Appendix A Servo Motor Testing Data

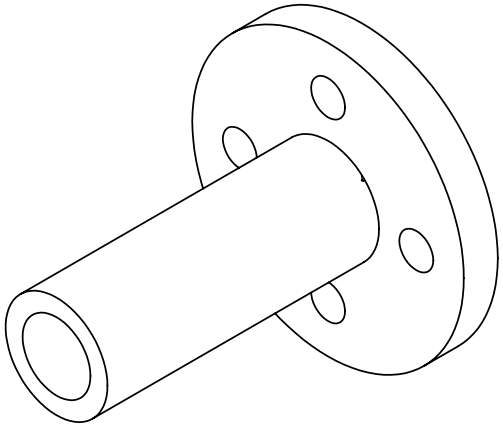
HS-475HB

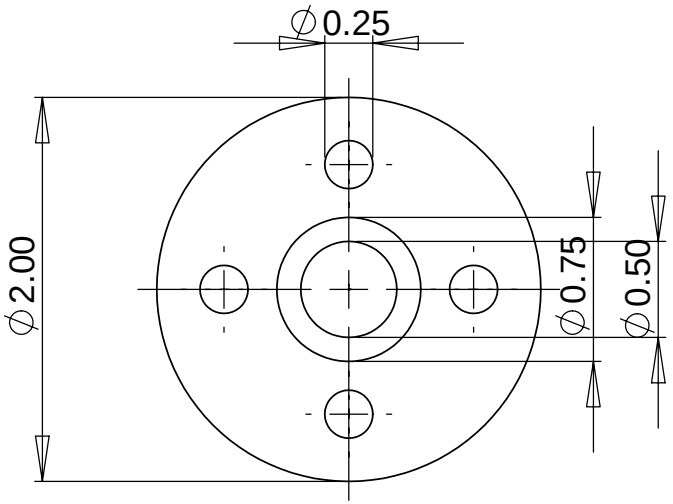
Time (ms)	Amplitude (V)	d(theta)	d(time)	ω	|ω|
151.5	3.1	1.576137	0.0175	90.06497	90.064973
169	4.55				
201	4.55	-1.57614	0.016	-98.5086	98.508565
217	3.1				
252	3.1	1.576137	0.016	98.50856	98.508565
268	4.55				
301	4.55	-1.51092	0.016	-94.4323	94.432348
317	3.16				
482	3.1	1.576137	0.016	98.50856	98.508565
498	4.55				
531	4.55	-1.57614	0.0185	-85.1966	85.196596
549.5	3.1				
1502	3.1	1.576137	0.02	78.80685	78.806852
1522	4.55				
1552	4.55	-1.51092	0.016	-94.4323	94.432348
1568	3.16				
1403	3.1	1.576137	0.0165	95.52346	95.523457
1419.5	4.55				
1452	4.55	-1.51092	0.017	-88.8775	88.877504
1469	3.16				
				ω_ave	92.29 or 1.6106942

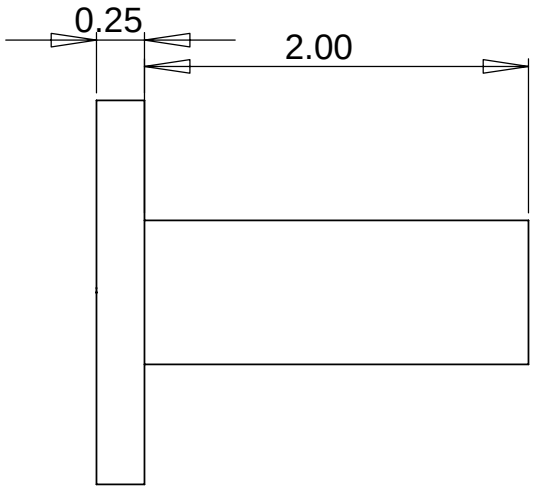
HSR5995TG

Time (ms)	Amplitude (V)	d(theta)	d(time)	ω
10730	4.3	0.869593	0.016	54.34955
10746	5.1			
10539	5.1	-0.86959	0.019	45.76804
10558	4.3			
10142	5.1	-0.97829	0.019	51.48905
10161	4.2			
10334	4.2	0.978292	0.017	57.54659
10351	5.1			
2831	1.3	1.304389	0.026	50.16882
2857	2.5			
2930	2.5	-1.30439	0.026	50.16882
2956	1.3			
3030	1.25	1.304389	0.028	46.58533
3058	2.45			
3129	2.45	-1.30439	0.025	52.17557
3154	1.25			
3429	1.2	1.304389	0.027	48.31071
3456	2.4			
3529	2.4	-1.30439	0.026	50.16882
3555	1.2			
			ω_ave	50.67313 or 0.884413

Appendix B CAD drawings of Support System





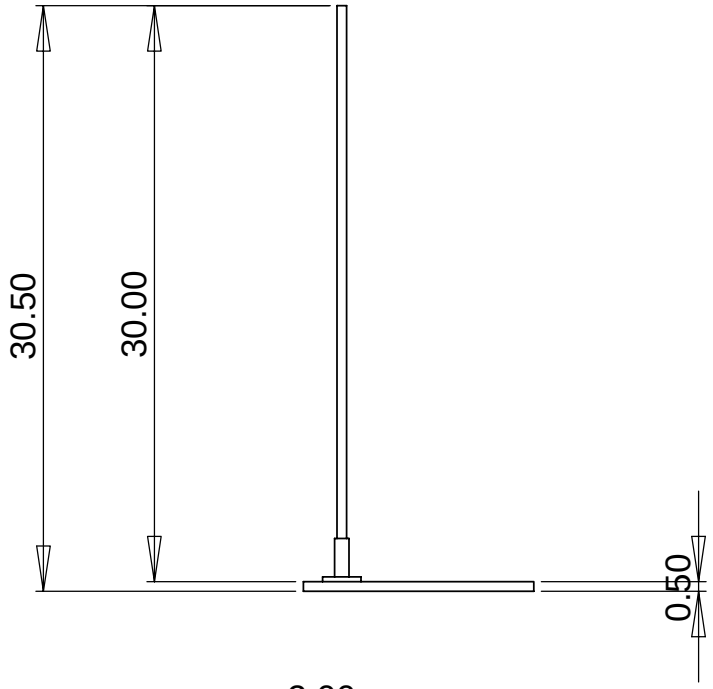


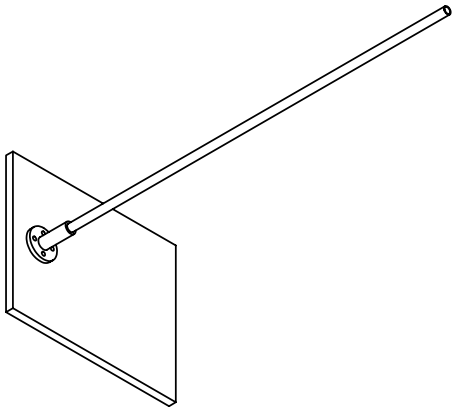
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

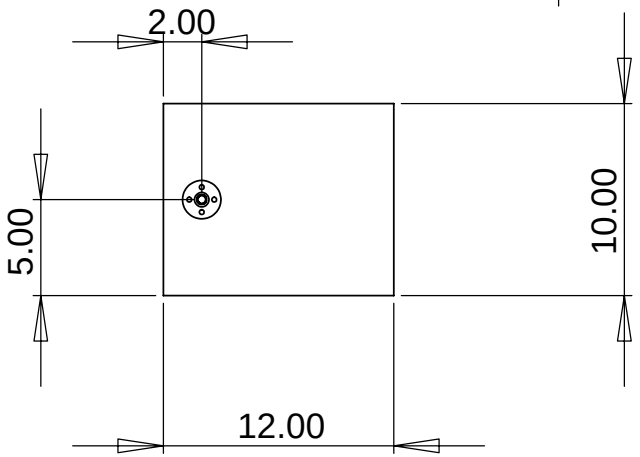
TITLE:

