ANALOG DEVICES

AN-900
应用笔记

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

通过单个加速度计增强计步器的性能

作者：Jim Scarlett

AN-602应用笔记考察了通过ADI公司的加速度计制作简单但相对精确的计步器的方法。然而，新款器件已经上市，使加速度计可以用于更具成本敏感度的应用中。如此一来，计步器一类应用在手机等许多消费设备中日渐盛行。

鉴于这一趋势，我们对采用单个加速度计的计步器进行了更加细致的考察。采用了AN-602中用到的技术，以便复现其结果。虽然算法运行顺畅，却未能重现相同的精度。特别地，人与人之间的差异远远大于预期，即使是同一个人也可能使用不同的步速和步长。结果促使我们开始考虑能否对AN-602所用算法进行改进。

测试时采用了两个不同的计步器测试板，各使用一个ADuC7020 ARM7®控制器。一种设置结合了ADuC7020微控制器和ADXL322加速度计评估板，另加一台16 × 2字符液晶显示屏。另一种设置则为一块定制板，采用ADuC7020和ADXL330 3轴加速度计，同样配了一台16 × 2字符液晶显示屏。对于定制板示意图，请见图5。

AN-602中的方法

AN-602方法的内在机制基于以下原理：个人步伐的垂直弹跳与此人的步长直接相关(见图1)。

由于α角和θ角相等，因而可证明步长为最大垂直位移的倍数。个人腿长差异也考虑了进去，因为在角度不变的情况下，垂直位移的大小将随身高而变化。

但是，用加速度计求出的是加速度的变化而不是位移变化。这些加速度测量值必须先转换为距离，然后才能使用。鉴于AN-602设置的计算能力有限，我们用一个简单的公式来近似模拟转换需要的二重积分。由于本实验有充足的处理能力，我们试图直接进行离散积分运算。

结果选择了一种简便方法来计算积分。确定每一步后，将该步的全部加速度样本相加，以获得一个速度样本集合。再对每步的速度样本进行归一化处理，以使最终样本为0。再将这些值相加，得到位移值。

起初，这种方法看似成功几率极大，因为对于在过道上行走多次的受试者来说，测量所得距离是相对一致的。然而，人与人之间的差异问题非常突出，同一受试者以不同步速行走时的差异问题也是如此。结果，我们开始研究问题是否出在模型自身上。

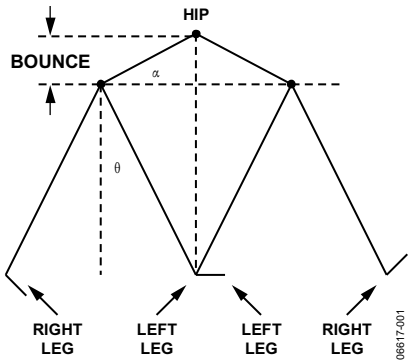

图1 行走时的臀部垂直运动

了解模型

这个模型有两个主要条件。首先，它假定脚实际上只在一点上触地。第二，它假定每只脚对地的冲击都具有完全弹性。当然，这两个条件没有一个是正确的。于是我们提出了这是否能解释实验中遇到的较大差异的问题。根据该实验，完全可以说这种情况能够解释大部分差异。

为便于理解，可考察对多步测得的加速度值，如图2所示。人们步伐中不同的弹力源如数据所示。

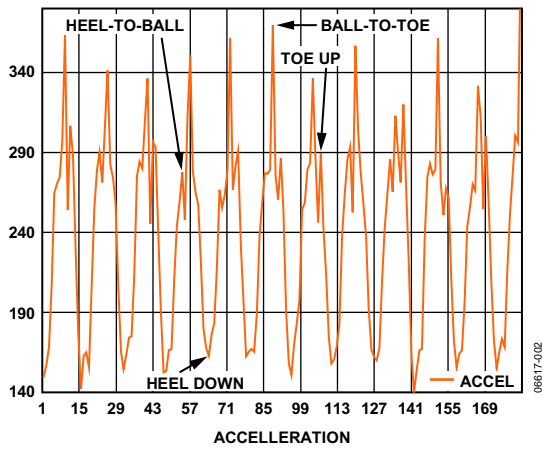

图2 受试者1以正常步速行走时的加速度曲线图

图2展示了在将加速度测量值转换为精确的距离值时遇到的问题。考虑峰峰值变化(或者，甚至对数据求积分)的方法在这类数据上会陷入困境。造成这种困难的原因在于不同人步伐的弹性差异，或者同一个人因在各次测量中使用不同步速而造成的步伐差异。

