

```

/*
This program is for use with the Wobbler RadFx Antenna Pointing System (WRAPS). This project was developed to provide an
affordable alternative to the Yaesu G5500 rotor system. The WRAPS is intended to be used as a portable rotor system,
operating with 12 volt batteries, USB connection to a computer running SatPC32 with EASYCOM protocol. The WRAPS is designed
to handle light-weight yagi antennas such as the ARROW or the Elk antenna. It is not weather proof or weather tight,
it is not intended for 24/7 operation. It is designed to be used with the operator in close proximity and within sight
of the antenna (at the on/off switch in the event of a stalled motor). The system was designed primarily for schools, teachers,
and/or students who want to access the telemetry transmitted by the Fox1A or Fox1B CubeSats.

```

```

The WRAPS rotor is configured for 0 to 359 degrees AZ and 0 to 90 degrees EL. Set up SatPC32 with the turn point north,
and elevation 0-90 degrees (this limitation is to compensate for the torque required to handle the antenna from the
hand-hold end, the antenna in the WRAPS is mounted so that it is approximately balanced). The system runs at 9600 baud,
and the on-board USB chip is the FTDI 232 chip (standard drivers available for WINDOWS).

```

```

*/

```

```

#include <htc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

```

```

// #define _XTAL_FREQ 32000000L
#define _XTAL_FREQ 4000000L
#define testbit(var,bit) ((var&(1<<bit)))
#define setbit(var,bit) ((var|= (1<<bit)))
#define clrbit(var,bit) ((var &=~(1<<bit)))
#define LOWBYTE(v) ((unsigned char) (v))
#define HIGHBYTE(v) ((unsigned char) (((unsigned int) (v)) >> 8))

```

```

__CONFIG(FOSC_INTOSC & WDTE_OFF & PWRTE_ON & MCLRE_OFF & CP_OFF & CPD_OFF &
BOREN_ON & CLKOUTEN_OFF & IESO_ON & FCMEN_ON);
__CONFIG(WRT_OFF & PLLEN_OFF & STVREN_ON & LVP_OFF);
/*
__CONFIG(FOSC_INTOSC & WDTE_SWDTEN & PWRTE_ON & MCLRE_OFF & CP_OFF & CPD_OFF &
BOREN_ON & CLKOUTEN_OFF & IESO_ON & FCMEN_OFF);
__CONFIG(WRT_OFF & PLLEN_OFF & STVREN_ON & BORV_19 & LVP_OFF);
*/

```

```

__EEPROM_DATA(0x00,0x31,0x03,0xdc,0x01,0x14,0x02,0xc7);
/* Defaults
AZ 0 deg in 0,1 = 0x0031
AZ 359 deg in 2,3 = 0x03dc
EL 0 deg in 4,5 = 0x0114
EL 90 deg in 6,7 = 0x02c7
*/

```

```

//declarations
void AZ_stop(void);
void EL_stop(void);
void get_cal(void);