SIZE	DWG. NO.	REV
A	flange1	
SCALE: 1:1	WEIGHT:	SHEET 1 OF 1

5 4 3 2 1

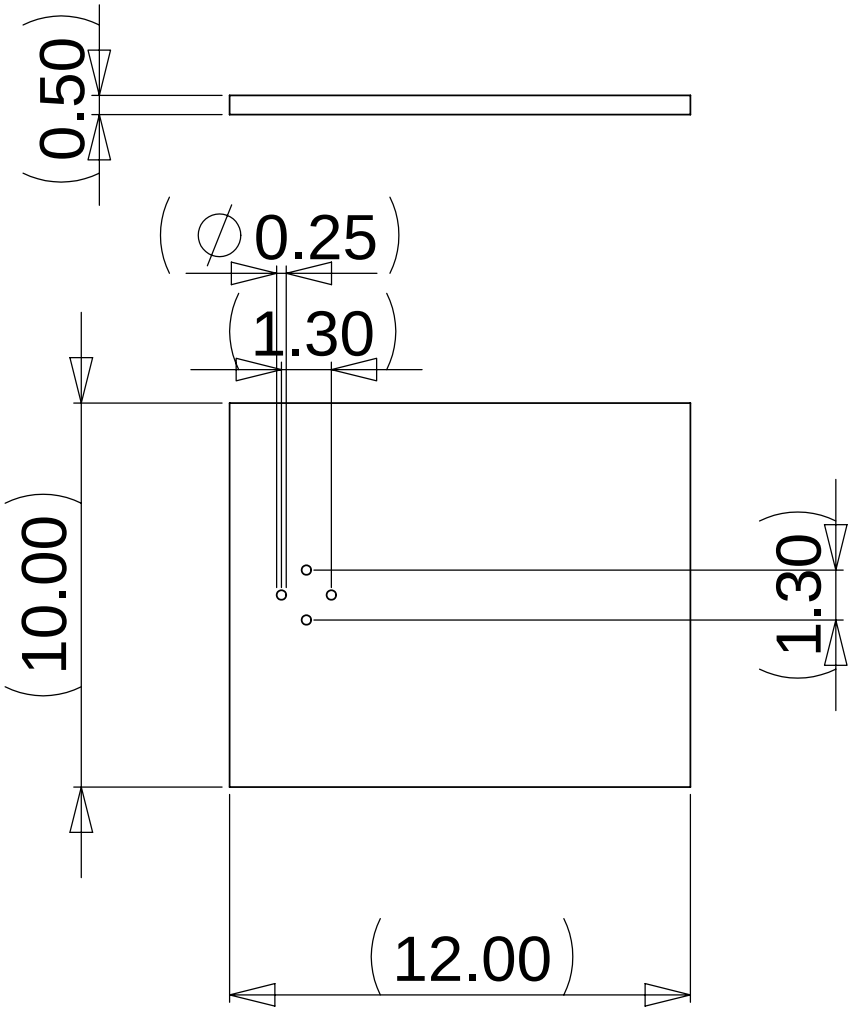






SIZE	DWG. NO.	REV
A	stand	
SCALE: 1:10	WEIGHT:	SHEET 1 OF 1

5 4 3 2 1



SIZE	DWG. NO.	REV.
A		
SCALE:1:5	WEIGHT:	SHEET 1 OF 1

Appendix C RC Biped Code

The bulk of the program is written by Steve Bagg.

```
#include "kobrain.h"
#include "Attitude Estimationdemo.h"



    #define MINUSONEFF 0x00008000    //FFloat number -1
     #define ZEROFF         0xFF800000    //FFloat number 0
     #define ONEFF      0x00014000    //FFloat number 1
     #define TWOFF      0x00024000    //FFloat number 2
     #define THREEFF        0x00026000    //FFloat number 3
     #define FOURFF         0x00034000    //FFloat number 4
     #define FIVEFF         0x00035000    //FFloat number 5
     #define SIXFF      0x00036000    //FFloat number 6
     #define SEVENFF        0x00037000    //FFloat number 7
     #define EIGHTFF        0x00044000    //FFloat number 8
     #define NINEFF         0x00044800    //FFloat number 9
     #define TENFF      0x00045000    //FFloat number 10
     #define ELEVENFF   0x00045800    //FFloat number 11
     #define TWELVEFF   0x00046000    //FFloat number 12

     #define LOOPTIME   0xFFF74189    //FFloat number 0.001

    #define WIRECONN   1

     typedef long unsigned int ffloat;
    typedef   unsigned char bool;


/*****************************************************************************
                          KoBrain Global Variables
*****************************************************************************/
/* Global variable def. for ROM-RAM copy. Note location - before function definitions.
A default value from the definition is stored in flash memory, but copied to RAM at
start. This allows the value to be easily modified during operation.
*/
short int Parameter[128];
short int PrevParameter[128];
short int TimeHip=0,TimeOuter=0,TimeInner=0,PWMvalue=0;
unsigned short int Elapsed=0;
short int AD7490_Values[16];
short int Variable[128];
ffloat GBL_Param[96];               //Contains control constants. The first 24 (0 to 23) are                    ¬
reserved.
ffloat GBL_PrevParam[96];
ffloat GBL_Data[128];

/************************************************************
                RC_Biped Global Variables
************************************************************/
ffloat THETA[3];
ffloat THETAd[3];
short int speed[8]={1,1,1,1,1,1,1,1};
short int lmov[8],mstate[8];
char step_complete;
long int acc[3],inter[3];
short int calibrate=0,calsec=0,check;
long int co[3],lo[3],hi[3];
char calidone;
char start;

long unsigned int GBL_Elapsed_mS;   //32-bit counter of mS since MCU reset or power-up
short int mov_index=0;


/*****************************************************************************
                          KoBrain FUNCTION PROTOTYPES
*****************************************************************************/

void init_GPIOA(void);
void init_GPIOB(void);
void init_GPIOD(void);
void init_GPIOE(void);
void Init_GPIOF(void);
void init_SPI0_AD(void);
void init_TMRC0(void);
void init_TMRC1(void);
```

```
void init_TMRC2(void);
void init_TMRC3(void);
void init_TMRD0(void);
void init_TMRD1(void);
void init_TMRD2(void);
void init_TMRD3(void);
void init_SCI0(void);
void init_SCI1(void);
void init_ADCA(void);
void init_ADCB(void);
void Rec_Data(void);
unsigned int Send_Data(unsigned int address,ffloat data);
void Data_Sequencer(void);
void timestamp(void);
void init_PWMA(void);
void init_PWMB(void);
void update_PWMA(void);
void update_PWMB(void);
void init_interrupts(void);
void intRoutine_TMRC0(void);
void intRoutine_TMRC1(void);
void intRoutine_TMRC3(void);
void intRoutine_TMRD1(void);
void intRoutine_TMRD3(void);
void update_AD12(void);
short int div_ls4q(long int l_numerator, short int s_denominator);


/*************************************************************
                RC_Biped Function Prototypes
 *************************************************************/
void init_parameters(void);
void sensor_read(void);
void update_Sensor(void);
void control_Output(void);
void motor_Control(void);
void move_ForLeft(void);
void move_ForRight(void);
void move_BackLeft(void);
void move_BackRight(void);
void step_Left(void);
void step_Right(void);
void setPWMval(void);
void checkmax(void);
void test_motor2(void);
void test_motor4(void);
void Rforward(void);
void Rbackward(void);
void Lforward(void);
void Lbackward(void);
void err_biped(void);
void fall_state(void);
void init_biped(void);
void movements(void);
void cali_gyro(void);
void init_gyro(void);
void Init_Wireless(void);
void max_pend(void);
void conv_wireless(void);
void accel_Theta(void);
short int angle2pos(short int angle);
void move_motor(short int mnum, short int ang);
short int ABS(short int num);
void go_neutral(void);
void motor_pend(void);
void pend_step(void);
void movements(void);
short int pos2angle(short int pos);
void AttitudeEstimator(void);
void testimator(void);
/*****************************************************************************
                          FFloat FUNCTION PROTOTYPES
*****************************************************************************/

asm ffloat FFabs(register ffloat ffnum);
asm ffloat FFneg(register ffloat ffnum);
asm ffloat S16int2FFloat(register short int inum);
asm ffloat S32int2FFloat(long int inum);
```

```
asm ffloat FFadd(register ffloat ffnum1,register ffloat ffnum2);
asm ffloat FFdiv(register ffloat ffnum1,register ffloat ffnum2);
asm short int FFloatTrunc2S16int(register ffloat ffnum);
asm short int FFloatRnd2S16int(register ffloat ffnum);
asm ffloat FFmult(register ffloat ffnum1,register ffloat ffnum2);
asm ffloat FFsub(register ffloat ffnum1,register ffloat ffnum2);
asm ffloat IEEE2FFloat(register float fnum);
float FFloat2IEEE(ffloat ffnum);
asm bool FFgt(register ffloat ffnum1, register ffloat ffnum2);
asm bool FFlt(register ffloat ffnum1, register ffloat ffnum2);
asm bool FFgte(register ffloat a, register ffloat b);
asm bool FFlte(register ffloat a, register ffloat b);
asm bool FFgtz(register ffloat ffnum);
asm bool FFltz(register ffloat ffnum);
asm bool FFeqz(register ffloat ffnum);
ffloat FFatan(ffloat xin);
ffloat FFsin(ffloat xin);
ffloat FFcos(ffloat xin);


//FFloat function definitions

#include "FFloat_AE.c"
/*********************************************************************************
/*********************************MISCELLANEOUS FUNCTIONS*************************
/*********************************************************************************/

/*********************************************************************************
    init_parameters gives the default values for the parameters
    this is only run once
 *********************************************************************************/

void init_parameters (void)
{
THETAdx = ZEROFF;
THETAdz = ZEROFF;
THETAdy = ZEROFF;
Kpx = IEEE2FFloat(0.0125);
Kpz = IEEE2FFloat(0.0125);
Kpy = IEEE2FFloat(0.1);
Tdx = IEEE2FFloat(0.1);
Tdy = IEEE2FFloat(0.1);
Tdz = IEEE2FFloat(0.1);
THETAx = ZEROFF;
THETAy = ZEROFF;
THETAz = ZEROFF;
step_complete = FALSE;
stepnum = 0;
RKnee=NEUTRAL;
LKnee=NEUTRAL;
LHipx=NEUTRAL;
LHipy=NEUTRAL;
LHipz=NEUTRAL;
RHipx=NEUTRAL;
RHipy=NEUTRAL;
RHipz=NEUTRAL;

Rhipxlast=NEUTRAL;
Lhipxlast=NEUTRAL;
Rkneelast=NEUTRAL;
Lkneelast=NEUTRAL;

setPWMval();
 calidone=0;
 start=1;
 }

/*****************************************************
Sam's Gyro Functions
*****************************************************/

void init_gyro(void)
{
     short int i;
     for(i=0;i<3;i++)
     {
          acc[i]=0;
          inter[i]=0;
```

```
          co[i]=0;
          lo[i]=-2048;
          hi[i]=2048;
      }
 }


 void cali_gyro(void)
 {
      short int i,temp0,temp1,temp2,temp;
          Variable[9]=check++;

    for(i=0;i<3;i++) //loop to run for x,y,z
    {
        if (acc[i]<0) //constant too low
        {
          lo[i]=co[i];
          co[i]=(co[i]+hi[i])>>1;
        }
        else if (acc[i]>0)//constant too high
        {
          hi[i]=co[i];
          co[i]=(co[i]+lo[i])>>1;
        }
        acc[i]=0; //resets acculmulator
        //Variable[3+i]=acc[i]>>16;
     }
     calibrate++;

     if (calibrate==12)
     {
         YELLED_OFF;
         calidone=1;
     }
     temp0=co[0];
     temp1=co[1];
     temp2=co[2];
     temp=calibrate;
     calsec=0;

 }
 void Init_Wireless(void)
 {


     unsigned long int ptr,i,j;
     unsigned long int data[]={ 0xFF, 0x02, 0x4E, 0x05,         //programming data for module
                                0xFF, 0x02, 0x4B, 0x10,
                                0xFF, 0x02, 0x4C, 0x10,
                                0xFF, 0x02, 0x4D, 0x03,
                                0xFF, 0x02, 0x4F, 0x01,
                                0xFF, 0x02, 0x50, 0x10,
                                0xFF, 0x02, 0x51, 0x00,
                                0xFF, 0x02, 0x53, 0x01,
                                0xFF, 0x02, 0x54, 0x80,
                                0xFF, 0x02, 0x55, 0x01,
                                0xFF, 0x02, 0x56, 0x01,
                                0xFF, 0x02, 0x58, 0x00};


     unsigned int temp;

     //*GPIOB_DR     = *GPIOB_DR & 0b0000000001111111;             //wireless CMD line low
       *GPIOD_DR     = *GPIOD_DR & 0b0000000011101111;             //wireless CMD line low

     //Data register
    *SCI1_SCIDR = 0x0000; //clears data register

