BE6503 BioInstrumentation
Final Project: Real-time electrocardiography (ECG)
Spring 2016
Ernesto Akio Yoshimoto

Introduction:
The objective of this project is to measure heart rate in real time from subject electrocardiograph signal.

ECG acquisition system:
ECG signals are collected from electrodes located on the right arm (RA), left arm (LA) and right leg (RL) of a subject (figure 1). The signals are amplified by an instrumentation amplifier and then filtered by a NOTCH filter (figure 2) before analog to digital conversion (ADC) using Teensy 3.2 development board. The ADC digitalized signals are then sent to the computer via serial port and displayed with a graphical user interface (GUI).

Electrodes:
Electrodes are placed on the right arm (RA), left arm (LA) and right leg (RL) of the subject. They are connected through wires to the input of the instrumentation amplifier, Vra and Vla, and the output of the driven right leg circuit, Vrl (figure 1).

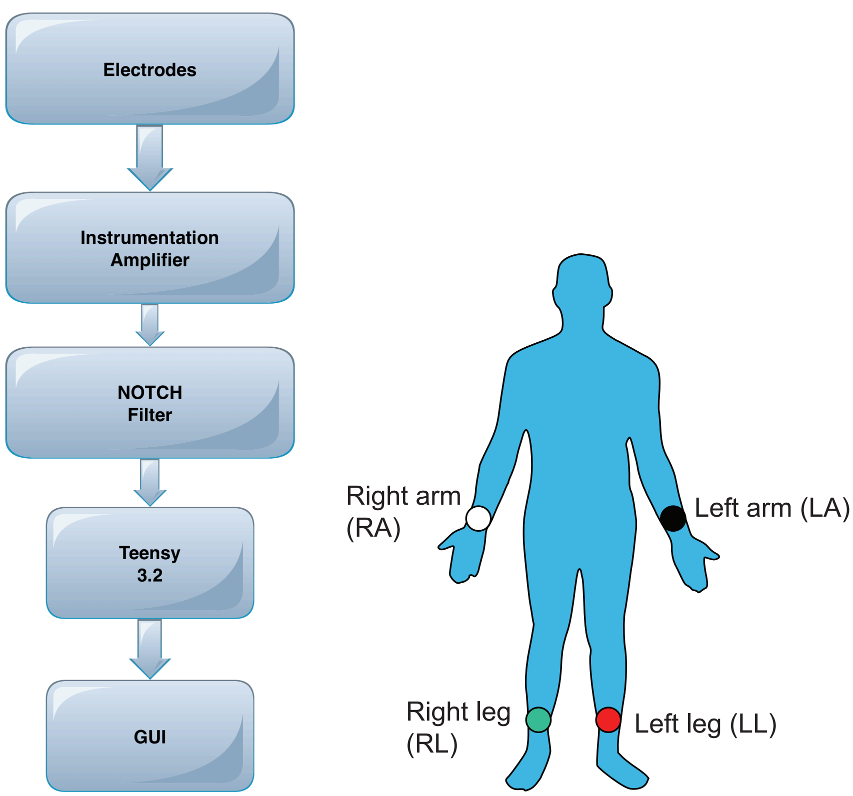

Figure 1. System diagram and electrode locations.

