The result is then passed to the moving average calculation function

```
double CEmgLabView::MovingAverage(double * packet)
{
double ma_result=0;
int MA_SIZE=500;
mMABuffer.push_back(packet[2]);
mMABufferCounter++;
if (mMABufferCounter>=MA_SIZE)
{
for (int i=0;i<MA_SIZE;i++)
{
ma_result=ma_result+mMABuffer[i]/MA_SIZE;
}
mMABuffer.erase(mMABuffer.begin());
mMABufferCounter--;
return ma_result;
}
else
{
 return ma_result;
}
}
```

The calculated result from the squaring operation is inserted into a buffer, then a Moving Average with depth of MA_SIZE is calculated with the For operation.

The next step is to calculate the screen value of the calculated data

```
Int lead0=Round(mVertOffset+0*mVertRes/6
+mVertRes/12-packet[0]/mAmplitude
+2048/mAmplitude);
```

The Y- screen coordinates are calculated based on the screen resolution, offset and selected Amplitude scale.

4. VISUALIZATION

The discretisation frequency is 2000Hz and the serial port data pooling is set at 2ms. The calculation stage is performed for every data element of the signal. The screen calculation is also made for every data element but it is not possible to draw directly to the screen with 2000Hz draw frequency. That is why the process of visualization is divided into two stages – Draw to memory and copying of the memory buffer to the screen at a reasonable frequency.

We first create the memory screen buffers

```
m_pMemDC->CreateCompatibleDC(pDC);
m_pBitmap->CreateCompatibleBitmap
(pDC,ClientRect.right,ClientRect.bottom);
```

Another buffer is also used for the raster of the screen.

A timer is started with a period of 20 ms.

Every time an event is triggered the event handling function executes a copy of the memory buffer to the screen.

```
m_pMemDC->BitBlt(ClientRect.left,
ClientRect.top,ClientRect.right,ClientRect
.bottom,m_pMemDCRaster,0,0,SRCAND);
pDC->BitBlt(ClientRect.left,
ClientRect.top,ClientRect.right,ClientRect
.bottom,m_pMemDC,0,0,SRCCOPY);
```

The first line copies the screen raster to the memory buffer and the second line copies the memory buffer to the screen.

5. EXPERIMENTS

The software was tested also to verify the accuracy of the muscle contraction detection algorithm. Below are screenshots during testing

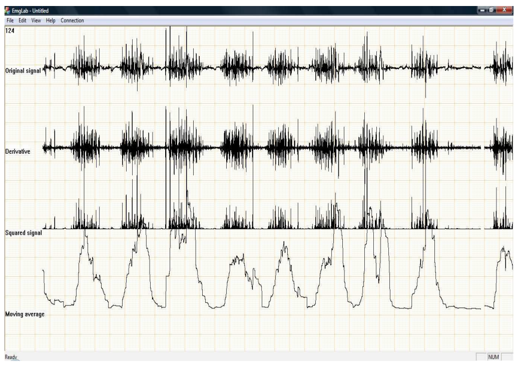

Fig. 4. Rectus femoris during walking

6. CONCLUSIONS

The software is working properly and is achieving its purpose to allow us to analyse and develop different algorithms for active prosthetics control. The real-time graphics is moving smoothly without glitches with the method of visualization chosen and the system is not loaded additionally by the program.

Further development of the software will include data storing and offline analysis of the stored signals.

Also an active prosthetics simulation module will be added.

References

[1] Pan J. and Tompkins W. J. “A real-time QRS detection algorithm.” IEEE Transaction on Biomedical Engineering, 32:230-236, 1985

[2] Rangaraj M. Rangayyan, “Biomedical Signal Analysis” Wiley-Interscience

[3] Joseph Bronzino, “The Biomedical Engineering Handbook” CRC Press

[4] Webster J, “Medical Instrumentation – application and design” John Wiley & Son.