     //Baud rate register
     *SCI1_SCIBR = 1563; //2400baud at 60Mhz clock (only for 1st iteration of j)
     //   *SCI1_SCIBR = 33;

     //Control register
     *SCI1_SCICR = 0x000C; //Transmit enable, receive enable, 8 bit, NP, 1 stop

     for(i=0;i<50000;i++){;} //delay

     for(j=0;j<48;j++)
```

```
    {
        ptr = data[j]; //sequence to bring baud rate to 115kBaud


        //CHANGED POLARITY!!
        //while (*SCI1_SCISR & 0x8000 == 0x8000){;} //wait for TDRE set
        while (!(*SCI1_SCISR & 0x8000)){;} //wait for TDRE set
        temp=*SCI1_SCISR;
        *SCI1_SCIDR = ptr & 0x000000FF; //Stores last 8bits into Data register

        for(i=0;i<50000;i++){;} //delay


        if(j==3) //Sets Baud rate register to 115kBaud once it has  executed at 2400baud
        {

            *SCI1_SCIBR = 33; //115 Kbaud (use decimal 260 for 9600) at 60 MHz clock
                for(i=0;i<50000;i++){;} //delay
        }

    }

    //*GPIOB_DR   = *GPIOB_DR | 0b0000000010000000;               //wireless CMD line high
    *GPIOD_DR     = *GPIOD_DR | 0b0000000000010000;
    //Baud rate register
    *SCI1_SCIBR = 33; //115 Kbaud (use decimal 391 for 9600) at 60 MHz clock

    //Control register
    *SCI1_SCICR = 0b0000000000001100; //Transmit enable, receive enable, 8 bit, NP, 1 stop

   //Data register
   *SCI1_SCIDR = 0x0000;
}



void go_neutral(void)
{
    RHipx = NEUTRAL;
    RHipy=NEUTRAL;
    RHipz=NEUTRAL;
    RKnee=NEUTRAL;
    LHipx=NEUTRAL;
    LHipy=NEUTRAL;
    LHipz=NEUTRAL;
    LKnee=NEUTRAL;
    setPWMval();
}

/*****************************************************
  update_Theta integrates the gyro data once per loop
 *****************************************************/
void AttitudeEstimator(void)
{

        static long int Angle_Gyro_Outer = 0;
        static long int Angle_Rate_Adjust = 0;
        static long int Angle_Rate_Offset = 0;
        static long int Error_Avg = 0;
        static bool        Reset = 0;   //When 1, reset integrator to zero error

        long int AccelZ_Int;

         //Angle_Rate_Adjust = FFloatRnd2S16int(GBL_Param[TestInput1]);

        Angle_Gyro_Outer += ((XGYRORAW*403) + Angle_Rate_Offset + Angle_Rate_Adjust);

        //GBL_Data[TestOutput1] = FFmult(S32int2FFloat(Angle_Gyro_Outer),0xFFE36487);
        Testoutput1 = FFmult(S32int2FFloat(Angle_Gyro_Outer),0xFFF84000);

        AccelZ_Int = -(*ADCB_ADRSLT7 - 16241);

        AccelZ_Int *= 715;

        Testoutput2 = S32int2FFloat(AccelZ_Int);

        Error_Avg = AccelZ_Int - (Angle_Gyro_Outer>>9);
```

```
        Testoutput3 = S32int2FFloat(Error_Avg);

        if (!(GBL_Elapsed_mS & 0x00000FFF)) //Alternate between reset and measure
        {
                if (Reset == 0)
                {
                  Angle_Rate_Adjust = ((Error_Avg*6) >> 6) - (Angle_Rate_Adjust >> 1);

                  Angle_Rate_Offset += (Angle_Rate_Adjust >> 4);

                  Testoutput4 = S32int2FFloat(Error_Avg);
                  Error_Avg = 0;
                  Reset = 1;

                }

                else
                {
                  Reset = 0;
                }
        }

}

void testimator(void)
{THETAx=Testoutput1;
speed[m_rhipx]=3;
 if(FFgtz(THETAx))
   move_motor(m_rhipx,-30);
 if(FFltz(THETAx))
   move_motor(m_rhipx,30);
 setPWMval();
 }


void sensor_read(void)
{

     //THETAx=Variable[3];
     //THETAy=Variable[4];
     //THETAz=Variable[5];

     static long Angavg; // angular rate average
     static long Xavg;   // filtered integrated angle
     static long Err;   // Gyro and Acc averaged error

     if (start == 1)
     {
          Xavg = 0;
          Err = 0;
          start= 0;
     }

Angavg += XGYRORAW + Err; // Integrated Gyro

Xavg += Angavg>>8 - Xavg>>8; // Filtered Integrated Gyro

Err = (Xavg - (XACCELRAW>>2)<<3)<<3; // GyroAvg - AccAvg

Variable[3]=Angavg;


//If right leg is in front of left leg, then the right foot is forward
//RHipxCheck=(RHipx-NEUTRAL)*(-1);
//LHipxCheck=(LHipx-NEUTRAL);

//if(RHipxCheck>=LHipxCheck)

/*switch(step_complete)
{
     case(TRUE):

if((RHipx-NEUTRAL)*(-1)>(LHipx-NEUTRAL))
{
     RFOOT=RightDownFront;
     LFOOT=LeftDownBack;
}
else
```

```
{
     RFOOT=RightDownBack;
    LFOOT=LeftDownFront;
}
step_complete=FALSE;
break;

     case(FALSE):
   break;

}*/
}


/*********************SET PWM VAL************************/
/*takes calues from step function and sets to pwm values*/
/********************************************************/
void setPWMval(void)
{
     RKneePWM = RKnee;
     RHipxPWM = RHipx;
     RHipyPWM = RHipy;
     RHipzPWM = RHipz;

     LKneePWM = LKnee;
     LHipxPWM = LHipx;
     LHipyPWM = LHipy;
     LHipzPWM = LHipz;
     update_PWMA();
    update_PWMB();
}

void checkmax()
{int numero;
for (numero = 26;numero <= 33; ++numero)
  { if (Variable[numero] < MINPOS)
    {Variable[numero] = MINPOS;}
    if (Variable[numero] > MAXPOS)
    {Variable[numero] = MAXPOS;}
  }

}

/*****************************************************************************
                        TEST FUNCTIONS
*****************************************************************************/
/*****************************************************************************
*****************************************************************************
*****************************************************************************/
//for motor connection with board.  Side with R label is ground, far side is signal.




short int angle2pos(short int angle)
{short int new_angle;
long int added_time,position;

added_time = angle*40/9;

position= ((added_time+1500)*32767)/8772;

new_angle = position;
return (new_angle);
}

short int pos2angle(short int pos)
{ffloat FFnew_pos,FFpos,conv;
short int new_pos;
FFpos=S16int2FFloat(pos-NEUTRAL);
conv=IEEE2FFloat(0.060234);
FFnew_pos=FFmult(FFpos,conv);
new_pos=FFloatRnd2S16int(FFnew_pos);
return (new_pos);
}

void conv_wireless(void)
{
```

```
ACCELx=S16int2FFloat(XACCELRAW);
ACCELy=S16int2FFloat(YACCELRAW);
ACCELz=S16int2FFloat(ZACCELRAW);
GYROx=S16int2FFloat(XGYRORAW);
GYROy=S16int2FFloat(YGYRORAW);
GYROz=S16int2FFloat(ZGYRORAW);

}

/*****************************************************************************
*****************************************************************************
                                move_motor
*****************************************************************************
*****************************************************************************/
void move_motor(short int mnum, short ang)
{short int new_pos,f_pos,dif,absdif,pos;
short int start_deccel[6]={1,4,12,20,26,32};


 /*vals has the information for the motor in it
 mnum - motor number (rk 1, )
 //start_deccel - dif at which to begin decelerating
 pos - motor position
 speed[8] - motor ss speed
 lmov[8] - last motor movement
 mstate[8] - phase indicator (0 is beginning, 1 accel, 2 const vel, 3 decel)
 way to give a motor command
 speed[m_rknee]=1;
 move_motor(m_rknee,45);
 */
    pos= Parameter[29+mnum];
    f_pos = angle2pos(ang);
    dif = f_pos - pos;
    absdif = ABS(dif);

//original values were [1 3 6 10 15 21]
    if (dif == 0)
    {mstate[mnum]=0;}

    switch(mstate[mnum])
    {
    case 0:
    //calculate number of steps
    //start motion new_pos=pos (+-) 1;
        if (dif <0)
        {new_pos = pos - 1;
        lmov[mnum]=1;
        mstate[mnum]=1;}
        else
        {
         if (dif > 0)
            {new_pos = pos + 1;
            lmov[mnum]=1;
            mstate[mnum]=1;}
            else
            {new_pos = pos;
            mstate[mnum]=0;}
        }
    break;

    case 1:
    //accel
    lmov[mnum]=lmov[mnum]+1;
    if (lmov[mnum]<speed[mnum])
    {
        if(dif<0)
        {new_pos = pos - lmov[mnum];}
        else
        {new_pos = pos + lmov[mnum];}
    }
    else
    {mstate[mnum]=2;
     new_pos=pos;}

    break;

    case 2:
```

```
     //const vel: new pos = pos+speed;
     lmov[mnum]=speed[mnum];
     if (absdif <= start_deccel[speed[mnum-1]])
          {mstate[mnum]=3;}

     if(dif<0)
          {new_pos = pos - speed[mnum];}
     else
          {new_pos = pos + speed[mnum];}


     break;

     case 3:
     //deccel
     lmov[mnum]=lmov[mnum]-1;
     if (lmov[mnum]==0)
     lmov[mnum]=1;
     if(dif<0)
          {new_pos = pos - lmov[mnum];}
     else
          {new_pos = pos + lmov[mnum];}
     if (absdif<2)
     mstate[mnum]=0;
     break;
     }
     Parameter[29+mnum]=new_pos;
//setPWMval();
}

short int ABS(short int num)
{short int absnum;
     if(num<0)
     absnum=num * -1;
     else
     absnum=num;
     return absnum;
}


/*****************************************************************************
*****************************************************************************
                  biped stuff
*****************************************************************************
*****************************************************************************/

void err_biped(void)
{ffloat d2rad,cz,sz,Epx,Epz,Epy;
   THETAx=S16int2FFloat(Variable[3]);
   //THETAx=Testoutput1;
   THETAy=S16int2FFloat(Variable[4]);
   THETAz=S16int2FFloat(Variable[5]);
   d2rad=IEEE2FFloat(0.01745);
   cz=FFcos(FFmult(d2rad,pos2angle(RHipz)));
   sz=FFsin(FFmult(d2rad,pos2angle(RHipz)));
   Epx=FFsub(THETAdx,THETAx);
   Epy=FFsub(THETAdy,THETAy);
   Epz=FFsub(THETAdz,THETAz);
   Uxx=FFmult(FFmult(Kpx,Epx),cz);
   Uxy=FFmult(FFmult(Kpx,Epx),sz);
   Uz=FFmult(Kpz,Epz);