```

```

void cal(void);
int get_ADC(char channel);
void RX_cmd(void);
void AZ_inc(int z, int AZ_cmd);
void AZ_dec(int z, int AZ_cmd);
//void AZ_ctr(int z);
void EL_up (int z, int EL_cmd);
void EL_down (int z, int EL_cmd);
void EL_up_manual (int z);
void EL_down_manual (int z);
void AZ_up_manual (int z);
void AZ_down_manual (int z);

float x, y, zf;
int i, speed, old_speed, temp_speed, EL_ADC, AZ_ADC, old_ADC, new_ADC,z, pot_ctr;
int AZ_range, EL_range, old_AZ_ADC, old_EL_ADC, new_AZ_ADC, new_EL_ADC, AZ_ADC_0; AZ_ADC_359, EL_ADC_0, EL_ADC_90;
int AZ_range,EL_range;
int AZ_max, AZ_min, EL_max, EL_min, AZ_deg, EL_deg;
char temp,cmd,dir, new_dir, channel, manual;
#define EL 1 //set EL ADC channel to AN1 CHS=00001
#define AZ 0 //set AZ ADC channel to AN0 CHS=00000
#define AZ_max_ADC 1002 //set AZ max ADC value
#define AZ_min_ADC 5 //set AZ minimum ADC Value
#define EL_max_ADC 732 //set AZ max ADC value to prevent over/undershoot
#define EL_min_ADC 231 //set AZ minimum ADC Value

void init(void)
{
    OSCCON = 0b01101000; //4 MHz
    PORTB=0;
    PORTA=0;

    //port setup
    // ANSELA = 0b00000011; //RA0 RA1 analog set up later
    TRISA = 0b00000011; //RA4 is data LED need output on this pin
    ANSA4=0; //RA4 output digital
    ANSELB = 0b00000000; //all digital
    WPUA5 = 1; //enable weak pull up on RA5
    WPUB = 0b11011100; //set weak pull up on B port 7, 6, 4, 3, 2
    nWPUEN = 0; //enable weak pull ups

    //alternative pin selection
    RXDTSEL = 0; //Rx on RB1
    TXCKSEL = 0; //Tx on RB2
    P2BSEL = 1; //P2B on RRA
    CCP2SEL = 1; //P2A on RA7
    CCP1SEL = 1; //P1A on RB0
    //P1B on RB5

    //ADC setup

```

```

ANSA0 = 1;
TRISA0 = 1;
ANSA1 = 1;
TRISA1 = 1;
ADCON1 = 0b11000000;
ADCON0 = 0b00000001;

//RA0 analog AZ ADC
//RA0 input
//RA1 analog EL ADC
//RA1 input
//R justified, Fosc/4, Vdd and Gnd on Vref
//AN0 on
//AN1 on = 0b00000101

//PWM setup
TRISB0 = 1;
TRISB5 = 1;
TRISA2 = 0;

//disable CCP1
//0=I/O output, 1= input
//make AZ control (RB7) line digital output

TRISA7 = 1;
TRISA6 = 1;
TRISA3 = 0;

//disable CCP2

//make EL control (RB6) line digital output

PR4= 0xff;
PR2= 0xff;
CCP1CON = 0b00001100;
CCPR1L=128;
CCP2CON = 0b00001100;
CCPR2L=128;
C1TSEL0 = 0;
C1TSEL1 = 0;
C2TSEL0 = 1;
C2TSEL1 = 0;

//PWM freq 31.25 kHz with 4 MHz clock

//CCP1 PWM mode, DC1B = 00, active high
//50% duty cycle (will change with speed)
//CCP2 PWM mode, DC2V = 00, active high
//50% duty cycle (will change with speed)
//CCP1 based off Timer 2

//CCP2 based off Timer 4

TMR2IF = 0;
T2CKPS0 = 0;
T2CKPS1 = 0;
TMR2ON = 1;

//clear TMR2 interrupt flag
//pre scale set to 1

//turn on TMR2

TMR4IF = 0;
T4CKPS0 = 0;
T4CKPS1 = 0;
TMR4ON = 1;

//clear TMR4 interrupt flag
//pre scale set to 1

//turn on TMR2

//set up motor drivers to off
RA3=0;
RA2 = 0;
STR2B=0;
STR2A=0;
CCP2CONbits.DC2B =0;
CCPR2L = 0;
STR1B=0;
STR1A=0;
CCP1CONbits.DC1B = 0;//get lsb or speed an put in DC1B
CCPR1L = 0;

//stop EL motor
//stop AZ motor
//make RA6 I/O
//put PWM on RA7
//get lsb or speed an put in DC1B
//get msbs and put in CCPR1L
//make RB5 I/O
//make RB0 PWM

//steering mode CCP1

