Preliminary Code Listings

D4 Thames

March 3, 2017

1 Hosted Telemetry and Tuning

1.1 Ncurses telemetry

```
#include "thameshost.h"

void initialise_screen(void)
{
        initscr();
        noecho();
        cbreak();
        nodelay(stdscr, TRUE);
}

PID* write

void write_base_tun(void)
{
        clear();
        attron(A_BOLD);
        mvprintw( 0, TEXT_OFFSET, "Tuning System.");
        mvprintw( 1, TEXT_OFFSET, "Flight not available while tuning.");
        attroff(A_BOLD);
        mvprintw( 3, TEXT_OFFSET, "p - Edit Pitch System");
        mvprintw( 4, TEXT_OFFSET, "r - Edit Roll System");
        mvprintw( 5, TEXT_OFFSET, "y - Edit Yaw System");
}

void write_base_tel(void)
{
        clear();
        attron(A_BOLD);
        mvprintw( 0, TEXT_OFFSET, "Telemetry System");
        attroff(A_BOLD);
        attron(A_UNDERLINE);
        mvprintw( 2, TEXT_OFFSET, "Remote Control Inputs");
        attroff(A_UNDERLINE);
        mvprintw( 3, TEXT_OFFSET, "Throttle --------");
        mvprintw( 4, TEXT_OFFSET, "Yaw      --------");
        mvprintw( 5, TEXT_OFFSET, "Pitch    --------");
        mvprintw( 6, TEXT_OFFSET, "Roll     --------");
        attron(A_UNDERLINE);
        mvprintw( 8, TEXT_OFFSET, "Linear Acceleration");
        attroff(A_UNDERLINE);
        mvprintw( 9, TEXT_OFFSET, "lX --------");
        mvprintw(10, TEXT_OFFSET, "lY --------");
        mvprintw(11, TEXT_OFFSET, "lZ --------");
        attron(A_UNDERLINE);
        mvprintw(13, TEXT_OFFSET, "Angular Acceleration");
        attroff(A_UNDERLINE);
        mvprintw(14, TEXT_OFFSET, "aX --------");
        mvprintw(15, TEXT_OFFSET, "aY --------");
        mvprintw(16, TEXT_OFFSET, "aZ --------");
}

int main(void)
{
        // Open File, Initialise System
```

```
FILE* serial_port = fopen(PORT_LOCATION, "r");
mode = telemetry;

// Initialise Screen
initialise_screen();
write_base_tel();

while (1) {
        // Read from file
        input_byte = fgetc(serial_port);

        // Deal with input
        switch (mode) {
                case telemetry:
                        // Get control code and data from input
                        if (!(input_byte & 0x80)) {
                                control_code = (input_byte & 0xE0) >> 5;
                                input_data = (input_byte & 0x1F);
                        } else {
                                control_code = (input_byte & 0xF0) >> 4;
                                input_data = (input_byte & 0x0F);
                        }
                        switch (control_code) {
                                // Remote Control Input
                                case S_THROTTLE:
                                        mvprintw( 3, TEXT_OFFSET, "Throttle %f",
                                             input_data/31);
                                        clrtoeol();
                                        break;
                                case S_YAW:
                                        mvprintw( 4, TEXT_OFFSET, "Yaw      %f",
                                             input_data/31);
                                        clrtoeol();
                                        break;
                                case S_PITCH:
                                        mvprintw( 5, TEXT_OFFSET, "Pitch    %f",
                                             input_data/31);
                                        clrtoeol();
                                        break;
                                case S_ROLL:
                                        mvprintw( 6, TEXT_OFFSET, "Roll     %f",
                                             input_data/31);
                                        clrtoeol();
                                        break;
                                // Linear Acceleration
                                case S_LINX:
                                        mvprintw( 9, TEXT_OFFSET, "lX %f",
                                            input_data/15);
                                        clrtoeol();
                                        break;
                                case S_LINY:
                                        mvprintw(10, TEXT_OFFSET, "lY %f",
                                            input_data/15);
                                        clrtoeol();
                                        break;
                                case S_LINZ:
                                        mvprintw(11, TEXT_OFFSET, "lZ %f",
                                            input_data/15);
                                        clrtoeol();
                                        break;
                                // Angular Acceleration
                                case S_ANGX:
                                        mvprintw(14, TEXT_OFFSET, "aX %f",
                                            input_data/15);
                                        clrtoeol();
                                        break;
                                case S_ANGY:
                                        mvprintw(15, TEXT_OFFSET, "aY %f",
                                            input_data/15);
                                        clrtoeol();
                                        break;
                                case S_ANGZ:
                                        mvprintw(16, TEXT_OFFSET, "aZ %f",
```

```
                                                input_data/15);
                                        clrtoeol();
                                        break;
                                // Start/Stop Signals
                                case S_STARTF:
                                        // Unexpected Start Signal
                                        mvprintw(18, TEXT_OFFSET, "E Unexpected
                                            start signal received (%d).",
                                            input_byte);
                                        clrtoeol();
                                        break;
                                case S_STOPF:
                                        // Flight has stopped
                                        // Change to tuning mode
                                        mvprintw(18, TEXT_OFFSET, "M Flight
                                            Stopped, changing to tuning mode.");
                                        mode = tuning;
                                        clrtoeol();
                                        write_base_tun();
                                        break;
                                default:
                                        mvprintw(18, TEXT_OFFSET, "E Unexpected
                                            symbol received (%d).", input_byte);
                                        clrtoeol();
                                        break;
                                }
                                break;
                        case tuning:
                                ui_input = wgetch(stdscr);
                                switch (ui_input) {
                                        case 'q':
                                                // Quit Tuning Mode
                                                // TODO Send End Symbol
                                                write_base_tel();
                                                mode = telemetry;
                                                break;
                                        case 'p':
                                                // Set p coefficient
                                                break;
                                        case 'r':
                                                // Set i coefficient
                                                break;
                                        case 'y':
                                                // Set i error threshold.
                                                break;
                                        default:
                                                break;
                                }
                                break;
                        }
                        refresh();
                        usleep(1000000);
                }
                return 0; // Unreachable
}
```