   Uyx=FFmult(FFmult(Kpy,Epy),sz);
   Uyy=FFmult(FFmult(Kpy,Epy),cz);

}
void motor_pend(void)
{
     Uxxi=FFloatRnd2S16int(Uxx);

    /*RHipx=Rhipxlast-Uxxi;
    LHipx=Lhipxlast-Uxxi;
    RKnee=Rkneelast+Uxxi;
    LKnee=Lkneelast+Uxxi;
    //RKnee=NEUTRAL;
    //LKnee=NEUTRAL;
    RHipy=NEUTRAL;
    LHipy=NEUTRAL;
    RHipz=NEUTRAL;
```

```
     LHipz=NEUTRAL;
     Rhipxlast=RHipx;
     Lhipxlast=LHipx;
     Rkneelast=RKnee;
     Lkneelast=LKnee;*/
     LHipx=Lhipxlast+Uxxi;
     RHipx=Rhipxlast-Uxxi;
     RKnee=NEUTRAL;
     LKnee=NEUTRAL;
     RHipy=NEUTRAL;
     LHipy=NEUTRAL;
     RHipz=NEUTRAL;
     LHipz=NEUTRAL;
     Rhipxlast=RHipx;
     Lhipxlast=LHipx;



     /*Uxxi=FFloatRnd2S16int(Uxx);
     Uzi= FFloatRnd2S16int(Uz);
     Uxyi=FFloatRnd2S16int(Uxy);
     RKnee=NEUTRAL;
     LKnee=NEUTRAL;
     RHipx=Rhipxlast;//-Uxxi;
     LHipx=NEUTRAL;
     RHipy=Rhipylast;//-Uxyi;//-Uyyi
     RHipz=Rhipzlast+Uzi;

     Rhipxlast=RHipx;
     Rhipzlast=RHipz;
     Rhipylast=RHipy;*/
     RHipz=RHipz-50;
     LHipz=LHipz+50;
     setPWMval();

 }

 void pend_step(void)
 {
      Uxxi=FFloatRnd2S16int(Uxx);

      LHipx=Lhipxlast+Uxxi;
     RHipx=Rhipxlast+Uxxi;
     RKnee=Rkneelast-Uxxi;
     LKnee=Lkneelast-Uxxi;
     RHipy=NEUTRAL;
     LHipy=NEUTRAL;
     RHipz=NEUTRAL;
     LHipz=NEUTRAL;

     Rhipxlast=RHipx;
     Lhipxlast=LHipx;
     Rkneelast=RKnee;
     Lkneelast=LKnee;

     RHipz=RHipz-50;
     LHipz=LHipz+50;

      setPWMval();
 }

 void fall_state(void)
 {
       /*figure out the case for stepping direction
       breaks it into four quadrants
         double check for correctness*/
    if (FFltz(Uxx))
    {    if (FFltz(Uz))
      {  CASE = ForLeft;}
          else
      { CASE = ForRight;}}
    else
    { if (FFltz(Uz))
    { CASE = BackLeft;}
         else
     { CASE = BackRight;}}

  switch(CASE)
```

```
 {   case ForLeft:
     move_ForLeft();
     break;

     case ForRight:
     move_ForRight();
     break;

     case BackLeft:
     move_BackLeft();
     break;

     case BackRight:
     move_BackRight();
     break;
 }

}

/*******************************************************
                     Moving Functions
*******************************************************/
void move_ForLeft(void)
{/*for 1D case
 read foor placement (up/down, front,back)
 decide which foot to move (most cases move back foot forward)
 if both feet are parallel move left foot first.
 Also based on foot placement decide which type of step to take.
 Call appropriate step function.

 (left neutral, rightneutral) falling forward
    to (left neutral, rightdownfront) (right forward)
 (left neutral, rightdownfront) falling back
    to (left neutral, rightdownback) (right back)
 falling forward
    to (left neutral, rightdownfront)

 */

if (RFOOT==RightDownBack || RFOOT==RightDownFront)
     {
          if (RFOOT==RightDownBack && LFOOT==LeftDownFront)
          {
               STEP = RSTEP;
          }

          else //(LFOOT==LeftDownBack || LFOOT==LeftDownFront)
          {
               //STEP = LSTEP;
          }

          //if (LFOOT==LeftUpBack || LFOOT==LeftUpFront)
          //{  //if LeftUpBack then just left hip step (knee neutral)
          //   //if LeftUpFront then just left hip(maybe half-size?) step
          //
     }

if (RFOOT==RightUpBack || RFOOT==RightUpFront)
     {
          //Always just right hip step
     }

switch(STEP)
{
     case RSTEP:
     //(crossing over) then full RIGHT step
     //move left to neutral, right to RightDownFront
     ++stepcounter;
     speed[m_rhipx]=2;
     speed[m_rknee]=2;
     move_motor(m_rhipx,-60);
     move_motor(m_rknee,30);
     if (stepcounter > 1000)
     {
          step_complete=TRUE;
          stepcounter=0;
     }
     break;
```

```
     case LSTEP:
     //if LeftDownBack then full left step
     //if LeftDownFront(parallel) then full(maybe half-size?) left step
     ++stepcounter;
     speed[m_lhipx]=2;
     speed[m_lknee]=2;
     move_motor(m_lhipx,60);
     move_motor(m_lknee,-30);
     if (stepcounter > 1000)
     {
          step_complete=TRUE;
          stepcounter=0;
     }
}
}

void move_ForRight(void)
{

}

void move_BackLeft(void)
{

if (RFOOT==RightDownBack || RFOOT==RightDownFront)
     {
          if (RFOOT==RightDownFront && LFOOT==LeftDownBack)
          {
               //full RIGHT back step
          }
          if (LFOOT==LeftDownBack || LFOOT==LeftDownFront)
          {
               //if LeftDownBack then full left back step
               //if LeftDownFront(parallel) then full(maybe half-size?) left step

               ++stepcounter;
               move_motor(m_rhipx,30);
               move_motor(m_rknee, 0);
               if(stepcounter>1000)
               {
                    step_complete=TRUE;
                    stepcounter=0;
               }

          }
          if (LFOOT==LeftUpBack || LFOOT==LeftUpFront)
          {    //if LeftUpBack then just left hip back step (knee neutral)
               //if LeftUpFront then just left hip(maybe half-size?) back step

               //++stepcounter;
               //move_motor(m_lhipx,-30);
               //move_motor(m_lknee, 0);
               //if(stepcounter>1000)
               //{
               //   step_complete=TRUE;
               //   stepcounter=0;
               //}


          }
     }

     if (RFOOT==RightUpBack || RFOOT==RightUpFront)
     {
          //Always just right hip back step
     }
}

void move_BackRight(void)
{

}
```

```
void movements(void)
{
static char ldone,rdone;
// Dodge Bullets 1
//short int rhipxm[7]={0, -90, -90, -90, 0, 0, 0}; //(-)
//short int lhipxm[7]={0, 90, 90, 90, 0, 0, 0}; // (+)
//short int rkneem[7]={0, 0, 30, 0, 0, 0, 0}; //(-)
//short int lkneem[7]={0, 0, -30, 0, 0, 0, 0}; // (+)

// Dodge Bullets 2
//short int rhipxm[7]={0, 10, 20, 45, 20, 10, 0}; //(-)
//short int lhipxm[7]={0, -10, -20, -45, -20, -10, 0}; // (+)
//short int rkneem[7]={0, 10, 20, 45, 20, 10, 0}; //(-)
//short int lkneem[7]={0, -10, -20, -45, -20, -10, 0}; // (+)

// Kick 1
//short int rhipxm[7]={0, -45, -90, -90, -45, 0, 0}; //(-)
//short int lhipxm[7]={0, 0, 0, 0, 0, 0, 0}; // (+)
//short int rkneem[7]={0, 0, 90, 0, 0, 0, 0}; //(-)
//short int lkneem[7]={0, 0, 0, 0, 0, 0, 0}; // (+)

// Bow
//short int lhipxm[7]={10,  30,  60, 90, 60, 30, 0}; // (+) (Positive CW viewed from left)
//short int rhipxm[7]={-10, -30, -60, -90, -60, -30, 0}; // (-) (Positive CW viewed from left)
//short int rkneem[7]={0, 0, 0, 0, 0, 0, 0}; //(-)
//short int lkneem[7]={0, 0, 0, 0, 0, 0, 0}; // (+)

// Kick 1
//short int lhipxm[7]={0,10,30,60,90,60,30}; // (+) (Positive CW viewed from left)
//short int rhipxm[7]={0,-10,-30,-60,-90,-60,-30}; // (-) (Positive CW viewed from left)
//short int rkneem[7]={0,0,0,0,0,0,0}; //(-)
//short int lkneem[7]={0,0,0,0,0,0,0}; // (+)

// Walking
short int lhipxm[7]={ 10, 10,  0,  0,-10,-10, 10}; // (+) (Positive CW viewed from left)
short int rhipxm[7]={  0, 10, 10,-10,-10,  0,  0}; // (-) (Positive CW viewed from left)
short int lkneem[7]={-10,-10,  0,  0, 10, 10,-10}; //(-)
short int rkneem[7]={  0,-10,-10, 10, 10,  0,  0}; // (+)
short int rhipzm[7]={-21, -6,  9,  9, -6,-21,-21}; //do not change
short int lhipzm[7]={ -9,  6, 21, 21,  6, -9, -9}; //do not change
// Stand-sit

//short int lhipxm[9]={1,4,10,52,10,32,0,32,0}; // (+) (Positive CW viewed from left)
//short int rhipxm[9]={-1,-4,-10,-52,-10,-32,0,-32,0}; // (-) (Positive CW viewed from left)
//short int lkneem[9]={-1,-4,-10,-52,-10,-32,0,-32,0}; //(-)
//short int rkneem[9]={1,4,10,52,10,32,0,32,0}; // (+)

//Hump
//short int lhipxm[7]={0,40,0,40,0,40,0}; // (+) (Positive CW viewed from left)
//short int rhipxm[7]={0,-40,0,-40,0,-40,0}; // (-) (Positive CW viewed from left)
//short int lkneem[7]={0,-30,0,-30,0,-30,0}; //(-)
//short int rkneem[7]={0,30,0,30,0,30,0}; // (+)

// Dodge Bullets
//short int lhipxm[7]={0,-10,-20,-30,-20,-10,0}; // (+) (Positive CW viewed from left)
//short int rhipxm[7]={0,10,20,30,20,10,0}; // (-) (Positive CW viewed from left)
//short int rkneem[7]={0,10,20,30,20,10,0}; //(-)
//short int lkneem[7]={0,-10,-20,-30,-20,-10,0}; // (+)

//one legged dips
//short int lhipxm[7]={0, 0,  0, 40,  0, 0,0}; // (+) (Positive CW viewed from left)
//short int rhipxm[7]={0, 0,-40,-40,-40, 0,0}; // (-) (Positive CW viewed from left)
//short int lkneem[7]={0, 0,  0,-40,  0, 0,0}; //(-)
//short int rkneem[7]={0, 0, 40, 40, 40, 0,0}; // (+)
//short int rhipzm[7]={-4,16, 16, 16, 16, -4,-4}; //do not change
//short int lhipzm[7]={4,24, 24, 24, 24, 4,4}; //do not change

//demo
//short int lhipxm[7]={0,45, 45, 45, 0,  40,0}; // (+) (Positive CW viewed from left)
//short int rhipxm[7]={0, 0,  0,  0, 0,  40,0}; // (-) (Positive CW viewed from left)
//short int lkneem[7]={0, 0,-30,-30, 0, -30,0}; //(-)
//short int rkneem[7]={0, 0,  0,  0, 0,  30,0}; // (+)
//short int rhipzm[7]={0, 0,  0,  0, 0, -30,0}; // - swing out
```

```
//short int lhipzm[7]={0, 0,  0,  45,0,  30,0}; //+ swing out

//don't do anything
//short int rhipzm[7]={0,0,0,0,0,0,0}; //do not change
//short int lhipzm[7]={0,0,0,0,0,0,0}; //do not change
//short int rhipzm[7]={0,0,0,0,0,0,0}; //do not change
//short int lhipzm[7]={0,0,0,0,0,0,0}; //do not change
//short int lhipzm[7]={0,0,0,0,0,0,0}; //swing out
//short int rhipzm[7]={0,0,0,0,0,0,0}; //swing out
short int ii;

speed[m_lknee]=1;
speed[m_lhipx]=1;
speed[m_lhipy]=1;
speed[m_lhipz]=1;