```

```

        TRISB0 = 0;           //enable PA1 and PB1
        TRISB5 = 0;
//steering mode CCP2
        TRISA7 = 0;           //enable PA2 and PB2
        TRISA6 = 0;
//set up USART
        SPBRGH = 1;
        SPBRGL = 25;          //9600 baud at 4MHz clock
        BRGH =1;
        BRG16 = 0;
        TXEN = 0;             //disable transmitter
        CREN=1;               //enable receiver
        SYNC = 0;
        SPEN = 1;

                                //alternate pins set above
        TRISB1 = 1;           //set as input for RX
//      TRISA4 = 0;           //set as output for TX

//manual control interrupt setup
        IOCBP = 0;            //disable positive change interrupt
        IOCBN = 0b11011100;   //enable neg change interrupts
        IOCBF = 0;            //clear all change interrupt flags
        IOCIE = 1;            //enable change interrupts

        RCIE=1;               //enable receiver interrupts
        PEIE=1;
        GIE=1;                //enable interrupts

//recover Default values on startup. If cal button pressed when
//power is applied, the original defaults for the N/S- Up/Down limits
//are recovered and stored back into EEPROM
/* Defaults
AZ 0 deg in 0,1 = 0x0031
AZ 359 deg in 2,3 = 0x03dc
EL 0 deg in 4,5 = 0x0114
EL 90 deg in 6,7 = 0x02c7
*/

        if (!RB7)
        {
            //AZ Low
            eeprom_write(0, 0x00);
            eeprom_write(1, 0x31);
            //AZ High
            eeprom_write(2, 0x03);
            eeprom_write(3, 0xdc);
            //EL Low
            eeprom_write(4, 0x01);
            eeprom_write(5, 0x14);
            //El High

```

```

    eeprom_write(6, 0x02);
    eeprom_write(7, 0xc7);
}
while (!RB7)                                //flash data light until cal button released to continue program
{
    RA4=1;
    __delay_ms(100);
    RA4=0;
    __delay_ms(100);
}
RA4=0;                                        //make sure data light is off
get_cal();                                  //get turn on positions
old_AZ_ADC = get_ADC(AZ);
old_EL_ADC = get_ADC(EL);
}

```

```

void get_cal(void)
{
    AZ_ADC_0 = eeprom_read(0) << 8;
    AZ_ADC_0 = AZ_ADC_0 + eeprom_read(1);
    AZ_ADC_359=eeprom_read(2)<<8;
    AZ_ADC_359=AZ_ADC_359 + eeprom_read(3);
    AZ_range=AZ_ADC_359 - AZ_ADC_0;
    EL_ADC_0 = eeprom_read(4) << 8;
    EL_ADC_0 = EL_ADC_0 + eeprom_read(5);
    EL_ADC_90 = eeprom_read(6) << 8;
    EL_ADC_90 = EL_ADC_90 + eeprom_read(7);
    EL_range = EL_ADC_90 - EL_ADC_0;
}

```

```

void cal(void)
{
    if (RA5)                                //if RA5 high then rotor set up for North Stop and calibration
                                            //is allowed. If RA5 is low, calibration not allowed
    {
        if (get_ADC(AZ) <= 512)            //store low end AZ
        {
            eeprom_write(0, ADRESH);
            eeprom_write(1, ADRESL);
        }
        else                                //store high end AZ
        {
            eeprom_write(2, ADRESH);
            eeprom_write(3, ADRESL);
        }
        if (get_ADC(EL) <= 512)            //store low end EL
        {
            eeprom_write(4, ADRESH);
            eeprom_write(5, ADRESL);
        }
    }
}