1.2 Python Tuning

```
print("D4 Thames Tuning System")

class PID:
        p = 0
        i = 0
        d = 0
        I = 0

pitch = PID()
roll = PID()
yaw = PID()

data_in_buffer = False

while True:
```

```
user_input = input("> ").split(" ")
if user_input[0].lower() == "pitch":
        user_input = user_input[1:]
        for i in range(len(user_input)):
                try:
                        if user_input[i][0] == 'p':
                                pitch.p = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set pitch's proportional coefficient to %
                                    f" % pitch.p)
                        if user_input[i][0] == 'i':
                                pitch.i = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set pitch's integral coefficient to %f" %
                                     pitch.i)
                        if user_input[i][0] == 'I':
                                pitch.I = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set pitch's integral threshold to %f" %
                                    pitch.I)
                        if user_input[i][0] == 'd':
                                pitch.d = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set pitch's differential coefficient to %
                                    f" % pitch.d)
                except:
                        print("Invalid Input!")
elif user_input[0].lower() == "roll":
        user_input = user_input[1:]
        for i in range(len(user_input)):
                try:
                        if user_input[i][0] == 'p':
                                roll.p = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set roll's proportional coefficient to %f
                                    " % roll.p)
                        if user_input[i][0] == 'i':
                                roll.i = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set roll's integral coefficient to %f" %
                                    roll.i)
                        if user_input[i][0] == 'I':
                                roll.I = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set roll's integral threshold to %f" %
                                    roll.I)
                        if user_input[i][0] == 'd':
                                roll.d = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set roll's differential coefficient to %f
                                    " % roll.d)
                except:
                        print("Invalid Input!")
elif user_input[0].lower() == "yaw":
        user_input = user_input[1:]
        for i in range(len(user_input)):
                try:
                        if user_input[i][0] == 'p':
                                yaw.p = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set yaw's proportional coefficient to %f"
                                     % yaw.p)
                        if user_input[i][0] == 'i':
                                yaw.i = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set yaw's integral coefficient to %f" %
                                    yaw.i)
                        if user_input[i][0] == 'I':
                                yaw.I = float(user_input[i][1:])
                                data_in_buffer = True
                                print("Set yaw's integral threshold to %f" % yaw
                                    .I)
                        if user_input[i][0] == 'd':
```

```
                                        yaw.d = float(user_input[i][1:])
                                        data_in_buffer = True
                                        print("Set yaw's differential coefficient to %f"
                                             % yaw.d)
                        except:
                                print("Invalid Input!")
        elif user_input[0].lower() == "send" or user_input[0].lower() == "update":
                print("Sending Data...")
                #TODO Send the Data.
                print("Done")
                data_in_buffer = False
        elif user_input[0].lower() == "status":
                if (data_in_buffer):
                        print("Data yet to be sent.")
                else:
                        print("Up-to-date with drone.")
                print("pitch p%f i%f d%f I%f" % (pitch.p, pitch.i, pitch.d, pitch.I))
                print("roll  p%f i%f d%f I%f" % (roll.p, roll.i, roll.d, roll.I))
                print("yaw   p%f i%f d%f I%f" % (yaw.p, yaw.i, yaw.d, yaw.I))
```