图3所示为同一受试者以更长、更快的步幅行走时的情况。峰峰加速度差变大，各弹力点也显现出差异。可见，与图2相比，代表弹性数据的数量与代表实际数据的数量存在差异。但是，算法只考虑了加速度测量值集合，却未注意到进行测量的具体环境。因此，问题是如何消除弹力对受试者步伐的影响而又不排除有用数据。

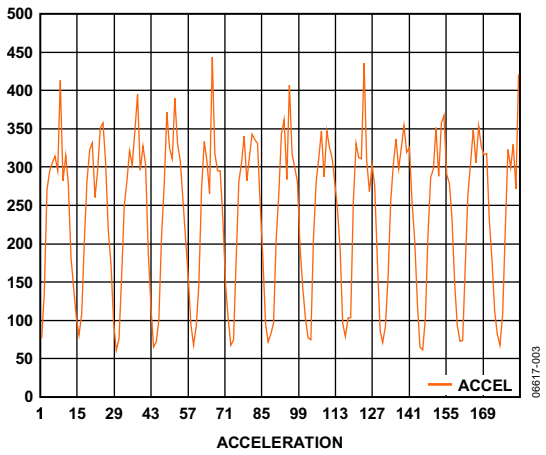

图3 受试者1以较快步速行走时的加速度曲线图

两个坐标图之间存在一些重要差异。在图3中，每步曲线的底部较窄，顶部更一致(特殊峰更少)。与最小和最大样本值相比，这些差异加大了样本的平均值。

为便于比较，请查看图4中关于另一受试者的数据坐标图。步长与图2中的受试者1非常相似，但数据本身却看起来非常不同。

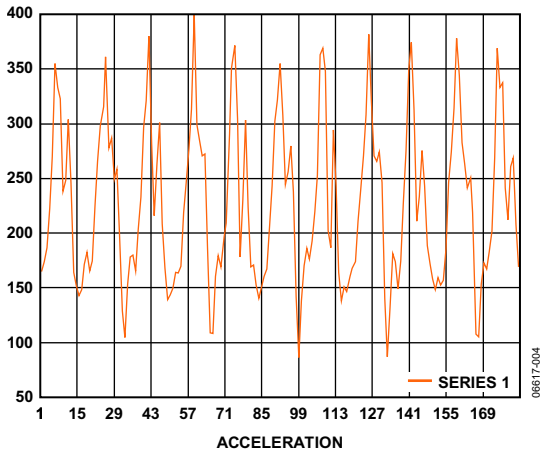

图4 受试者2以正常步速行走时的加速度曲线图

受试者2步幅中的弹力远远超过受试者1(如图2所示)。但两组数据基本代表相同的行走距离。仅仅依靠峰值来计算距离会导致差异巨大的结果。使用简单的二重积分法面临同样的问题。

解决弹力问题

试图通过直接计算妥善解决这个问题的所有方法都面临相同的问题。结果，我们多次尝试对数据进行归一化处理，以消除弹力，但这些尝试均以失败告终。主要原因似乎是，这些方法都需要了解数据的采集环境。但在实际使用中，系统并不了解外部情况，有的只是数据点。解决方案必须具备仅依靠数据而无需了解外部情况就能进行运算的能力。

这个问题的一种可能解决方案渐渐浮出水面。对于从较慢步速换为较快步速时数据的变化方式，前文已有论述。在步幅较长、较快时，因弹力引起的显著差异较少。结果，相对于最小和最大数据，样本点的平均值加大。如图4所示，由于步伐中的弹量较多，很难直接确定这一点。但计算结果显示，平均值与峰值和图2极为相似。因此，我们采用一种简便的可能算法来确定行走的距离。即

$$d = k \times \frac{(avg - min)}{(max - min)} \quad (1)$$

其中：
*d*为求出的距离。
*k*为一常数乘数。
*max*为该步中测得的最大加速度值。
*min*为最小加速度值。
*avg*为该步的平均加速度值。

用等式1求出每步的距离，而步伐则由一种不同的求步算法算出。求步算法采用一个8点移动平均数，以使数据变得平滑。算法先找最大峰值，然后找最小峰值。当移动平均数越过0点时，则算一步，0是每步的总平均数。距离算法中用到的数据考虑了移动平均数的4点延迟。