```

```

    }
    else                                     //store high end EL
    {
        eeprom_write(6, ADRESH);
        eeprom_write(7, ADRESL);
    }
    get_cal();
    IOCFE = 1;                             //turn on change interrupts
}

void hundred_ms_delay(int t)               //100 ms intervals
{
    for(i=0;i<t;i++)
    {
        __delay_ms(100);
    }
}

int get_ADC(char channel)
{
    int ADC;
    ADCON0bits.CHS = channel;              //set ADC channel
    __delay(100);                          //delay to allow channel to settle
    GO_NDONE = 1;                          //start conversion
    while (GO_NDONE) continue;             //wait until it is done
    ADC = ADRESH << 8;                     //load high byte into variable
    ADC = ADC + ADRESL;                    //add low byte into variable
    return ADC;                           //return the 16 bit value
}

void RX_cmd(void)                         //"A" received to get here
{
    //EASYCOM commands antenna pointing in the following format: AZ###.# EL###.#. The characters in the
    //string are in ASCII format, therefore they must be converted into decimal format for use in calculations
    //and for comparisons. This routine tests for specific characters, and converts ASCII representations
    //of decimal numbers into the numbers (by subtracting 48 from the ASCII values)
    RA4=1;                                //turn on data LED
    while (!RCIF) continue;               //wait for next character
    if (RCREG != 'Z') goto ex_RX;         //if next not "Z" then abort
    while (!RCIF) continue;               //wait for next character
    AZ_deg = (RCREG - 48)*100;             //get hundreds, convert from ascii to dec, put in place
    while (!RCIF) continue;               //wait for next character
    AZ_deg = AZ_deg + (RCREG - 48)*10;     //get tens, convert from ascii to dec, put in place
    while (!RCIF) continue;               //wait for next character
    AZ_deg = AZ_deg + (RCREG - 48);        //get ones, convert from ascii to dec, put in place
    //the next 3 characters ".0 space" and captured and discarded
    while (!RCIF) continue;               //wait for next character
    EL_deg = RCREG;

```

```

while (!RCIF) continue;          //wait for next character
EL_deg = RCREG;
while (!RCIF) continue;          //wait for next character
EL_deg = RCREG;
                                  //next looking for "EL"
while (!RCIF) continue;          //wait for next character
if (RCREG != 'E') goto ex_RX;     //if next not "E" then abort
while (!RCIF) continue;          //wait for next character
if (RCREG != 'L') goto ex_RX;     //if next not "L" then abort
while (!RCIF) continue;          //wait for next character
EL_deg = (RCREG - 48)*100;         //get hundreds, convert from ascii to dec, put in place
while (!RCIF) continue;          //wait for next character
EL_deg = EL_deg + (RCREG - 48)*10; //get tens, convert from ascii to dec, put in place
while (!RCIF) continue;          //wait for next character
EL_deg = EL_deg + (RCREG - 48);    //get ones, convert from ascii to dec, put in place

```

```

if (!RA5)                        //if RA5 is grounded, then rotor set up for South Stop
{
  if (AZ_deg >=180)
  {
    AZ_deg=AZ_deg-180;
  }
  else
  {
    AZ_deg=AZ_deg+180;
  }
}

```

```

if ((EL_deg < 0) || (EL_deg > 90) || (AZ_deg < 0) || (AZ_deg > 359)) goto ex_RX;
                                  //check if data is in range
x = AZ_deg;                      //convert int to float for the division math
y = x/359;                      //if you don't do this, the fraction result returns zero
new_AZ_ADC = ((y)*AZ_range) + AZ_ADC_0; //get percent of range
x= EL_deg;
y = x/90;
new_EL_ADC = ((y)*EL_range) + EL_ADC_0;

```

```

if (new_AZ_ADC > old_AZ_ADC)      //if above the old value, increase direction
{
  AZ_inc(1023, new_AZ_ADC);
  goto el;
}
if (new_AZ_ADC < old_AZ_ADC)      //if below the old value, decrease direction
{
  AZ_dec(1023, new_AZ_ADC);
}
//if no change in AZ go right to EL