speed[m_rknee]=1;
speed[m_rhipx]=1;
speed[m_rhipy]=1;
speed[m_rhipz]=1;


move_motor(m_lhipx,lhipxm[mov_index]);
move_motor(m_lhipz,lhipzm[mov_index]);
move_motor(m_lknee,lkneem[mov_index]);
move_motor(m_rhipx,rhipxm[mov_index]);
move_motor(m_rhipz,rhipzm[mov_index]);
move_motor(m_rknee,rkneem[mov_index]);

/*if(mstate[m_lhipx]==0)
{if(mstate[m_lknee]==0)
{ldone=1;}
}
if(mstate[m_rhipx]==0)
{if(mstate[m_rknee]==0)
{rdone=1;}
}

if(ldone==1)
{if(rdone==1)
{mov_index=mov_index+1;
ldone=0;
rdone=0;
}
}*/

if(mstate[m_lhipx]==0)
{if(mstate[m_lknee]==0)
{if(mstate[m_lhipz]==0)
ldone=1;
}
}
if(mstate[m_rhipx]==0)
{if(mstate[m_rknee]==0)
{if(mstate[m_rhipz]==0)
rdone=1;
}
}

if(ldone==1)
{if(rdone==1)
{mov_index=mov_index+1;
ldone=0;
rdone=0;
}
}

if(mov_index==7)
{mov_index=0;}

RHipy=NEUTRAL-150;
//LHipz=NEUTRAL;
setPWMval();
}
/*****************************************************************************

                    KoBrain Specific Code
*****************************************************************************/
/*  KoBrain control code for the Marathon Walking 'Bot (MWB).
```

```
 Code which is not time-critical goes in the marked space in function main.
 Time-critical code, which must execute at specific (1 mS) time intervals,
 goes in function intRoutine_TMRC0 */




     // ------ Read AD7490 values, store in global 16-element array AD7490_Values
 void update_AD12(void)
 {
 #pragma interrupt called

     static short unsigned int count=0;
     short int data=0;
     short unsigned int address=0;


     if (*SPI0_SPSCR&0b0000000000001000) //Receive buffer full
     {
          *GPIOE_DR  |=   0b0000000010000000;     //Set E7 high, disabling AD7490
          data = *SPI0_SPSCR;                          //Read SPSCR
          data = *SPI0_SPDRR;                          //Read SPDRR, clear read buffer flag
          address = (unsigned short int)data>>12;
          data = data<<4;                              //Make 16-bit full scale
          AD7490_Values[address]=data;

          *GPIOE_DR   &=  0b1111111101111111; //Set E7 low, enabling AD7490

     }

     if (*SPI0_SPSCR&0b0000000000000001)     //Transmit buffer empty
     {
          count+=0x400;
          if (count>0x3C00)
          {
               count = 0x0;
          }

          *SPI0_SPDTR = (0b1000001101000000|count);    //Reload control register
     }


 }


     // ------ Timer C1 interrupt handler ------
 void intRoutine_TMRC1(void)
 {
 #pragma interrupt

     *TMRC1_SCR &= 0x7fff; //Clear timer compare flag (?)

     REDLED_ON;
     GRNLED_OFF;

     update_AD12();         //Get next analog input value from A/D converter

     REDLED_OFF;
 }

     // ------ TMRC3 encoder pulse timer interrupt routine
 //HIP timer
 void intRoutine_TMRC3(void)
 {
     #pragma interrupt

     static short prev_position = 0;

     Elapsed = *TMRC0_CNTR;
                          //Load start time of interrupt
     REDLED_ON;
     GRNLED_OFF;

     if (*TMRC2_CNTR - prev_position == 0x4)  //Positive velocity
     {
          TimeHip = *TMRC3_CAP;
          *TMRC3_SCR &= 0b0101011111111111;   //Clear interrupt flags
          *TMRC3_CNTR = 0x0;                       //Reset counter to zero
```

```
        *TMRC3_CTRL |= 0x2000;              //Restart counter C3
    }
    else if (prev_position - *TMRC2_CNTR == 0x4)
    {
        TimeHip = -(*TMRC3_CAP);
        *TMRC3_SCR &= 0b0101011111111111;   //Clear interrupt flags
        *TMRC3_CNTR = 0x0;                      //Reset counter to zero
        *TMRC3_CTRL |= 0x2000;              //Restart counter C3


}
    else
    {
        TimeHip = 0x7fff;                   //Direction change: max time
        *TMRC3_SCR &= 0b0101011111111111;   //Clear interrupt flags
        *TMRC3_CNTR = 0x0;                      //Reset counter to zero
        *TMRC3_CTRL |= 0x2000;              //Restart counter C3
    }

    prev_position = *TMRC2_CNTR;
    Elapsed = *TMRC0_CNTR - Elapsed;        //Calculate interrrupt time
    REDLED_OFF;

}
    //OUTER
    // ------ TMRD1 encoder pulse timer interrupt routine
    void intRoutine_TMRD1(void)
{
    #pragma interrupt

    static short prev_position = 0;

    Elapsed = *TMRC0_CNTR;
                        //Load start time of interrupt
    REDLED_ON;
    GRNLED_OFF;

    if (*TMRD0_CNTR - prev_position == 0x4)  //Positive velocity
    {
        TimeOuter = *TMRD1_CAP;
        *TMRD1_SCR &= 0b0101011111111111;   //Clear interrupt flags
        *TMRD1_CNTR = 0x0;                      //Reset counter to zero
        *TMRD1_CTRL |= 0x2000;              //Restart counter D1
    }
    else if (prev_position - *TMRD0_CNTR == 0x4)
    {
        TimeOuter = -(*TMRD1_CAP);
        *TMRD1_SCR &= 0b0101011111111111;   //Clear interrupt flags
        *TMRD1_CNTR = 0x0;                      //Reset counter to zero
        *TMRD1_CTRL |= 0x2000;              //Restart counter D1


}
    else
    {
        TimeOuter = 0x7fff;                     //Direction change: max time
        *TMRD1_SCR &= 0b0101011111111111;   //Clear interrupt flags
        *TMRD1_CNTR = 0x0;                      //Reset counter to zero
        *TMRD1_CTRL |= 0x2000;              //Restart counter D1
    }

    prev_position = *TMRD0_CNTR;
    Elapsed = *TMRC0_CNTR - Elapsed;        //Calculate interrrupt time
    REDLED_OFF;

}
    //INNER
    // ------ TMRD3 encoder pulse timer interrupt routine
    void intRoutine_TMRD3(void)
{
    #pragma interrupt

    static short prev_position = 0;

    Elapsed = *TMRC0_CNTR;
                        //Load start time of interrupt
    REDLED_ON;
    GRNLED_OFF;
```

```
    if (*TMRD2_CNTR - prev_position == 0x4)  //Positive velocity
    {
        TimeInner = *TMRD3_CAP;
        *TMRD3_SCR &= 0b0101011111111111;   //Clear interrupt flags
        *TMRD3_CNTR = 0x0;                      //Reset counter to zero
        *TMRD3_CTRL |= 0x2000;              //Restart counter D3
    }
    else if (prev_position - *TMRD2_CNTR == 0x4)//Negative velocity
    {
        TimeInner = -(*TMRD3_CAP);
        *TMRD3_SCR &= 0b0101011111111111;   //Clear interrupt flags
        *TMRD3_CNTR = 0x0;                      //Reset counter to zero
        *TMRD3_CTRL |= 0x2000;              //Restart counter D3


}
    else
    {
        TimeInner = 0x7fff;                     //Direction change: max time
        *TMRD3_SCR &= 0b0101011111111111;   //Clear interrupt flags
        *TMRD3_CNTR = 0x0;                      //Reset counter to zero
        *TMRD3_CTRL |= 0x2000;              //Restart counter D3
    }

    prev_position = *TMRD2_CNTR;
    Elapsed = *TMRC0_CNTR - Elapsed;        //Calculate interrrupt time
    REDLED_OFF;

}


// ------ PortA init routine, 0-7 digital inputs with pullups, 8 - 13 outputs --------
void init_GPIOA( void )
{
*GPIOA_PUR    = 0b0000000011111111;
*GPIOA_IAR    = 0x0000;
*GPIOA_IENR   = 0b0000000000000000;             //No interrupts
*GPIOA_IESR   = 0x0000;
*GPIOA_DDR    = 0b0011111100000000;             //8-13 are outputs
*GPIOA_PER    = 0b0000000000000000;             //shared peripherals disabled
*GPIOA_IPOLAR = 0b0000000000000000;
*GPIOA_PPMODE = 0b0011111111111111;             //Push-pull (totem-pole) outputs
}

// ------ PortB init routine, all outputs --------
void init_GPIOB( void )
{
*GPIOB_PUR    = 0b0000000000000000;
*GPIOB_IAR    = 0x0000;
*GPIOB_IENR   = 0b0000000000000000;             //No interrupts
*GPIOB_IESR   = 0x0000;
*GPIOB_DDR    = 0b0000000011111111;             //all 8 outputs
*GPIOB_PER    = 0b0000000000000000;             //shared peripherals disabled
*GPIOB_IPOLAR = 0b0000000000000000;
*GPIOB_PPMODE = 0b0011111111111111;             //Push-pull (totem-pole) outputs
}

// ------ PortD init routine --------
void init_GPIOD( void )
{
*GPIOD_PUR    = 0b0000000000101111;
*GPIOD_IAR    = 0x0000;
*GPIOD_IENR   = 0b0000000000000000;             //No interrupts
*GPIOD_IESR   = 0x0000;
*GPIOD_DDR    = 0b0000000000010000;             //all input except CMD (4)
*GPIOD_PER    = 0b0000000011000000;             //6&7 used for SCI1
*GPIOD_IPOLAR = 0b0000000000000000;
*GPIOD_PPMODE = 0b0011111111111111;             //Push-pull (totem-pole) outputs
*GPIOD_DR     = 0b0000000000010000;             //wireless CMD line high
}

// ------ PortE init routine --------
void init_GPIOE( void )
{
*GPIOE_PUR    = 0b1111111111111111;             //pullups enabled
*GPIOE_IAR    = 0x0000;
*GPIOE_IENR   = 0b0000000000000000;             //No interrupts
```

```
*GPIOE_IESR   = 0x0000;
*GPIOE_DDR    = 0b0000000010001100;             //E2, E3, E7 are outputs
*GPIOE_PER    = 0b0011111101110011;             //shared peripherals enabled
                                                    //except for ~SS and addresses
*GPIOE_IPOLAR = 0b0000000000000000;
*GPIOE_PPMODE = 0b0011111111111111;             //Push-pull (totem-pole) outputs
}

// ------ PortF init routine, All Outputs --------


// ------ SPI0 serial peripheral interface and AD7490 5V A/D init routine ------
void init_SPI0_AD(void)
{
     short int i;

     *SPI0_SPSCR =   0b0110000110000000;     //Baud rate 3.75MHz,MSB first,
                                                 //polarity 1, phase 0
     *SPI0_SPDSR =   0b0000000000001111;     //16-bit data words, no wired OR
     *SPI0_SPSCR |=    0b0000000000100000;       //Enable SPI

     *GPIOE_DR   |=  0b0000000010000000;     //Set E7 high
     *GPIOA_DR   &=  0b1111110111111111;     //Set A9 low, enabling onboard SPI0