受试者1采用不同步长时，这种简单的解决方案均能有效应付。对其他受试者也有不错的表现。但有些受试者产生的距离与小组平均距离之差高达10%。超出了非校准测量±7.5%的目标误差范围。需要另一种解决方案。

上次测试所用比例似乎反映了不同受试者步伐中的弹力差异。因此可以将此处考察的两种方法结合起来。先回到用二重积分的最初想法，该比例充当校正系数，用以消除计算中的弹力数据。故有以下公式

$$d = k \times \sqrt{\frac{(max - min)}{(avg - min)} * \sum\sum(accel - avg)} \quad (2)$$

其中，*accel*表示相关步伐的所有测得加速度值。

这种算法可有效处理多个受试者和多种步速，所有差异在6%左右的范围内波动。这种算法下，通过调整乘数k，即可轻松校正具体个人/步速。列出的代码中也含有求步长平均值的函数，以消除步间差异。此处所用结果未经这种平均算法处理。

在本实验中，只使用了X轴和Y轴。出于灵活性考虑，选择了3轴加速度计，但结果发现，2轴加速度计足以胜任。可用ADXL323代替ADXL330。两种情况下可使用同一种布局，因为除Z轴输出以外，二者的引脚配置完全相同。

这些实验的重点是在计步器的距离测量中取得好的结果。除确保计数算法能在行走(奔跑)时正常运行以外，并未对记步的算法作出更深入的研究。在测量的数百步中，测得步数与实际步数的误差不超过1至2步。但是，记步的简单算法是可能被非行走运动愚弄的。这方面还可以进行改进，比如AN-602中描述的时间窗函数。其目的是忽略预期时间窗之外的步子，同时保留在受试者改变步速时的自适应能力。

小结

本应用笔记对一组实验结果进行了简要描述，实验旨在使采用单个加速度计的简单计步器产生良好的性能。考察了面临的一些障碍。最终结果达到了规定的精度目标，还有可能通过校准提高精度。尽管可通过更复杂的系统(比如，采用多个加速度计)取得最高的精度，但本应用笔记提供的算法构成了简单的低成本应用的起点。

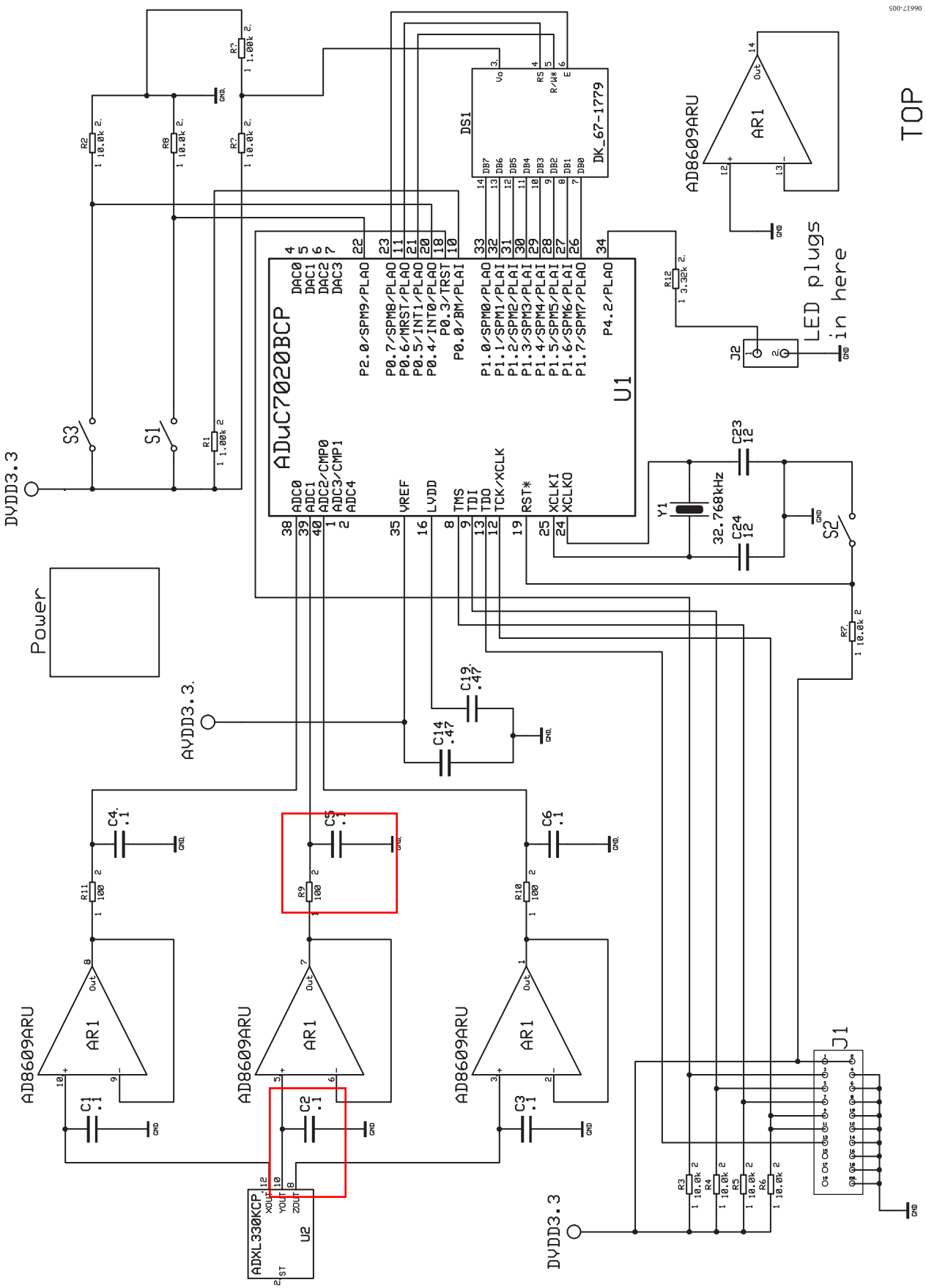
TOP
ADuC7020BCP
U1
DS1
DK_67-1779
AD8609ARU
AR1
ADXL330KCP
U2
J1
J2
LED plugs in here
Power
DVDD3.3
AVDD3.3
32.768kHz
Y1
S1
S2
S3