```

el:

```

        if (new_EL_ADC > old_EL_ADC)
        {
            EL_up(1023, new_EL_ADC);
            goto exit_move;
        }
        if (new_EL_ADC < old_EL_ADC)
        {
            EL_down(1023, new_EL_ADC);
        }
exit_move:
    old_AZ_ADC = new_AZ_ADC;
    old_EL_ADC = new_EL_ADC;
//after the new position, the new becomes the old

ex_RX:
manual = 0;
CREN = 0;
RCIE = 1;
CREN = 1;
RA4 = 0;
IOCIIE = 1;
}
//EL routines*****

void EL_up_manual (int z)
{
    RA3=0;
    STR2B=0;
    RA6=0;
    STR2A=1;
    CCP2CONbits.DC2B = z & 0b00000011;
    CCPR2L = z >>2;
    RA3=1;
    while (!RB4)
    {
        if (get_ADC(EL) >= EL_max_ADC)
        {
            EL_stop();
        }
    }
    EL_stop();
}

void EL_down_manual (int z)
{
    RA3=0;
    STR2A=0;
    RA7=0;
    STR2B=1;
    CCP2CONbits.DC2B = z & 0b00000011;
    //stop EL motor
    //make RA6 I/O
    //put PWM on RA7
    //get lsb or speed an put in DC1B
    //get msbs and put in CCPR1L
    //use with manual control RA4 to ground

```

```

    CCPR2L = z >>2;                                //get msbs and put in CCPR1L
    RA3=1;                                           //use with manual control RB6 ground
    while (!RB6)
    {
        if (get_ADC(EL) <= EL_min_ADC)
        {
            EL_stop();
        }
    }
    EL_stop();
}

void EL_stop(void)
{
    RA3=0;                                           //turn off EL motor
    STR2A=0;                                         //make RA7 I/O
    RA7=0;                                           //forces pin low
    STR2B=0;                                         //make RA6 I/O
    RA6=0;                                           //forces pin low
    CCP2CONbits.DC2B = 0;
    CCPR2L = 0;
    old_AZ_ADC = get_ADC(AZ);
    old_EL_ADC = get_ADC(EL);
    IOCE = 1;
}

void EL_up (int z, int EL_cmd)
{
    RA3=0;                                           //stop EL motor
    STR2B=0;                                         //make RA6 I/O
    RA6=0;                                           //forces this pin to go low
    STR2A=1;                                         //put PWM on RA7
    CCP2CONbits.DC2B = z & 0b00000011; //get lsb or speed an put in DC1B
    CCPR2L = z >>2;                                //get msbs and put in CCPR1L
    RA3=1;
    while ((get_ADC(EL) <= EL_max_ADC)&& (get_ADC(EL) < EL_cmd)) continue;
    EL_stop();
//    IOCE = 1;                                     //after exit from manual control, reset change interrupt
}

void EL_down (int z, int EL_cmd)
{
    RA3=0;                                           //stop EL motor
    STR2A=0;                                         //make RA7 I/O
    RA7=0;                                           //forces this pin to go low
    STR2B=1;                                         //put PWM on RA7
    CCP2CONbits.DC2B = z & 0b00000011; //get lsb or speed an put in DC1B
    CCPR2L = z >>2;                                //get msbs and put in CCPR1L
    RA3=1;
    while ((get_ADC(EL) >= EL_min_ADC)&&(get_ADC(EL) > EL_cmd)) continue;
}

```

```

        EL_stop();
//      IOIE = 1;                                     //after exit from manual control, reset change interrupt
}
//end EL routines*****

//AZ routines*****

void AZ_up_manual (int z)
{
    RA2=0;                                           //stop AZ motor
    STR1B=0;                                         //make RB5 I/O
    RB5=0;
    STR1A=1;                                         //make RB0 PWM
    CCP1CONbits.DC1B = z & 0b00000011;             //get lsb or speed an put in DC1B
    CCPR1L = z >>2;                                 //get msbs and put in CCPR1L
    RA2=1;                                           //start AZ motor
    while (!RB3)                                     //use with manual control
    {
        if (get_ADC(AZ) >= AZ_max_ADC)
        {
            AZ_stop();
        }
    }
//      IOIE = 1;                                     //after exit from manual control, reset change interrupt
//      old_AZ_ADC = get_ADC(AZ);
//      old_EL_ADC = get_ADC(EL);
    AZ_stop();
}

void AZ_inc(int z, int AZ_cmd)
{
    RA2=0;                                           //stop AZ motor
    STR1B=0;                                         //make RB5 I/O
    RB5=0;
    STR1A=1;                                         //make RB0 PWM
    CCP1CONbits.DC1B = z & 0b00000011; //get lsb or speed an put in DC1B
    CCPR1L = z >>2;                                 //get msbs and put in CCPR1L
    RA2=1;                                           //start AZ motor
    while ((get_ADC(AZ) <= AZ_max_ADC)&& (get_ADC(AZ) < AZ_cmd)) continue;
    AZ_stop();
//      IOIE = 1;                                     //after exit from manual control, reset change interrupt
}

void AZ_dec(int z, int AZ_cmd)
{
    RA2=0;                                           //stop AZ motor
    STR1A=0;                                         //making RB0 I/O
    RB0=0;
    STR1B=1;                                         //make RB5 PWM
    CCP1CONbits.DC1B = z & 0b00000011;             //get lsb or speed an put in DC1B