2 Control

2.1 Buffer

```
/*

Harry Beadle
D4 Thames
Buffer (buffer.c)

Rotational FIFO buffers for sensor, RC and tuning data.

*/

//////////////////
// 8-Bit Buffer //
//////////////////

uint8_t buffer8_pop(buffer8* b)
{
        // Store the value at the outdex ready
        // for return and increment outdex.
        uint8_t rv = b->buffer[b->outdex++]
        // If we're at the end of the buffer,
        // go back to the start.
        if (b->outdex == BUFFER_SIZE)
                b->outdex = 0;
        // Return the stored value.
        return rv;
}

void buffer8_add(buffer8* b, uint8_t c)
{
        // Set the value at the index location
        // of the buffer to the input value and
        // increment the index.
        b->buffer[b->index++] = c;
        // If we're at the end of the buffer,
        // go back to the start.
        if (b->index == BUFFER_SIZE)
                b->index = 0;
}

int buffer8_rdy(buffer8* b)
{
        // If the buffer is ready the index
        // will not equal the outdex.
        return b->index != b->outdex;
}

///////////////////
// 16-Bit Buffer //
```

```
//////////////////

uint16_t buffer16_pop(buffer16* b)
{
        // Store the value at the outdex ready
        // for return and increment outdex.
        uint16_t rv = b->buffer[b->outdex++];
        // If we're at the end of the buffer,
        // go back to the start.
        if (b->outdex == BUFFER_SIZE)
                b->outdex = 0;
        // Return the stored value.
        return rv;
}
void buffer16_add(buffer16* b, uint16_t c)
{
        // Set the value at the index location
        // of the buffer to the input value and
        // increment the index.
        b->buffer[b->index++] = c;
        // If we're at the end of the buffer,
        // go back to the start.
        if (b->index == BUFFER_SIZE)
                b->index = 0;
}

int buffer16_rdy(buffer16* b)
{
        // If the buffer is ready the index
        // will not equal the outdex.
        return b->index != b->outdex;
}
```

2.2 Control

```
/*

Harry Beadle
D4 Thames
Control (control.c)

PID Control

*/

#include "inc/control.h"

TIR0 = (uint8_t) tick_control(yaw_buffer_pop(), yaw);


double tick_control(double input, system* s)
{
        // Calculate Errors on this tick.
        double current_error;
        current_error = s->setpoint - input;
        s->e_d = (current_error - s->e_p)/s->time_period;
        s->e_i += s->k_i * s->e_p * s->time_period;
        s->e_p =  current_error;

        // Handle Maximum Integral Error
        if (s->e_i > s->i_max || s->e_i < -s->i_max)
                s->e_i = 0;

        // Calculate Output
        double output;
        output += s->k_p * s->e_p;
        output += s->e_i; // k_i already applied.l
        output += s->k_d * s->e_d;

        // Handle Maximum Output
        if (output > s->o_max)
                output = s->o_max;
        if (output < s->o_min)
```

```
                output = s->o_min;

        return output;
}
```