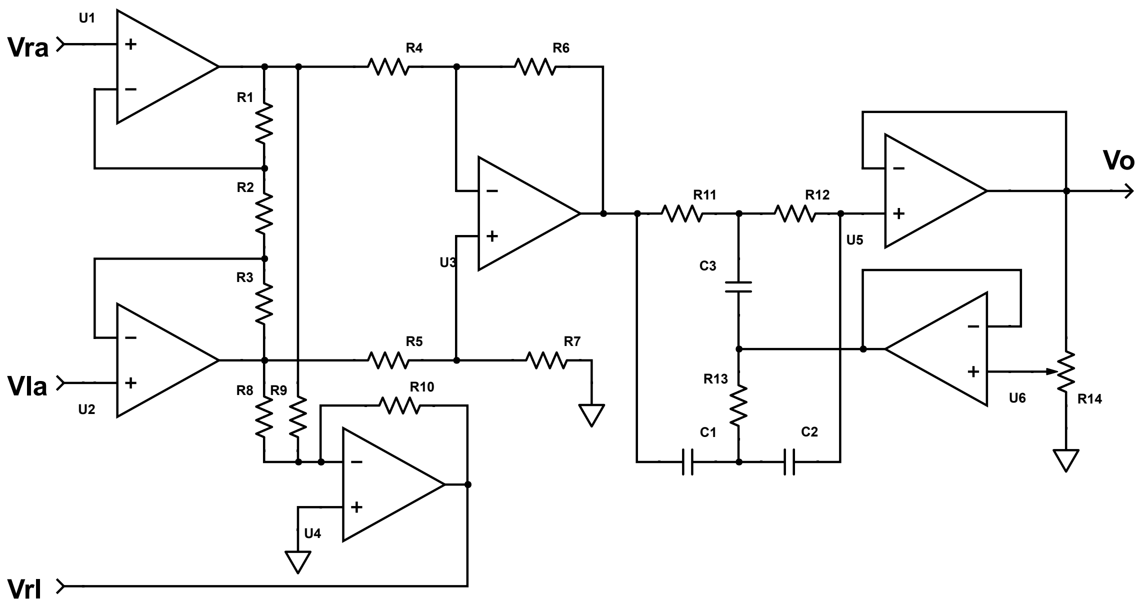

Figure 2. Instrumentation amplifier with driven right leg circuit and NOTCH filter.

Instrumentation amplifier:
The amplifier gain has been set in 1156 (61dB). The resistor values used in the instrumentation amplifier and the right driven leg circuit are in table 1.

Instrumentation amplifier

Resistor	Value (Ohms) ±20%
R1	39000
R2	220
R3	39000
R4	12000
R5	12000
R6	39000
R7	39000
Gain	1156

Right Driven Leg Circuit

Resistor	Value (kOhms) ±20%
R8	39
R9	39
R10	220

Table 1. Resistors used in instrumentation amplifier and right driven leg circuit.

NOTCH filter:
A second order NOTCH filter is used to diminish 60 Hz noise signal from power source. The element values are presented in table 2. The obtained cutoff frequency was 60.5 Hz.

NOTCH filter	
Cutoff Freq (Hz)	60.5
R11 (Ohm)	5600
R12 (Ohm)	5600
R13 (Ohm)	2800
C1 (uF)	0.47
C2 (uF)	0.47
C3 (uF)	0.94

Table 2. NOTCH filter capacitance and resistor values.

ADC:
The ADC in the Teensy 3.2 board (figure 3) samples the analog signal at 1 kHz and sends the digital values to a computer via serial port. The Arduino code (appendix A) stores the ADC data in a buffer of size 'numofblocks' and sends each buffer to the computer via serial port.

Serial Port:
The Teensy 3.2 serial port is set to work at 57,6 kbauds.

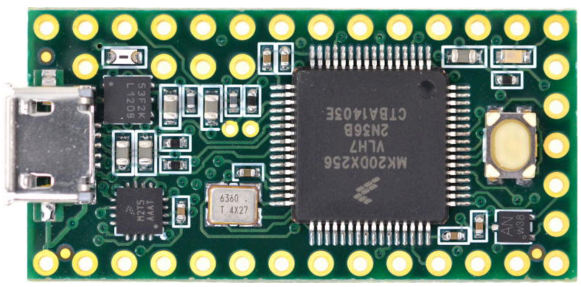

Figure 3. Teensy 3.2 development board.

GUI:
In the computer, a GUI has been implemented in Python 3.5 (figure 4). The GUI allows the visualization of the digitalized signal and its spectra (1st row), the filtered signal after a band-pass filter and it spectra (2nd row) and also the Pan and Tompkins peak finding algorithm and their locations (3rd row).
The bottom row of the GUI has buttons and boxes that allow the user to start/stop the data reading, clear the buffered data, select low-pass and high-pass filter orders and cutoff frequencies, the window size in seconds to be visualized and the heart beat per minutes.