```

```

        CCPR1L = z >>2;                //get msbs and put in CCPR1L
        RA2=1;                          //start AZ motor
        while ((get_ADC(AZ) >= AZ_min_ADC)&&(get_ADC(AZ) > AZ_cmd)) continue;
        AZ_stop();
//      IOIE = 1;                        //after exit from manual control, reset change interrupt
    }

void AZ_down_manual (int z)
{
    RA2=0;                              //stop AZ motor
    STR1A=0;                            //making RB0 I/O
    RB0=0;
    STR1B=1;                            //make RB5 PWM
    CCP1CONbits.DC1B = z & 0b00000011; //get lsb or speed an put in DC1B
    CCPR1L = z >>2;                    //get msbs and put in CCPR1L
    RA2=1;                              //start AZ motor
    while (!RB2)                        //use with manual control
    {
        if (get_ADC(AZ) <= AZ_min_ADC)
        {
            AZ_stop();
        }
    }
    AZ_stop();
//      IOIE = 1;                        //after exit from manual control, reset change interrupt
//      old_AZ_ADC = get_ADC(AZ);
//      old_EL_ADC = get_ADC(EL);
}

void AZ_stop(void)
{
    RA2=0;                              //turn off AZ motor
    STR1A=0;                            //making RB0 I/O
    RB0=0;
    STR1B=0;                            //make RB5 I/O
    RB5=0;
    CCP1CONbits.DC1B = 0;
    CCPR1L = 0;
    old_AZ_ADC = get_ADC(AZ);
    old_EL_ADC = get_ADC(EL);
    IOIE = 1;
}
//end AZ routines*****

void interrupt ISR(void)
{
    manual=0;                           //clear flags
    if (IOCIF)                           //if interrupt from change of manual switches

```

```

{
    IOIE = 0;
    manual = IOCBF;

    IOCBF = 0;
}
if (RCIF)
{
    if (RCREG == 'A')
    {
        manual = 1;
        RCIE = 0;
    }
}

//disable further change interrupts
//store flags in manual variable
//up to 0b11011100 from change interrupts
//clear all flags

//if interrupt came from received character

//valid first character is "A"

//use 1 as flag for received character
//disable receiver interrupts and dedicate to receive effort
}

void main (void)
{
    init();

    while (1)
    {
        switch (manual)
        {
            case 0:
                //no manual exit switch
                {
                    break;
                }
            case 0b00000001:
                //received character
                {
                    RX_cmd();
                    break;
                }
            case 0b00001000:
                //RB3 grounded
                {
                    AZ_up_manual(1023);
                    AZ_stop();
                    break;
                }
            case 0b00000100:
                //RB2 grounded
                {
                    AZ_down_manual(1023);
                    AZ_stop();
                    break;
                }
            case 0b00010000:
                //RB4 grounded
                {
                    EL_up_manual(1023);
                    EL_stop();
                }
        }
    }
}

```

```
break;
}
case 0b01000000:           //RB6 grounded
{
  EL_down_manual(1023);
  EL_stop();
  break;
}
case 0b10000000:           //RB7 grounded
{
  cal();
  break;
}
}
}
}
```