     *GPIOA_DR   |=    0b0000010000000000;       //Set A10 high, disabling offboard SPI0

     *GPIOE_DR   &=  0b1111111101111111;     //Set E7 low, selecting AD7490
     *SPI0_SPDTR = 0x0000;                   //Send 0-value word to AD7490 for init
     while (!(*SPI0_SPSCR&0b0000000000001000)) //receive buffer empty
     {
          asm(NOP);                              //wait
     }
     *GPIOE_DR  |=   0b0000000010000000;     //Set E7 high, disabling AD7490
     i = *SPI0_SPSCR;                        //Read SPSCR
     i = *SPI0_SPDRR;                        //Read SPDRR, clear read buffer flag



     for (i=0;i<1000;i++)
     {
          asm(NOP);                          //wait
     }

     *GPIOE_DR   &=  0b1111111101111111; //Set E7 low, enabling AD7490
     *SPI0_SPDTR =   0b1000001101000000; //Initialize AD7490 control reg.
     while (!(*SPI0_SPSCR&0b0000000000001000)) //receive buffer empty
     {
          asm(NOP);                              //wait
     }
     *GPIOE_DR  |=   0b0000000010000000;     //Set E7 high, disabling AD7490
     i = *SPI0_SPSCR;                        //Read SPSCR
     i = *SPI0_SPDRR;                        //Read SPDRR, clear read buffer flag


}

// ------ Timer C0 with compare interrupt enabled and 1KHz interrupt rate ------
void init_TMRC0( void )
{
*TMRC0_CMP1 = 60000;           //Count to 60000, interrupt and reset counter
*TMRC0_CMP2 = 0x0000;
*TMRC0_CAP  = 0x0000;
*TMRC0_LOAD = 0x0000;
*TMRC0_HOLD = 0x0000;
*TMRC0_CNTR = 0x0000;
*TMRC0_CTRL = 0b0011000000100000;     //60 MHz clock, count to cmp1 and reset
*TMRC0_SCR  = 0x4000;                 //Interrupt on successful compare
}

// ------ Timer C1 with compare interrupt enabled and 30KHz interrupt rate ------
void init_TMRC1( void )
{
*TMRC1_CMP1 = 2000;            //Count to 2000, interrupt and reset counter
*TMRC1_CMP2 = 0x0000;
*TMRC1_CAP  = 0x0000;
*TMRC1_LOAD = 0x0000;
*TMRC1_HOLD = 0x0000;
*TMRC1_CNTR = 0x0000;
*TMRC1_CTRL = 0b0011000000100000;     //60 MHz clock, count to cmp1 and reset
```

```
*TMRC1_SCR  = 0x4000;                //Interrupt on successful compare
}

// ------ Timer C2, set as quad decoder  ------
void init_TMRC2( void )
{
*TMRC2_CMP1 = 0xffff;
*TMRC2_CMP2 = 0x0000;
*TMRC2_CAP  = 0x0000;
*TMRC2_LOAD = 0x0000;
*TMRC2_HOLD = 0x0000;
*TMRC2_CNTR = 0x0000;
*TMRC2_CTRL = 0b1000000010000010;
*TMRC2_SCR  = 0x0000;
}

// ------ Timer C3 with 1.875 MHz clock, compare and capture interrupt ------
void init_TMRC3( void )
{
*TMRC3_CMP1 = 0x7fff;               //Count to 7fff
*TMRC3_CMP2 = 0x0000;
*TMRC3_CAP  = 0x0000;
*TMRC3_LOAD = 0x0000;
*TMRC3_HOLD = 0x0000;
*TMRC3_CNTR = 0x0000;
*TMRC3_CTRL = 0b0011101001000000;   //1.875 MHz clock, count to cmp1 and stop
*TMRC3_SCR  = 0b0100010001000000;   //Interrupt on compare, positive-going input edges
}

// ------ Timer D0, set as quad decoder ------
void init_TMRD0( void )
{
*TMRD0_CMP1 = 0xffff;
*TMRD0_CMP2 = 0x0000;
*TMRD0_CAP  = 0x0000;
*TMRD0_LOAD = 0x0000;
*TMRD0_HOLD = 0x0000;
*TMRD0_CNTR = 0x0000;
*TMRD0_CTRL = 0b1000001000000010;
*TMRD0_SCR  = 0x0000;
}

// ------ Timer D1 with interrupt and 1.875 MHz clock ------
void init_TMRD1( void )
{
*TMRD1_CMP1 = 0x7fff;               //Count to 7fff
*TMRD1_CMP2 = 0x0000;
*TMRD1_CAP  = 0x0000;
*TMRD1_LOAD = 0x0000;
*TMRD1_HOLD = 0x0000;
*TMRD1_CNTR = 0x0000;
*TMRD1_CTRL = 0b0011101001000000;   //1.875 MHz clock, count to cmp1 and stop
*TMRD1_SCR  = 0b0100010001000000;   //Interrupt on compare, positive-going input edges
}

// ------ Timer D2, set as quad decoder ------
void init_TMRD2( void )
{
*TMRD2_CMP1 = 0xffff;
*TMRD2_CMP2 = 0x0000;
*TMRD2_CAP  = 0x0000;
*TMRD2_LOAD = 0x0000;
*TMRD2_HOLD = 0x0000;
*TMRD2_CNTR = 0x0000;
*TMRD2_CTRL = 0b1000010110000010;
*TMRD2_SCR  = 0x0000;
}

// ------ Timer D3 with interrupt and 1.875 MHz clock ------
void init_TMRD3( void )
{
*TMRD3_CMP1 = 0x7fff;               //Count to 7fff
*TMRD3_CMP2 = 0x0000;
*TMRD3_CAP  = 0x0000;
*TMRD3_LOAD = 0x0000;
*TMRD3_HOLD = 0x0000;
*TMRD3_CNTR = 0x0000;
*TMRD3_CTRL = 0b0011101101000000;   //1.875 MHz clock, count to cmp1 and stop
```

```
*TMRD3_SCR  = 0b0100010001000000;   //Interrupt on compare, positive-going input edges
}

// ------ initialize PWMA ------
void init_PWMA()
{
     short temp;
     *PWMA_PWMCM =      0x7FFF;   //1500 for 20KHz; 0x7FFF for 114 Hz with 1/8 prescale
     *PWMA_PMDEADTM =  0x0000; //No deadtime
     *PWMA_PWMVAL0 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMA_PWMVAL1 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMA_PWMVAL2 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMA_PWMVAL3 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMA_PWMVAL4 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMA_PWMVAL5 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMA_PMDISMAP1 = 0b0000000000000000; //Disable all fault inputs
     *PWMA_PMDISMAP2 = 0b0000000000000000; //Disable all fault inputs
     *PWMA_PMCFG =      0b0010000000001110; //Center-aligned, positive polarity, independent
     *PWMA_PMCCR =      0b0000000000000000; //non-masked, independent PWM values, no channel swaps
     *PWMA_PMICCR =     0x0000;
     *PWMA_PMFCTL =     0b0000000000000000; //No interrupts, manual fault clear
     *PWMA_PMOUT =      0b0000000000000000; //Disable output pads for now; PWM control of outputs
     *PWMA_PMCTL =      0b0000100011000000; //Disable PWM outputs for now, update on half cycles,          ¬
prescaler 1/8
     temp = *PWMA_PMCTL|0x0002; //Read LDOK
     *PWMA_PMCTL = *PWMA_PMCTL|0x0002; //Set LDOK
     *PWMA_PMCTL = *PWMA_PMCTL|0x0001; //Enable PWM outputs
     *PWMA_PMOUT = *PWMA_PMOUT|0x8000; //Enable PWM output pads
}

// ------ initialize PWMB ------
void init_PWMB()
{
     short temp;
     *PWMB_PWMCM =      0x7FFF;   //1500 for 20KHz; 0x7FFF for 114 Hz with 1/8 prescale
     *PWMB_PMDEADTM =  0x0000; //No deadtime
     *PWMB_PWMVAL0 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMB_PWMVAL1 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMB_PWMVAL2 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMB_PWMVAL3 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMB_PWMVAL4 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMB_PWMVAL5 =   0x0000; //Initial PWM duty cycle = 0% at 20 KHz
     *PWMB_PMDISMAP1 = 0b0000000000000000; //Disable all fault inputs
     *PWMB_PMDISMAP2 = 0b0000000000000000; //Disable all fault inputs
     *PWMB_PMCFG =      0b0010000000001110; //Center-aligned, positive polarity, independent
     *PWMB_PMCCR =      0b0000000000000000; //non-masked, independent PWM values, no channel swaps
     *PWMB_PMICCR =     0x0000;
     *PWMB_PMFCTL =     0b0000000000000000; //No interrupts, manual fault clear
     *PWMB_PMOUT =      0b0000000000000000; //Disable output pads for now; PWM control of outputs
     *PWMB_PMCTL =      0b0000100011000000; //Disable PWM outputs for now, update on half cycles,          ¬
prescaler 1/8
     temp = *PWMB_PMCTL|0x0002; //Read LDOK
     *PWMB_PMCTL = *PWMB_PMCTL|0x0002; //Set LDOK
     *PWMB_PMCTL = *PWMB_PMCTL|0x0001; //Enable PWM outputs
     *PWMB_PMOUT = *PWMB_PMOUT|0x8000; //Enable PWM output pads
}

// Update PWMA - update the PWM output pulse characteristics to reflect PWM_VAL, etc.
void update_PWMA(void)
{
     short temp;

     temp = *PWMA_PMCTL|0x0002; //Read LDOK
     *PWMA_PMCTL = *PWMA_PMCTL|0x0002; //Set LDOK
}

// Update PWMB - update the PWM output pulse characteristics to reflect PWM_VAL, etc.
void update_PWMB(void)
{
     short temp;

     temp = *PWMB_PMCTL|0x0002; //Read LDOK
     *PWMB_PMCTL = *PWMB_PMCTL|0x0002; //Set LDOK
}

// ------ ADCA analog-to-digital converter init routine ------
void init_ADCA( void )
{
```

```
     *ADCA_ADCR1 =   0b0000000000000001;
     *ADCA_ADCR2 =   0b0000000000001001; //2 MHz ADC clock
     *ADCA_ADZCC =   0b0000000000000000; //Zero crossing disabled
     *ADCA_ADLST1 =  0b0011001000010000; //Default channel order 0,1,2,3
     *ADCA_ADLST2 =  0b0111011001010100; //Default channel order 4,5,6,7
     *ADCA_ADSDIS =  0b0000000000000000; //All channels enabled, test mode off
     *ADCA_ADLLMT0 = 0b0000000000000000; //Low limit checking disabled
     *ADCA_ADLLMT1 = 0b0000000000000000; //Low limit checking disabled
     *ADCA_ADLLMT2 = 0b0000000000000000; //Low limit checking disabled
     *ADCA_ADLLMT3 = 0b0000000000000000; //Low limit checking disabled
     *ADCA_ADLLMT4 = 0b0000000000000000; //Low limit checking disabled
     *ADCA_ADLLMT5 = 0b0000000000000000; //Low limit checking disabled
     *ADCA_ADLLMT6 = 0b0000000000000000; //Low limit checking disabled
     *ADCA_ADLLMT7 = 0b0000000000000000; //Low limit checking disabled
     *ADCA_ADHLMT0 = 0b1111111111111111; //High limit checking disabled
     *ADCA_ADHLMT1 = 0b1111111111111111; //High limit checking disabled
     *ADCA_ADHLMT2 = 0b1111111111111111; //High limit checking disabled
     *ADCA_ADHLMT3 = 0b1111111111111111; //High limit checking disabled
     *ADCA_ADHLMT4 = 0b1111111111111111; //High limit checking disabled
     *ADCA_ADHLMT5 = 0b1111111111111111; //High limit checking disabled
     *ADCA_ADHLMT6 = 0b1111111111111111; //High limit checking disabled
     *ADCA_ADHLMT7 = 0b1111111111111111; //High limit checking disabled