Digital signal processing software:
The digital signals, sent by the hardware, are filtered to remove baseline and high frequency noise using programmable low-pass and high-pass filters (figure 4), in which the order and cutoff frequencies of the filters can be chosen by the user on the GUI. Then the signal is prepared to detect the R-peaks of the ECG by using the Pan and Tompkings' algorithm. The peak detection was performed using PeakUtils library.

The heart beat estimation is calculated by the inverse of the time distance between detected peaks multiplied by the ADC sampling rate and 60.

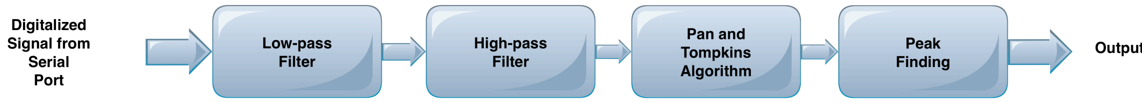

Figure 4. Digital signal processing implemented in Python.

ECG reading using the system:

Figure 5 shows an example of an ECG signal measured by the implemented system. The top left window shows the digitalized ECG signal sent by the Teensy 3.2 board; on the left, the respective spectra is shown. The second row of windows display the signal after applying a band-pass filter on the raw data (left), in this case the low and high cutoff frequencies are 1 and 30 Hz respectively, with a low-pass filter order of 5 and high-pass filter order of 1. The respective spectra is displayed on the right column and it can be seen the low and high frequency attenuations. The third row displays on the left the Pan and Tompkins algorithm output after applying a derivative, signal squared and window integration, and on the right side the peak locations for the detected R-peaks of the ECG. The estimated hear beat is shown on the bottom-left part of the GUI, which for the example is 114 heart beats per minute.