图5. 定制板示意图

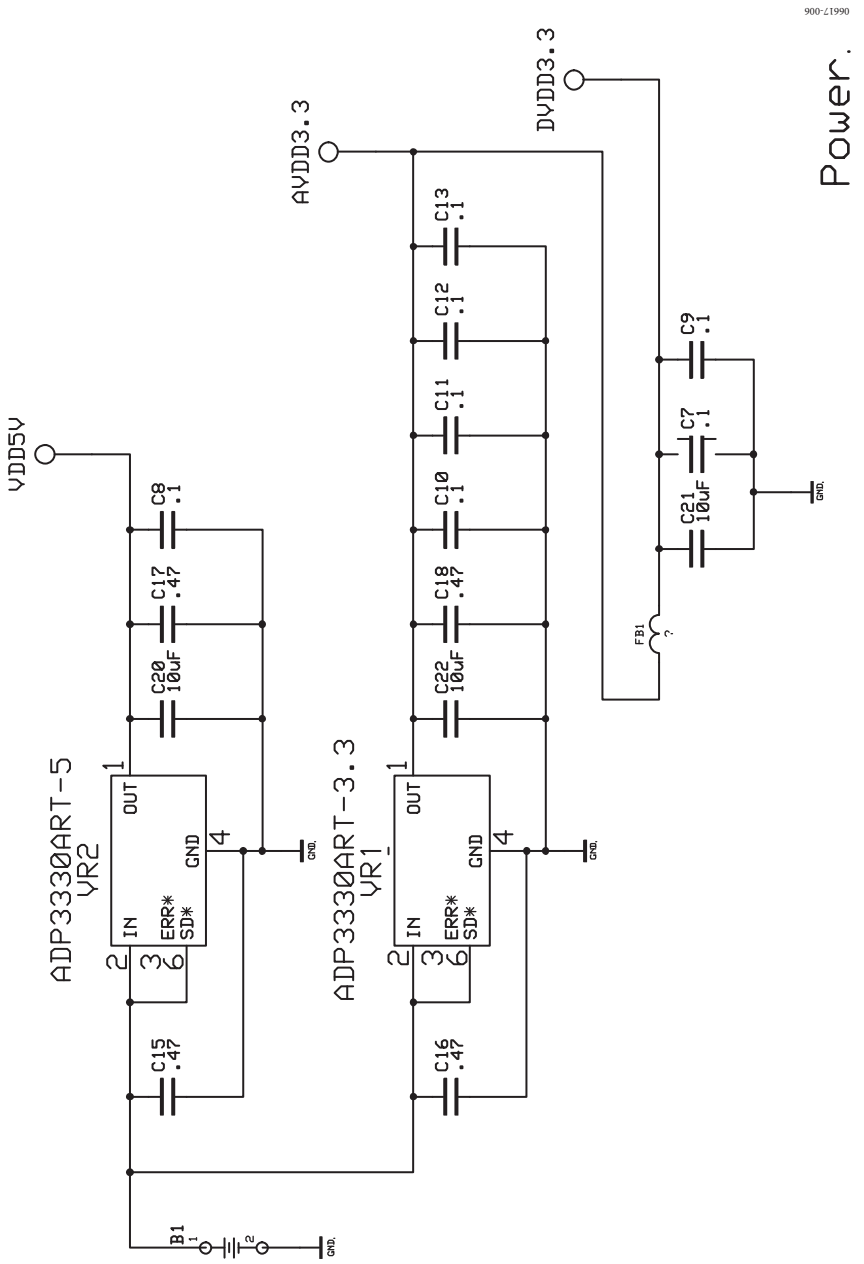
Power.
DVDD3.3
AVDD3.3
VDD5V
C13 .1
C12 .1
C11 .1
C10 .1
C18 .47
C22 10uF
C9 .1
C7 .1
C21 10uF
FB1
C8 .1
C17 .47
C20 10uF
ADP3330ART-5
VR2
ADP3330ART-3.3
VR1
OUT
GND
IN
ERR*
SD*
C15 .47
C16 .47
B1
GND