2.3 Drone

```
/*.

Harry Beadle
D4 Thames
Drone (drone.c)

Combines the function of all other modules into a complete system.

*/

#include "drone.h"

int main(void)
{
        // Initialise Buffers
        // ***TODO*** This should be in the communications file.
        buffer8 control_buffer;
        buffer8 thrust_buffer;
        buffer16 pitch_buffer;
        buffer16 roll_buffer;
        buffer16 yaw_buffer;

        // Initialise Mode
        mode = flight;

        while (1) {
                switch (mode) {
                        case flight:
                                if (buffer16_rdy(pitch_buffer)) {
                                        pitch = buffer16_pop(pitch_buffer);
                                        pitch_adjust = tick_control(pitch, pitch_system)
                                            ;
                                }
                                if (buffer16_rdy(roll_buffer)) {
                                        roll = buffer16_pop(roll_buffer);
                                        roll_adjust = tick_control(roll, roll_system);
                                }
                                if (buffer16_rdy(yaw_buffer)) {
                                        yaw = buffer16_pop(yaw_buffer);
                                        yaw_adjust = tick_control(yaw, yaw_system);
                                }
                                if (buffer8_rdy(thrust_buffer)) {
                                        // Get thrust
                                        thrust = buffer8_pop(thrust_buffer);
                                        // If thrust is zero then assume we've langed
                                            and
                                        // go into tuning mode.
                                        if (thrust == 0) {
                                                mode = tuning;
                                        }
                                }
                                /// Update the system
                                // Change the comparator registers to change PWM
                                // duty cycle.
                                // ***TODO***
                                break;
                        case tuning:
                                if (control_buffer_ready()) {
                                        control_data = control_buffer_pop();
                                        switch (control_data) {
                                                default:
                                                        break;
                                        }
                                }
                                break;
                }
```

```
        }
}
```

3 Serial Telemetry Embedded

```
/*

Harry Beadle
D4 Thames
Telemetry (telemetry.c)

Collects data from IMU and RC and outputs over a Bluetooth serial
connection to the hosted application.

** Inputs
UART1 Serial from PPM Decoder (RC Data)
        TODO
SPI   SPI from the IMU
        MOSI Data from IMU
        SCK  Clock from IMU
        ~SS  Pull Low

** Outputs
UART0 Serial to Bluetooth Module
        TODO

*/

buffer8 b;

ISR(USART0_Rx_vect)
{
        // RC Data Rx, put it in the buffer.
        buffer8_add(b, UDR0);
}

ISR(SPI_STC_vect)
{
        // SPI Data Rx, put it in the buffer.
        buffer8_add(b, SPDR);
}

void init_SPI(void)
{
        // Set MISO as an Input
        DDR_SPI = _BV(DD_MISO);
        // Enable SPI
        SPCR =_BV(SPE);
}

void init_UART0(unsigned int baud)
{
        // Set Baud Rate
        UBRRH0 = (unsigned char) baud >> 8;
        UBBRL0 = (unsigned char) baud;
        // Enable Tx and Rx
        UCSR0B = _BV(RXEN0) | _BV(TXEN0)
        // Set frame: 8 Data, 2 Stop
        UCSR0C = _BV(USBS0) | (3 << UCSZ00])
}

void init_UART1(unsigned int baud)
{
        // Set Baud Rate
        UBRRH1 = (unsigned char) baud >> 8;
        UBBRL1 = (unsigned char) baud;
        // Enable Tx and Rx
        UCSR1B = _BV(RXEN1) | _BV(TXEN1)
        // Set frame: 8 Data, 2 Stop
        UCSR1C = _BV(USBS1) | (3 << UCSZ10)
}
```

```
int main(void)
{
        /* NOTE we may need to start UART1 in 9600 to initialise the
        /* Bluetooth module. For testing with a hosted platform directly
        /* however, we don't need this. */

        // Initialise Communications
        init_SPI();
        init_UART1(14400);
        init_UART0(9600);
        // Loop Forever
        while (1) {
                // If the buffer is empty and the UART is not busy then
                // output the next item from the buffer though UART0.
                if (buffer8_rdy(b) && UARTREADY)
                {
                        UDR0 = buffer8_pop(b);
                }
                // Since
        }
        return 0; // Unreachable
}
```