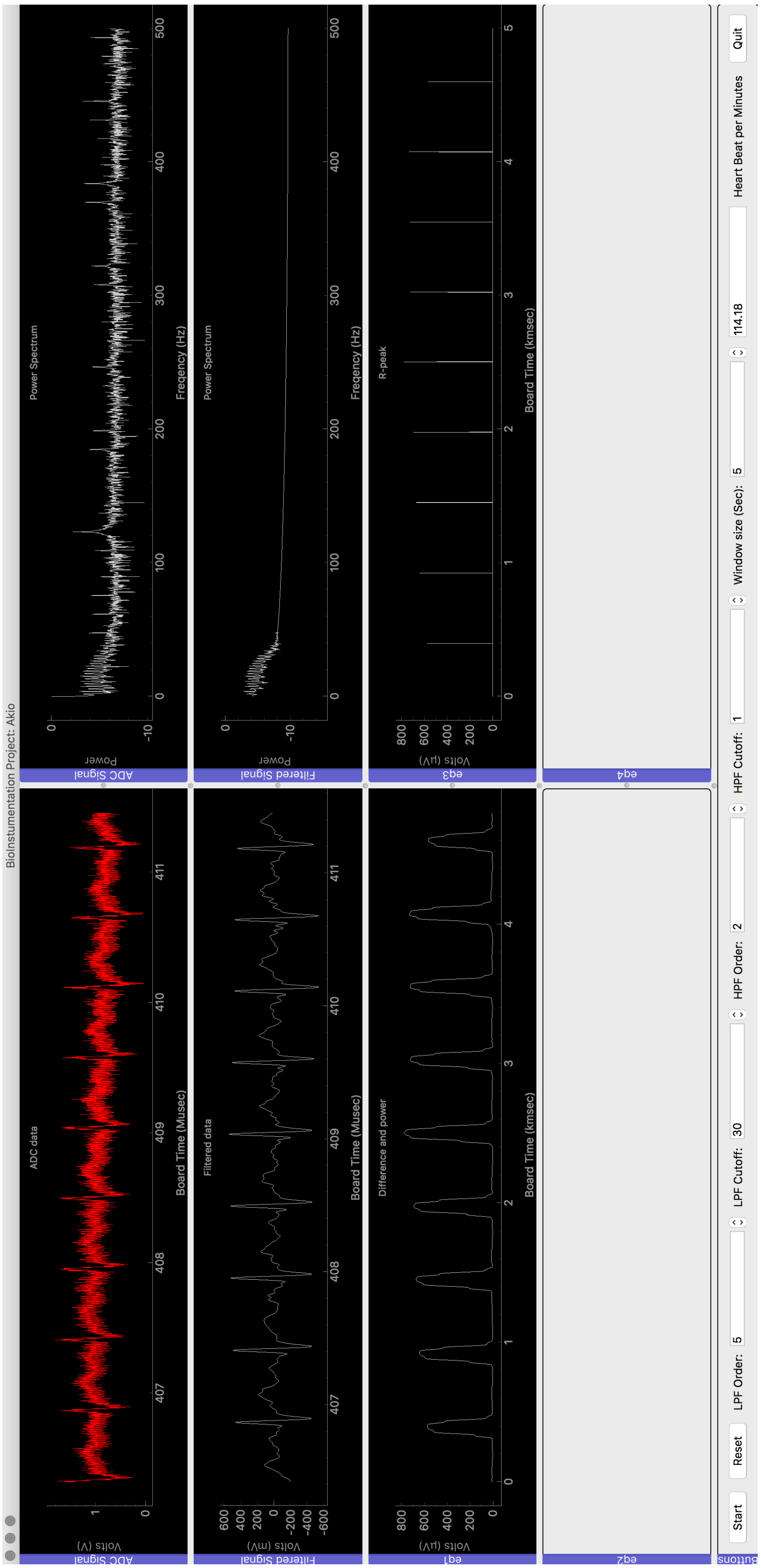

Figure 5. Graphical user interface in Python showing ECG signal, filtered signal and Pan and Tompkins algorithm output.

Appendix A: Arduino Code

```
// Arrays to save our results in
unsigned long values;
unsigned long time_m;
int numofblocks = 200;
int delay_const = 1000;
// Setup the serial port and pin 2
void setup() {
  // 300, 600, 1200, 2400, 4800, 9600, 14400,
  // 19200, 28800, 38400, 57600, or 115200
  Serial.begin(57600);
}
void loop() {
  for(int k=0;k<numofblocks;k++){
    time_m = micros();
    values = analogRead(A0);
    Serial.print(time_m);
    Serial.print("  ");
    Serial.print(values);
    Serial.print("  ");
    delayMicroseconds(delay_const);
  }
  Serial.print("\n");
}
```