图6. 定制板示意图(续)

ADuC7020 C代码

以下三个文件所含为用于在采用ADuC7020微控制器的测试板上部署该算法的代码。

MAIN.C

*main.c*文件用于器件的初始化：

```
/*****************************************************************************
 Author        : J Scarlett
 Date          : Nov 2006
 Files          : main.c, display.c, ped.c
 Hardware      : ADuC7020
 Description   : Implements a simple pedometer based on application note AN-602

*****************************************************************************/
#include <ioaduc7020.h>

//       Function Prototype with required function attribute.
extern void Monitor_Function(void);
extern void Display_Init(void);

//*****************************************************************************
// Main Function for ADuC7020 Pedometer essentially performs startup functions
//*****************************************************************************
int main (void)
{
         POWKEY1 = 0x01;
         POWCON = 0x06;                                          // set to 653kHz core clock
         POWKEY2 = 0xF4;
         REFCON = 0x2;                                           // use external reference
                                                                 // (connected to VDD)
//*****************************************************************************
//       Initialize Peripherals
//*****************************************************************************
// GPIO Configuration
         GP4DAT = 0x04000000;                                    // P4.2 configured as an output.
                                                                 // LED is turned on
         GP0CON = 0x00000000;                                    //
         GP0DAT = 0xE0000000;                                    // 0.7, 0.6, and 0.5 are outputs
                                                                 // 0.7 = E, 0.5 = R/W*, 0.6 = RS
         GP1DAT = 0xFF000000;                                    // All P1 pins are outputs

         ADCCON = 0x20;                                          // Turn ADC on but do not enable

         Display_Init();                                         // found in File display.c
         Monitor_Function();                                     // found in File ped.c
//*****************************************************************************
// Main Loop
//*****************************************************************************
         while(1)
         {
         }
} // main()
```