4 PPM Decoder

```
/** original version: PPMDECODER.c

*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/twi.h>

#ifndef F_CPU
#define F_CPU 12000000UL
#endif

#define MIN_SYNC_LEN 4000L
#define MIN_PULSE_LEN 800L
#define MAX_PULSE_LEN 2200L
#define NEUTRAL_PULSE_LEN 1500L
#define NEG_PULSE_LEN 200L
#define NORM_FRAME_LEN 22500L
#define SYNC_SPACER NORM_FRAME_LEN/5
#define MIN_FRAME_LEN NORM_FRAME_LEN-SYNC_SPACER
#define MAX_FRAME_LEN NORM_FRAME_LEN+SYNC_SPACER

#define NUM_PULSE_PER_FRAME 5
#define MIN_NUM_VALID_FRAMES 2 // number of valid frames required to switch back from
    NORC
#define MAX_NUM_INVALID_FRAMES 4 // number of invalid frames required to switch to NORC

#define STATE_NORC 0 // no valid ppm, use neutral value
#define STATE_MANUAL 1 // valid ppm

#define MAX_TWI_MSG_LEN 21
#define

#define TWI_ACK _BV(TWINT)|_BV(TWEA)|_BV(TWIE)|_BV(TWEN) // TWINT=Interrupt flag,TWEA=
    Enable acknowledge,TWIE=Interrupt enable,TWEN=TWI enable

uint16_t inPW[NUM_PULSE_PER_FRAME]; // servo values from R/C receiver
uint16_t digitalData[NUM_PULSE_PER_FRAME]; // 5 bits control data
uint16_t neutralPW[NUM_PULSE_PER_FRAME] = NEUTRAL_PULSE_LEN; // neutral servo values
uint16_t stat[NUM_PULSE_PER_FRAME];

volatile uint8_t CurrentState = STATE_NORC;

volatile uint8_t i2cIn[MAX_TWI_MSG_LEN];
valatile uint8_t i2cFinished = 0;

uint16_t DecimaltoBinary(const uint8_t Decnumber)
```

```
{
        uint16_t Result = 0;
        uint8_t i = 1;
        uint8_t tmp = Decnumber;
        while (tmp > 0) {
                Result = (tmp % 2)*i;
                i = i*10;
                tmp = (tmp/2) - (tmp%2);
        }
        return Result;
}
        static inline uint8_t ValidPulseLen (const uint16_t pulseLen)
{
        return (pulseLen > MIN_PULSE_LEN && pulseLen < MAX_PULSE_LEN);
}

int main(void)
{
        uint8_t i;
        uint16_t lastIcrTime = 0;
        uint8_t inPulseCount = 0;
        uint16_t inSyncTime = 0;
        uint16_t inFrameLen = 0;

        uint8_t validSync = 0;

        uint8_t invalidFrameCount = 0;
        uint8_t validFrameCount = 0;

        // Power and noise reduction
        PRR = _BV(PRTIM2)|_BV(PRTIM0)|_BV(PRSPI)|_BV(PRUSART0)|_BV(PRADC);

        i2cIn[0] = 0x80;

        DDRB = 0b00000010;

        ICR1 = 0;

        TCCR1B = (0 << ICES1)|_BV(CS11); //prescaler 8, input capture according to PPM
            negative edge

        TWCR = _BV(TWEN)|_BV(TWIE)|_BV(TWINT)|_BV(TWEA)|_BV(TWSTA)|_BV(TWSTO)|_BV(TWWC);

        sei();

        while (1) {
                if (TIFR1 & _BV(ICF1)) {
                        uint16_t currIcrTime = ICR1;
                        TIFR1 = _BV(ICF1);
                        uint16_t PulseLen = currIcrTime - lastIcrTime;
                        lastIcrTime = currIcrTime;
                        if ((PulseLen > MIN_SYNC_LEN) && (PulseLen < NORM_FRAME_LEN)) {
                                //sync detected
                                inFrameLen = currIcrTime - inSyncTime;
                                inSyncTime = currIcrTime;
                                if (validSync && (inFrameLen > MIN_FRAME_LEN) && (
                                    inFrameLen < MAX_FRAME_LEN) && (inPulseCount ==
                                    NUM_PULSE_PER_FRAME)) {
                                        invalidFrameCount = 0;
                                        if (validFrameCount < MIN_NUM_VALID_FRAMES)
                                            validFrameCount++;
                                } else {
                                        validFrameCount = 0;
                                        if (invalidFrameCount < MAX_NUM_INVALID_FRAMES)
                                            invalidFrameCount++;
                                }
                                validSync = 1;
                                inPulseCount = 0;
                        } else if (validSync && ValidPulseLen(PulseLen) && (inPulseCount
                             < NUM_PULSE_PER_FRAME)) {
                                inPW[inPulseCount++] = PulseLen;
                        } else {
                                validSync = 0;
```

```
                        }
                }

                for (i=0; i<=NUM_PULSE_PER_FRAME;i++) {
                        digitalData[i] = DecimalToBinary(inPW[i]-800)/44
                } // convert into 5 bits data
        }
}
```