Appendix B: Python Code

```
import numpy as np
import pyqtgraph as pg
from pyqtgraph.Qt import QtGui, QtCore
from pyqtgraph.dockarea import *
import serial
import scipy.signal as signal
import peakutils as pu

epsilon = np.finfo(float).eps
app = QtGui.QApplication([])
win = QtGui.QMainWindow()
# win = pg.GraphicsWindow(title="BioInstumentation Project: Akio")
area = DockArea()
win.setCentralWidget(area)
win.resize(1800, 950)
win.setWindowTitle('BioInstumentation Project: Akio')
# Create docks, place them into the window one at a time.
d11 = Dock("ADC Signal", size=(900, 300))
d12 = Dock("Filtered Signal", size=(900, 300))
```

```
d13 = Dock("Pan and Tompkins", size=(900, 300))
d15 = Dock("Buttons", size=(1800, 50))
d21 = Dock("ADC Signal", size=(900, 300))
d22 = Dock("Filtered Signal", size=(900, 300))
d23 = Dock("Peak Location", size=(900, 300))
area.addDock(d11, 'top')
area.addDock(d12, 'bottom', d11)
area.addDock(d13, 'bottom', d12)
area.addDock(d15, 'bottom', d13)

area.addDock(d21, 'right', d11)
area.addDock(d22, 'right', d12)
area.addDock(d23, 'right', d13)
# Widget plot p1:
p11 = pg.PlotWidget(title="ADC data")
p11.setLabel('left', "Volts", units='V')
p11.setLabel('bottom', "Board Time", units='usec')
p11.enableAutoRange('x', True)
p11.disableAutoRange('y')
p11.setYRange(0, 4)
curve11 = p11.plot(pen="r")
d11.addWidget(p11)
# Widget plot p3:
p21 = pg.PlotWidget(title="Power Spectrum")
p21.setLabel('left', "Power")
p21.setLabel('bottom', "Freqency", units='Hz')
p21.disableAutoRange('y')
p21.setYRange(-10, 0)
curve21 = p21.plot(pen="r")
d21.addWidget(p21)
# Widget plot p2:
p12 = pg.PlotWidget(title="Filtered data")
p12.setLabel('left', "Volts", units='V')
p12.setLabel('bottom', "Board Time", units='usec')
p12.enableAutoRange('x', True)
curve12 = p12.plot(pen="g")
d12.addWidget(p12)
# Widget plot p4:
p22 = pg.PlotWidget(title="Power Spectrum")
p22.setLabel('left', "Power")
p22.setLabel('bottom', "Freqency", units='Hz')
p22.disableAutoRange('y')
p22.setYRange(-15, 0)
curve22 = p22.plot(pen="g")
d22.addWidget(p22)
# Widget plot p2:
```

```
p13 = pg.PlotWidget(title="Difference, Squared and Integral")
p13.setLabel('left', "Volts", units='V')
p13.setLabel('bottom', "Board Time", units='msec')
p13.enableAutoRange('x', True)
curve13 = p13.plot(pen="c")
d13.addWidget(p13)
# Widget plot p2:
p23 = pg.PlotWidget(title="R-peak")
p23.setLabel('left', "Volts", units='V')
p23.setLabel('bottom', "Board Time", units='msec')
p23.enableAutoRange('x', True)
curve23 = p23.plot(pen="c")
d23.addWidget(p23)
# Widget for buttons:
p15 = pg.LayoutWidget()
B1 = QtGui.QPushButton('Start')
B2 = QtGui.QPushButton('Reset')
spinbox1 = pg.SpinBox(value=5, bounds=[1, 10], step=1)
spinbox12 = QtGui.QLabel("LPF Order:")
spinbox2 = pg.SpinBox(value=30, bounds=[5, 100], step=1)
spinbox22 = QtGui.QLabel("LPF Cutoff:")
spinbox3 = pg.SpinBox(value=2, bounds=[1, 6], step=1)
spinbox32 = QtGui.QLabel("HPF Order:")
spinbox4 = pg.SpinBox(value=1, bounds=[0.1, 10], step=0.1)
spinbox42 = QtGui.QLabel("HPF Cutoff:")
spinbox5 = pg.SpinBox(value=100, bounds=[10, 1000], step=1)
spinbox52 = QtGui.QLabel("Integral Window size:")
spinbox6 = pg.SpinBox(value=5, bounds=[2, 60], step=1)
spinbox62 = QtGui.QLabel("Window size (Sec):")
TBox1 = QtGui.QLineEdit()
TBox12 = QtGui.QLabel("Heart Beat per Minutes")
B100 = QtGui.QPushButton('Quit')
p15.addWidget(B1)
p15.addWidget(B2)
p15.addWidget(spinbox12)
p15.addWidget(spinbox1)
p15.addWidget(spinbox22)
p15.addWidget(spinbox2)
p15.addWidget(spinbox32)
p15.addWidget(spinbox3)
p15.addWidget(spinbox42)
p15.addWidget(spinbox4)
p15.addWidget(spinbox52)
p15.addWidget(spinbox5)
p15.addWidget(spinbox62)
p15.addWidget(spinbox6)
```

```
p15.addWidget(TBox1)
p15.addWidget(TBox12)
p15.addWidget(B100)
d15.addWidget(p15)
start_stop = False
# Show Window
win.show()
# Serial port: OSX
raw = serial.Serial('/dev/cu.usbmodem1507501', 57600)
data0 = []
data1 = []

def update():
    global curve11, curve12, curve13, curve21, curve22, curve23
    global data0, data1