DISPLAY.C

*display.c*文件含有访问显示屏的所有函数：

```
// File "display.c"
// performs all LCD display interface functions

#include <ioaduc7020.h>

extern char stepbcd[6];                 // found in File ped.c
extern char distbcd[6];                 // this too

void Display_Init(void);
void display_data(void);
void display_data_clear(void);
void char_load(int RS, int data);
void delay(unsigned int cycles);
int reverse_data(int data);

void Display_Init()
{
                                        // used once to initialize display and write
                                        // the "Steps" and "Distance" headers
        int rs, data;

        // Display initialization
        rs = 0;                                         // no RAM access yet

        data = 0x30;                                    // function set:  2-line mode, display off
        char_load(rs, data);

        data = 0x38;                                    // function set:  2-line mode, display off
        char_load(rs, data);

        data = 0x08;                                    // display off, cursor off, blink off
        char_load(rs, data);

        data = 0x01;                                    // clear display
        char_load(rs, data);
        delay(49);                                      // ~1.5 ms additional delay is required

        data = 0x06;                                    // increment mode, no shift
        char_load(rs, data);

        data = 0x0C;                                    // display on, cursor off, blink off
        char_load(rs, data);

        data = 0x80;                                    // set data address to home, just to be sure
        char_load(rs, data);

        rs = 1;                                         // now writing to data RAM
        data = 0x53;                                    // start of sequence to send
        char_load(rs, data);                            // "Steps   Distance" for title line

        data = 0x74;                                    // "t"
```

```
        char_load(rs, data);

        data = 0x65;                                    // "e"
        char_load(rs, data);

        data = 0x70;                                    // "p"
        char_load(rs, data);

        data = 0x73;                                    // "s"
        char_load(rs, data);

        data = 0x20;                                    // " " times 3
        char_load(rs, data);
        char_load(rs, data);
        char_load(rs, data);

        data = 0x44;                                    // "D"
        char_load(rs, data);

        data = 0x69;                                    // "i"
        char_load(rs, data);

        data = 0x73;                                    // "s"
        char_load(rs, data);

        data = 0x74;                                    // "t"
        char_load(rs, data);

        data = 0x61;                                    // "a"
        char_load(rs, data);

        data = 0x6E;                                    // "n"
        char_load(rs, data);

        data = 0x63;                                    // "c"
        char_load(rs, data);

        data = 0x65;                                    // "e"
        char_load(rs, data);

} // Display_Init()


void display_data()
{
                                                        // displays the data contained in stepbcd[] & distbcd[]
                                                        // beginning at the first and ninth characters
                                                        // on row 2 of the display
        int i, rs, data, zero;

        rs = 0;                                         // want to set address, not data
        data = 0xC0;                                    // start of second line
        char_load(rs, data);
```

```
        rs = 1;
        zero = 0;
        for (i=5; i>=0; i--)                                    // display steps
        {
                if ((stepbcd[i] > 0) || (zero ==1))             // suppress leading zeroes,
                {                                               // but not embedded zeroes
                        zero = 1;
                        data = 0x30 | stepbcd[i];               // numbers on display character table
                                                                // begin at 0x30
                        char_load (rs, data);
                } // if
        } // for

        rs = 0;                                                 // set address
        data = 0xC8;                                            // ninth character of second line
        char_load(rs, data);

        rs = 1;
        zero = 0;
        for (i=5; i>=0; i--)                                    // display distance
        {
                if ((distbcd[i] > 0) || (zero ==1))
                {
                        zero = 1;
                        data = 0x30 | distbcd[i];
                        char_load (rs, data);
                } // if
        } // for

} // display_data()


void display_data_clear(void)
{
                                                                // used to clear display data field
                                                                // before new measurement
        int i, rs, data;

        rs = 0;                                                 // want to set address
        data = 0xC0;                                            // start of second line
        char_load(rs, data);

        rs = 1;
        data =0x20;
        for (i=0; i<16; i++)                                    // put spaces across Row 2
                char_load (rs, data);

} // display_data_clear()


void char_load(int rs, int data)
{
```

```
        // signal timing assumes a core clock < 4MHz
        // delay at end is ~61us, to allow write to complete

        data = reverse_data(data);                              // board layout dictated reversing MSB/LSB
        GP1CLR = 0x00FF0000;                                    // ensure clean slate for next character
        GP1SET = data << 16;                                    // set Port 1 to new character data

        if (rs)
                GP0SET = 0x00400000;                            // RS bit = 1
        else
                GP0CLR = 0x00400000;                            // RS bit = 0

        GP0CLR = 0x00200000;                                    // WR bit = 0 (this is a write command)

        GP0SET = 0x00800000;                                    // set E bit to begin transfer process
        GP0CLR = 0x00800000;                                    // clear E bit to complete transfer process

        delay(2);

} // char_load()

void delay(unsigned int cycles)
{
        T2CON = 0;
        T2CLRI = 0;
        T2LD = cycles;
        T2CON = 0xC0;                                           // enable Timer 2, periodic mode, 32.768 kHz
        while (!(IRQSIG & WAKEUP_TIMER_BIT));                   // wait for timeout
        T2CON = 0;                                              // disable Timer 2

} // delay()

int reverse_data(int data)
{
        int i, temp;

        temp = 0;

        for (i=0; i<4; i++)
        {
                temp |= (((0x01 << i) & data) << (7 - (2 * i)));
        } // fill top 4 bits of temp

        for (i=4; i<8; i++)
        {
                temp |= (((0x01 << i) & data) >> ((2 * i) - 7));
        } // fill bottom 4 bits of temp

        return temp;

} // reverse_data()
```