     *ADCA_ADOFS0  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCA_ADOFS1  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCA_ADOFS2  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCA_ADOFS3  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCA_ADOFS4  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCA_ADOFS5  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCA_ADOFS6  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCA_ADOFS7  = 0b0000000000000000; //Zero offset adjustment - pos. output
}

// ------ ADCB analog-to-digital converter init routine ------
void init_ADCB( void )
{
     *ADCB_ADCR1 =   0b0000000000000001;
     *ADCB_ADCR2 =   0b0000000000001001; //2 MHz ADC clock
     *ADCB_ADZCC =   0b0000000000000000; //Zero crossing disabled
     *ADCB_ADLST1 =  0b0011001000010000; //Default channel order 0,1,2,3
     *ADCB_ADLST2 =  0b0111011001010100; //Default channel order 4,5,6,7
     *ADCB_ADSDIS =  0b0000000000000000; //All channels enabled, test mode off
     *ADCB_ADLLMT0 = 0b0000000000000000; //Low limit checking disabled
     *ADCB_ADLLMT1 = 0b0000000000000000; //Low limit checking disabled
     *ADCB_ADLLMT2 = 0b0000000000000000; //Low limit checking disabled
     *ADCB_ADLLMT3 = 0b0000000000000000; //Low limit checking disabled
     *ADCB_ADLLMT4 = 0b0000000000000000; //Low limit checking disabled
     *ADCB_ADLLMT5 = 0b0000000000000000; //Low limit checking disabled
     *ADCB_ADLLMT6 = 0b0000000000000000; //Low limit checking disabled
     *ADCB_ADLLMT7 = 0b0000000000000000; //Low limit checking disabled
     *ADCB_ADHLMT0 = 0b1111111111111111; //High limit checking disabled
     *ADCB_ADHLMT1 = 0b1111111111111111; //High limit checking disabled
     *ADCB_ADHLMT2 = 0b1111111111111111; //High limit checking disabled
     *ADCB_ADHLMT3 = 0b1111111111111111; //High limit checking disabled
     *ADCB_ADHLMT4 = 0b1111111111111111; //High limit checking disabled
     *ADCB_ADHLMT5 = 0b1111111111111111; //High limit checking disabled
     *ADCB_ADHLMT6 = 0b1111111111111111; //High limit checking disabled
     *ADCB_ADHLMT7 = 0b1111111111111111; //High limit checking disabled
     *ADCB_ADOFS0  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCB_ADOFS1  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCB_ADOFS2  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCB_ADOFS3  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCB_ADOFS4  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCB_ADOFS5  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCB_ADOFS6  = 0b0000000000000000; //Zero offset adjustment - pos. output
     *ADCB_ADOFS7  = 0b0000000000000000; //Zero offset adjustment - pos. output
}



// Enable interrupts in status register and for selected peripheral modules;
// Note that enabling interrupts also requires setting specific peripheral interrupt enable bits,
// and including the interrupt handler name in the interrupt vector file.
void init_interrupts(void)
{

     *ITCN_IPR6 = *ITCN_IPR6|0b0110001000000000;  //Set priority to 0 for TMRC0
                                                  //Set priority to 1 for TMRD1
                                                  //Set priority to 1 for TMRD3
```

```
     *ITCN_IPR7 = *ITCN_IPR7|0b0000000000100001; //Set priority to 1 for TMRC3
                                                 //Set priority to 0 for TMRC1

     asm
     {
          BFCLR #$0300,SR    //Enable interrupts

     }
}

/* Parse one incoming byte, if available, on serial port 1, and build up a long (32 bit)
word at 7 bits per incoming byte, most significant to least. Discard the three most
significant bits. Valid data bytes are in the range of 0 to 127 decimal
(0b00000000 to 0b01111111); 192 to 255 (0b11000000 to 0b11111111) are valid parameter
addresses. Upon receiving a valid address byte, the routine begins looking for valid
data bytes. If 5 successive valid data bytes are received, split the long word into
two 16-bit words, and save into two successive positions in global array Parameter. */
void Rec_Data(void)
{
     static unsigned char rbytenum = 0, rlintlabel = 0;
     static unsigned long int rcurrentdata = 0;
     unsigned char temp = 0;
     unsigned long int new_data;

#ifdef WIRELES

     if (*SCI1_SCISR&0x2000)  //Receive buffer full - process new byte
     {
          new_data = *SCI1_SCISR&0x2000; //Read status register buffer full flag
          new_data = *SCI1_SCIDR;    //Read data register, resetting flag

#else

     if (*SCI0_SCISR&0x2000)  //Receive buffer full - process new byte
     {
          new_data = *SCI0_SCISR&0x2000; //Read status register buffer full flag
          new_data = *SCI0_SCIDR;    //Read data register, resetting flag

#endif

          if ((new_data>=160)&&(new_data<=255)&&(rbytenum==0))  //New byte is a parm. address
          {
               rlintlabel = new_data - 160;
               temp = rlintlabel;
               rcurrentdata = 0x0;
               rbytenum++;             //Start looking for data bytes
          }
          else if ((new_data>=0x0)&&(new_data<=0x7f)&&(rbytenum>=0x1)&&(rbytenum<=0x5)) //new byte is
valid data
          {
               if (rbytenum == 0x1)      //trim off all but lowest 4 bits
               {
                    new_data&=0xf;
               }
               temp = new_data;
               new_data<<=((5 - rbytenum) * 7);
               rcurrentdata|=new_data;
               rbytenum++;
               if (rbytenum == 0x6)       //parsing of new long word is complete
               {
                    GBL_Param[rlintlabel] = rcurrentdata; //Store new data in ffloat array GBL_Param
                    rcurrentdata = 0x0;                   //and start looking for new label byte
                    rbytenum = 0x0;
               }
          }
          else                          //Data is not valid - start over, looking for new label byte
          {
               rcurrentdata = 0x0;
               rbytenum = 0x0;
          }
     }


#ifdef WIRELES

     if (*SCI1_SCISR&0b0000111100000000) //Receive error
     {
          new data = *SCI1 SCISR;             //Clear flags
```

```
          *SCI1_SCISR = 0;
     }
}

#else

     if (*SCI0_SCISR&0b0000111100000000) //Receive error
     {
          new_data = *SCI0_SCISR;             //Clear flags
          *SCI0_SCISR = 0;
     }
}

#endif


/* Call send_data at controlled intervals to send out three types of data: 8 selected high-
speed data channels, at a rate such that all eight values go out every 16 ms;
16 selected low-speed data channels, one every 16 mS, thus sending all 16 every 128 ms;
and one parameter value update confirmation every 16 mS, checking the full range
of parameters for changes every 128 mS. */
void Data_Sequencer(void)
{
     static unsigned char SendBusy = 0,Address = 0, ParmSend= 0;
     static unsigned int PrevTimeSeg = 0, Index = 0;
     static ffloat Data = 0;
     unsigned int timeseg,i,t32,test1,test2;


     timeseg = (GBL_Elapsed_mS & 0x0000007F)>>4; //8 numbered time segments of 16 mS each
     if (timeseg != PrevTimeSeg)                 //New time segment
     {
          PrevTimeSeg = timeseg;
          Index = 0;                             //Start sending new data sequence

          if (SendBusy) //Data transmission must finish before the next time segment
          {
               REDLED_ON;   //Data overrun
          }

     }

#ifdef WIRELES

     //if (*SCI1_SCISR&0x8000 && !(*GPIOB_DR&0x0040)) //Transmitter empty and CTS low
     //if (*SCI1_SCISR&0x8000 && !(*GPIOD_DR&0x0020))      //Transmitter empty and CTS low
     if (*SCI1_SCISR&0x8000)        //Transmitter empty ****test code

#else

     if (*SCI0_SCISR&0x8000)    //Transmitter empty

#endif

     {
          test1 = Index;
          test2 = PrevTimeSeg;
          if (!SendBusy)
          {
               switch (Index)
               {
                    case 0:           //Send first in time segment
                    {
                         Address = 128;
                         Data = GBL_Elapsed_mS;

                         break;
                    }

                    case 1:
                    {
                         Address = 129;
                         Data = GBL_Data[GBL_Param[0]];

                         break;
                    }

                    case 2:
```

```
                    {
                         Address = 130;
                         Data = GBL_Data[GBL_Param[1]];

                         break;
                    }

                    case 3:
                    {
                         Address = 131;
                         Data = GBL_Data[GBL_Param[2]];

                         break;
                    }

                    case 4:
                    {
                         Address = 132 + timeseg;
                         Data = GBL_Data[GBL_Param[3+timeseg]];

                         break;
                    }

                    case 5:                         //Send last in time segment
                    {
                         Address = 0;
                         t32 = timeseg<<5;


                         if (timeseg < 3)
                         {
                              //Check for updated parameter values in 32-element range
                              for (i=0;i<32;i++)
                              {
                                   if (GBL_PrevParam[t32+i]!=GBL_Param[t32+i])
                                   {
                                        Address = 160+t32+i;
                                        Data = GBL_Param[t32+i];