    numofblocks = 200  # Number of blocks in a package
    rate = 1000  # sampling rate in HZ
    window_size = int(spinbox6.value())  # Window size in seconds
    integral_win = int(spinbox5.value())
if start_stop:
        xaxissize = window_size * rate
        line = raw.readline()
        line_val = [float(val) for val in line.split()]
        if len(line_val) == 2 * numofblocks:
            for k in range(0, numofblocks):
                line_val[2 * k] = line_val[2 * k]
                line_val[2 * k + 1] = 3.3 * (line_val[2 * k + 1] - 1) / 1023
            if (not data0) | (len(data0) < xaxissize):
                for k in range(0, numofblocks):
                    data0.append(int(line_val[2 * k]))
                    data1.append(line_val[2 * k + 1])
            else:
                for k in range(0, numofblocks):
                    data0.pop(0)
                    data1.pop(0)
                    data0.append(int(line_val[2 * k]))
                    data1.append(line_val[2 * k + 1])
            xdata = np.array(data0, dtype='int')
            ydata = np.array(data1, dtype='float')
            curve11.setData(xdata, ydata)
            ydata_filt = low_pass_filter(ydata, spinbox2.value(), rate,
spinbox1.value())
            ydata_filt = high_pass_filter(ydata_filt, spinbox4.value(), rate,
spinbox3.value())
            curve12.setData(xdata[integral_win:], ydata_filt[integral_win:])
            # ptr += 1
```

```
        app.processEvents()
        if len(xdata) > 2*integral_win:
            tdata = range(0, xaxissize)
            fx, fy = fouriertransform(tdata, ydata)
            curve21.setData(fx, fy)
            fx, fy = fouriertransform(tdata, ydata_filt)
            curve22.setData(fx, fy)
            ydata_peak = find_peak(ydata_filt, integral_win)
            curve13.setData(ydata_peak)
            ydata_max = np.zeros(xaxissize)
            indexes = pu.indexes(ydata_peak, thres=0.5, min_dist=100)
            ydata_max[indexes] = ydata_peak[indexes]
            curve23.setData(tdata, ydata_max)
            # print(np.diff(indexes))
            heart_beat = 60*rate/np.mean(np.diff(indexes))
            TBox1.setText(str(np.round(heart_beat,decimals=2)))

timer = QtCore.QTimer()
timer.timeout.connect(update)
timer.start(0)

def find_peak(x, n):
    y = np.diff(x)
    y = np.power(y, 2)
    y = np.cumsum(np.insert(y, 0, 0))
    y = (y[n:] - y[:-n]) / n
    y = y[100:]
    return y

def low_pass_filter(x, hf, rt, order):
    lowpass = signal.butter(order, hf/(rt/2.0), 'low')
    y = signal.filtfilt(*lowpass, x)
    return y

def high_pass_filter(x, lf, rt, order):
    highpass = signal.butter(order, lf/(rt/2.0), 'high')
    y = signal.filtfilt(*highpass, x)
    return y

def fouriertransform(x, y):
    # Perform fourier transform. If x values are not sampled uniformly,
    # then use np.interp to resample before taking fft.
    dx = np.diff(x)
    uniform = not np.any(np.abs(dx - dx[0]) > (abs(dx[0]) / 1000.))
    if not uniform:
        x2 = np.linspace(x[0], x[-1], len(x))
```

```
        y = np.interp(x2, x, y)
        x = x2
    f = np.abs(np.fft.fft(y) / len(y)) ** 2
    y = np.log10(f[0:len(f) / 2] + epsilon)
    dt = (x[-1] - x[0]) / 1000
    x = np.linspace(0, 0.5 * len(x) / dt, len(y))
    return x, y

def start_scope():
    global start_stop
    if start_stop:
        start_stop = False
        B1.setText("Start")
    else:
        start_stop = True
        B1.setText("Stop")

B1.clicked.connect(start_scope)

def reset_scope():
    global data0, data1
    data0 = []
    data1 = []

B2.clicked.connect(reset_scope)
spinbox5.valueChanged.connect(reset_scope)

def close_scope():
    raw.close()
    exit()

B100.clicked.connect(close_scope)

def clear_data():
    global xdata, ydata
    xdata = []
    ydata = []

spinbox5.sigValueChanging.connect(clear_data)

if __name__ == '__main__':
    import sys
    if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
        QtGui.QApplication.instance().exec_()
```