PED.C

*ped.c*文件含有计步器算法中用到的函数：

```
// file "ped.c"
// performs pedometer and misc functions

#include <ioaduc7020.h>
#include <math.h>
#include <stdlib.h>

//      Function Prototype & variables
char stepbcd[6];
char distbcd[6];
char stepflag;
float stride, avgstride, accel_dat[50];
float maxavg, minavg, accel_avg, velocity, displace;
float distance;
int steps;

void Monitor_Function(void);
void get_sample(unsigned int *xdat, unsigned int *ydat, unsigned int *zdat);
char IsStep(float avg, float oldavg);
void display_prep(void);
long int bin_to_bcd(long int bin_no);
extern void display_data(void);                         // found in File display.c
extern void display_data_clear(void);                   // found in File display.c

// functions

void Monitor_Function()
{
        char flag;
        unsigned int xdat, ydat, zdat;
        int i, cycle_count, tot_samples, avgconst = 1, latency = 4, avglen = 8;
        float rssdat, newmax, newmin, oldavg, newavg, avgthresh=1.0;
        float walkfudge = 0.0249;


        flag = 0;

        T1CON = 0;                                      // turn off interval timer and clear any IRQ
        T1CLRI = 0;

        while (1)
        {
                if (IRQSIG & XIRQ0_BIT)                 // XIRQ0 button has been pressed
                {
                        while(GP0DAT & 0x00010);        // wait for XIRQ to be low again
                        if (!flag)
                        {
                                T1CON = 0;              // turn off interval timer
                                T1CLRI = 0;             // clear any timer IRQ

                                stepflag = 2;
```

```
                maxavg = -10000.0;
                minavg = 10000.0;
                newmax = -10000.0;
                newmin = 10000.0;
                oldavg = 0.0;
                newavg = 0.0;
                cycle_count = 0;
                tot_samples = 0;
                steps = 0;
                distance = 0.0;
                accel_avg = 0.0;
                velocity = 0.0;
                displace = 0.0;
                avgstride = 0.0;

                display_data_clear();           // clear old data from display
                flag = 1;
                T1LD = 1092;                    // ~30 Hz sample rate
                T1CON = 0x2C0;                  // 32.768 kHz clock, timer on,
                                                // periodic mode
        } // if not running, start.
} // look for stop button
if (GP2DAT & 0x01)
{
                while(GP2DAT & 0x01);
        flag = 0;
} // if running, stop

if (((IRQSIG & GP_TIMER_BIT) && (flag)) != 0)   // wait for timeout
                                                // and flag
{
        T1CLRI = 0;
        if (tot_samples > 7)                    // subtract first sample in sliding boxcar avg
        {
                oldavg = newavg;
                newavg -= accel_dat[cycle_count - avglen];
        } // if

        get_sample(&xdat, &ydat, &zdat);        // get data from accelerometer
        xdat -= 8192;                           // subtract Zero g value
        ydat -= 8192;
        rssdat = sqrt((float)(xdat*xdat + ydat*ydat)/16.0);     // vector sum
        accel_dat[cycle_count] = rssdat;        // place current sample data in buffer

                        newavg += rssdat;       // add new sample to sliding boxcar avg
        if((abs(newavg-oldavg)) < avgthresh)
                newavg = oldavg;

                 if (rssdat > newmax)
                newmax = rssdat;
        if (rssdat < newmin)
                newmin = rssdat;
```

```
tot_samples++;
cycle_count++;                          // increment count of samples in current step

if (tot_samples > 8)
{
        if (IsStep(newavg, oldavg))
        {
                for (i = latency; i < (cycle_count - latency); i++)
                        accel_avg += accel_dat[i];
                accel_avg /= (cycle_count - avglen);

                for (i = latency; i < (cycle_count - latency); i++)
                {
                        velocity += (accel_dat[i] - accel_avg);
                        displace += velocity;
                } // create integration and double integration

        // calculate stride length
                stride = displace * (newmax - newmin) / (accel_avg - newmin);
                stride = sqrt(abs(stride));

        // use appropriate constant to get stride length
                stride *= walkfudge;

        // generate exponential average of stride length to smooth data
                if (steps < 2)
                        avgstride = stride;
                else
                        avgstride = ((avgconst-1)*avgstride + stride)/avgconst;

                steps++;
                distance += avgstride;

        // need all data used in calculating newavg
                for (i = 0; i < avglen; i++)
                        accel_dat[i] = accel_dat[cycle_count + i - avglen];

                cycle_count = avglen;
                newmax = -10000.0;
                newmin = 10000.0;