                                        GBL_PrevParam[t32+i] = GBL_Param[t32+i];
                                   }
                              }
                         }


                         //Update a parameter value even if no apparent change
                         if (Address == 0)
                         {
                              Address = 160+ParmSend;
                              Data = GBL_Param[ParmSend];
                              GBL_PrevParam[ParmSend] = GBL_Param[ParmSend];
                              ParmSend++;
                              if (ParmSend == 96)
                              {
                                   ParmSend = 0;
                              }
                         }
                         break;
                    }

               }
          }

          if ((Index >=0)&&(Index <= 5))
          {
               SendBusy = Send_Data(Address,Data);
               if (SendBusy == 6)
               {
                    SendBusy = 0;
                    Index++;
               }
          }
     }
}

/* Using global array elements Variable[0] and Variable[1], maintain a 30-bit count
of elapsed mS since startup. */
void timestamp(void)
{
```

```
     Variable[0]++;
     if (Variable[0] == -32768)
     {
          Variable[0] = 0;
          Variable[1]++;
     }
}

/* Combine pairs of 16-bit words into 32 bit words, then into strings of 6 bytes,
and send them out serial port 1. The first byte tranmitted is an address byte, with
valid values between 128 and 255. Next is a byte containing the
most significant 4 bits of the long word in its lower four bits. Finally, the least
significant 28 bits of the long word are sent out 7 bits at a time, in four transmit
bytes. Valid values for the 5 data-containing bytes are 0 to 127.

Send each value in the array and repeat; one byte is transmitted each time this routine
is called.*/
unsigned int Send_Data(unsigned int address,ffloat data)
{
     static unsigned int sbytenum = 0;
     static unsigned long int scurrentdata = 0;
     unsigned long int ltemp;
     unsigned int index = 0, temp;
     void *VoidPointer;
     unsigned long int *LongPointer;


#ifdef WIRELES

//   if (*SCI1_SCISR&0x8000 && !(*GPIOB_DR&0x0040))    //Transmitter empty and CTS low
//   if (*SCI1_SCISR&0x8000 && !(*GPIOD_DR&0x0020))        //Transmitter empty and CTS low
     if (*SCI1_SCISR&0x8000)         //Transmitter empty  ***Test Code
     {

          //Load long int data into unsigned long int scurrentdata for sending.
          if ((sbytenum == 0)||(sbytenum == 6))
          {
               temp = *SCI1_SCISR;                 //read status register
               *SCI1_SCIDR = address;          //send address byte

               VoidPointer = &(data);     //obtain pointer to long int data
               LongPointer = VoidPointer;      //convert from signed to unsigned type
               scurrentdata = *LongPointer;
               sbytenum = 1;                   //ready to send data bytes
          }


          else if ((sbytenum >= 1)&&(sbytenum <= 5))  //need to send data byte
          {
               //Shift right until only desired byte number remains. No sign extension!
               ltemp = scurrentdata>>((5 - sbytenum) * 7);
               temp = *SCI1_SCISR;          //read status register
               *SCI1_SCIDR = ltemp;       //send data byte
               if (sbytenum == 1)
               {
                    scurrentdata &= 0x0fffffff;         //Clear top four bits
               }
               else
               {
                    //Clear next 7 most significant bits
                    scurrentdata &= (0x0fffffff>>((sbytenum - 1) * 7));
               }
               sbytenum++;

          }

          else
          {
               sbytenum = 0;
          }
     }
     return(sbytenum);
}


#else

     if (*SCI0 SCISR&0x8000)    //Transmitter empty
```

```
     {

          //Load long int data into unsigned long int scurrentdata for sending.
          if ((sbytenum == 0)||(sbytenum == 6))
          {
               temp = *SCI0_SCISR;                 //read status register
               *SCI0_SCIDR = address;          //send address byte

               VoidPointer = &(data);     //obtain pointer to long int data
               LongPointer = VoidPointer;      //convert from signed to unsigned type
               scurrentdata = *LongPointer;
               sbytenum = 1;                   //ready to send data bytes
          }


          else if ((sbytenum >= 1)&&(sbytenum <= 5))  //need to send data byte
          {
               //Shift right until only desired byte number remains. No sign extension!
               ltemp = scurrentdata>>((5 - sbytenum) * 7);
               temp = *SCI0_SCISR;         //read status register
               *SCI0_SCIDR = ltemp;       //send data byte
               if (sbytenum == 1)
               {
                    scurrentdata &= 0x0fffffff;         //Clear top four bits
               }
               else
               {
                    //Clear next 7 most significant bits
                    scurrentdata &= (0x0fffffff>>((sbytenum - 1) * 7));
               }
               sbytenum++;

          }

          else
          {
               sbytenum = 0;
          }
     }
     return(sbytenum);
}
#endif



void init_SCI0(void)
{
     //Baud rate register
     //*SCI0_SCIBR = 391; //9600 baud
          *SCI0_SCIBR = 33; //115 Kbaud (use decimal 391 for 9600) at 60 MHz clock
     //Control register
     //*SCI0_SCICR = 0x000C; //Transmit enable, receive enable, 8 bit, NP, 1 stop
       *SCI0_SCICR = 0b0000010000001100; //inverted, transmit enable, receive enable, 8 bit, NP, 1 ¬
stop
    //Data register
    *SCI0_SCIDR = 0x0000;
}

void init_SCI1(void)
{
     //Baud rate register
     *SCI1_SCIBR = 33; //115 Kbaud (use decimal 391 for 9600) at 60 MHz clock

     //Control register
//   *SCI1_SCICR = 0b0000000000001100; //Transmit enable, receive enable, 8 bit, NP, 1 stop

     //Control register
     *SCI1_SCICR = 0b0000010000001100; //Inverted, Transmit enable, receive enable, 8 bit, NP, 1 ¬
stop

    //Data register
    *SCI1_SCIDR = 0x0000;
}



/*****************************************************************************************
*********************                                    ************************************
```

```
**********************              MAIN LOOP           ************************************
**********************                                  ************************************
*****************************************************************************************/
     // ------ Timer C0 interrupt handler ------
void intRoutine_TMRC0(void)
{
     short int q=0;
     #pragma interrupt saveall  //DO NOT ERASE




     *TMRC0_SCR &= 0x7fff;      //DO NOT ERASE    //Clear timer compare flag
     GBL_Elapsed_mS++;          //Increments elapsed time counter
     timestamp();               //Increments elapsed time counter (Variable[0] and [1])

// Begin Initial State
  //My code HERE
  /*update_Sensor();
  update_Theta();
  control_Output();
  motor_Control();*/
  calsec++;

     //Gyroscope values + length4 running avg filter
     Variable[6]=AD7490_Values[13];
     Variable[7]=AD7490_Values[14];
     Variable[8]=AD7490_Values[15];

     for(q=0;q<3;q++)//loop to run for x,y,z
     {
          //Variable[6+q]=co[q];
          acc[q]+=(AD7490_Values[13+q]+co[q]);//+co[q])>>2;//inter[q]>>2;
          Variable[3+q]=6.67*(acc[q]>>16);
     }
  if(calidone==1)
  {
  //AttitudeEstimator();
 conv_wireless();
  //testimator();
  err_biped();
  //motor_pend();
  pend_step();
  //movements();
  //go_neutral();
  }

 // test_motor4();

// run a function that updates the motor stuff
}

/*****************************************************************************************
*****************************************************************************************/

void main(void)  //Run initialization routines, then enter untimed endless loop.
{

     //Init_Wireless();
     init_interrupts();
     init_TMRC0();
     init_TMRC1();
     init_TMRC2();
     init_TMRC3();
     init_TMRD0();
     init_TMRD1();
     init_TMRD2();
     init_TMRD3();
     init_GPIOA();
     init_GPIOB();
     init_GPIOD();
     init_GPIOE();

     init_SCI0();
     init_SCI1();
     init_SPI0_AD();
     init_ADCA();
     init_ADCB();
```

```
     init_PWMA();
     init_PWMB();
     //Init_Wireless();
     Variable[0] = 0;       //Initialize elapsed time counter to zero at startup
     Variable[1] = 0;

     GRNLED_OFF;
     REDLED_OFF;
     YELLED_OFF;
     init_parameters();
     init_gyro();

     //initialize default values for Parameters

     //Initialize elapsed time counter to zero at startup
     GBL_Elapsed_mS = 0;

//
     while (1)                  //Main endless loop
     {
     //Calibration
     if(calibrate<13 && calsec>=500)
     {
        YELLED_ON;
        calsec=0;
        cali_gyro();
     }



     Data_Sequencer();      //Send out data from Variable array as requested from external computer
     Rec_Data();
              //Receive and parse incoming parameter data, put in Parameter global array
     GRNLED_ON;                  //Green LED on for main loop, red for interrupt


    /* wait until next interrupt
     asm(wait); */

     }
}



/* ----- end of KoBrain control framework ------ */
```