                maxavg = -10000.0;
                minavg = 10000.0;
                accel_avg = 0;
                velocity = 0;
                displace = 0;

                display_prep();
                display_data();

        // temporary
                if (GP4DAT & 0x04)              // toggle LED to reflect step
                        GP4CLR = 0x040000;
                else
```

```
                                        GP4SET = 0x040000;

                                } // we have a new step
                        } // enough samples to start checking for step (need at least 8)

                } // if timeout
        } // continual loop

}  // Monitor_Function()


void get_sample(unsigned int *xdat, unsigned int *ydat, unsigned int *zdat)
{
                                                        // gets new samples for x, y, z axes
                                                        // sums together 4 measurments to get average
        int i;

        *xdat = 0;
        *ydat = 0;
        *zdat = 0;

        for (i=0; i<15; i++)
        {
                ADCCP = 0;                              // x axis
                i++;                                    // delay one command cycle
                ADCCON = 0xA3;
                while (!(ADCSTA));
                *xdat += ((ADCDAT >> 16) & 0xFFF);      // data is in bits 16 - 27, so shift is necessary

                ADCCP = 1;                              // y axis
                i++;
                ADCCON = 0xA3;
                while (!(ADCSTA));
                *ydat += ((ADCDAT >> 16) & 0xFFF);

                ADCCP = 2;                              // z axis
                i++;
                ADCCON = 0xA3;
                while (!(ADCSTA));
                *zdat += ((ADCDAT >> 16) & 0xFFF);
        } // for

} // get_sample()


char IsStep(float avg, float oldavg)
{
                                                // this function attempts to determine when a step is complete

        float step_thresh = 5.0;                // used to prevent noise from fooling the algorithm

        if (stepflag == 2)
        {
```

```
                if (avg > (oldavg + step_thresh))
                        stepflag = 1;
                if (avg < (oldavg - step_thresh))
                        stepflag = 0;
                return 0;
        } // first time through this function

        if (stepflag == 1)
        {
                if ((maxavg > minavg) && (avg >
                                ((maxavg+minavg)/2)) &&
                                (oldavg < ((maxavg+minavg/2))))
                        return 1;
                if (avg < (oldavg - step_thresh))
                {
                        stepflag = 0;
                        if (oldavg > maxavg)
                                maxavg = oldavg;
                } // slope has turned down
                return 0;
        } // slope has been up

        if (stepflag == 0)
        {
                if (avg > (oldavg + step_thresh))
                {
                        stepflag = 1;
                        if (oldavg < minavg)
                                minavg = oldavg;
                } // slope has turned up
                return 0;
        } // slope has been down

        return 0;

} // IsStep()


void display_prep()
{
        int i;
        long int temp;

        // convert steps to BCD values for sending to display
        temp = steps;
        temp = bin_to_bcd(temp);                                // function to convert binary
        for (i=0; i<6; i++)                                     // to BCD
        {
                stepbcd[i] = (char)(0xF & temp);                // load each digit
                temp = temp >> 4;
        } // for

        // convert distance to BCD values for sending to display
```

```
        temp = (long int)(distance);                    // convert float to long int
        temp = bin_to_bcd(temp);
        for (i=0; i<6; i++)
        {
                distbcd[i] = (char)(0xF & temp);        // load each digit
                temp = temp >> 4;
        } // for

} // display_prep()


long int bin_to_bcd(long int bin_no)
{
        int i;
        long int divisor, multiplier, bcd_no, temp;

        divisor = 100000;
        multiplier = 1048576;
        bcd_no = 0;
        temp = 0;

        if (bin_no > 999999)
                bin_no = 999999;

        for (i=0; i<6; i++)
        {                                               // determine each digit starting
                temp = bin_no/divisor;                  // with most significant
                bin_no -= temp*divisor;                 // subtract this amt
                temp *= multiplier;                     // generate hex equivalent
                bcd_no += temp;                         // put bcd value together

                divisor /= 10;                          // go to next digit
                multiplier = multiplier >> 4;
        } // for

        return bcd_no;

} // bin_to_bcd()
```


AN06617-0-7/07(0)