```
21          UBRR0H = (F_CPU/(57600*16L)-1) >> 8;
22          UBRR1H = (F_CPU/(9600*16L)-1) >> 8;
23          UBRR0L = (F_CPU/(57600*16L)-1);
24          UBRR1L = (F_CPU/(9600*16L)-1);
25          // Enable Tx, Rx, Tx Interupt, Rx Interupt.
26          UCSR0B = _BV(RXEN0) | _BV(RXCIE0) | _BV(TXEN0) | _BV(TXCIE0);
27          UCSR1B = _BV(RXEN1) | _BV(RXCIE1) | _BV(TXEN1) | _BV(TXCIE1);
28          // 1 Start, 8 Data and 1 Stop bit.
29          UCSR0C = _BV(UCSZ00) | _BV(UCSZ01);
30          UCSR1C = _BV(UCSZ10) | _BV(UCSZ11);
31
32          state = control;
33          nstate = control;
34  }
35
36  // MPU Decoder
37  ISR(USART0_RX_vect, ISR_BLOCK)
38  {
39          // UART Entry from MPU Decoder
40          state = nstate;
41          temp_byte = UDR0;
42          switch (state) {
43                  case control:
44                          PORTB |=  _BV(PB0);
45                          // If valid control bit, store it and go onto the next
46                      // byte, if not keep waiting for a valid one, we *should*
47                          // eventually sync up.
48                          // bad_control = 1;
49                          if ((temp_byte & 0xF0) == 0xF0) {
50                          //      // Top nibble is 0xF
51                          //      if ((temp_byte & 0x0F) != 0x00) {
52                          //              // Data nibble is not 0
53                          //              if ((temp_byte & 0x0F) <= 0x03) {
54                          //                      // Data nibble is not above 3
55                                                          // bad_control = 0;
56                                                          control_byte = temp_byte;
57                                                          nstate = high;
58                                          // }
59                                  // }
60                          }
61                          // if (bad_control) {
62                          //      // Invalid Control Byte
63                          //      PORTB |= _BV(PB1);
64                          //      nstate = control;
65                          // }
66                          PORTB &= ~_BV(PB0);
67                          break;
68                  case high:
69                          // Store the high byte, wait for the low byte.
70                          high_byte = temp_byte;
71                          nstate = low;
72                          break;
73                  case low:
74                          // Store the low byte, switch on the control byte, put the
75                          // Rx'd data wherever it belongs, and tell the control
76                          // System it's been updated.
77                          low_byte = temp_byte;
78                          switch (control_byte) {
79                                  case PITCH_SYMBOL:
80                                          pitch = high_byte << 8 | low_byte;
81                                          pitch_update = 1;
82                                          break;
```



```
83                                  case ROLL_SYMBOL:
84                                          roll = high_byte << 8 | low_byte;
85                                          roll_update = 1;
86                                          break;
87                                  case YAW_SYMBOL:
88                                          yaw = high_byte << 8 | low_byte;
89                                          yaw_update = 1;
90                                          break;
91                                  default:
92                                          break;
93                          }
94                          nstate = control;
95                          break;
96          }
97  }
98
99  // Remote Controller Communications
100 ISR(USART1_RX_vect)
101 {
102         temp_byte_rc = UDR1;
103         control_code = 0xC0 & temp_byte_rc;
104         data_byte = 0x3F & temp_byte_rc;
105         switch (control_code) {
106                 case S_THROTTLE:
107                         if (data_byte != 0) {
108                                 thrust <<= 11;
109                         }
110                         break;
111                 case S_PITCH:
112                         pitch_system.setpoint = data_byte;
113                         break;
114                 case S_ROLL:
115                         roll_system.setpoint = data_byte;
116                         break;
117                 case S_YAW:
118                         yaw_system.setpoint = data_byte;
119                         break;
120         }
121 }
```

1.2 Buffer

1.2.1 Header

```
1   /*
2
3   Harry Beadle
4   D4 Thames
5   Buffer (buffer.h)
6
7   Generic
8
9   */
10
11  #ifndef _BUFFER_H_
12          #define _BUFFER_H_
13
14                  #include <avr/io.h>
15                  #define BUFFER_SIZE 128
16
17                  // 8-Bit Buffer
18                  typedef struct {
19                          uint8_t buffer[128];
```

```
20                      uint8_t index;
21                      uint8_t outdex;
22              } buffer8;
23              uint8_t buffer8_pop(buffer8* b);
24              void buffer8_add(buffer8* b, uint8_t c);
25              int buffer8_rdy(buffer8* b);
26
27              // 16-Bit Buffer
28              typedef struct {
29                      uint16_t buffer[128];
30                      uint8_t index;
31                      uint8_t outdex;
32              } buffer16;
33              uint16_t buffer16_pop(buffer16* b);
34              void buffer16_add(buffer16* b, uint16_t);
35              int buffer16_rdy(buffer16* b);
36
37              #include "../buffer.c"
38
39  #endif
```

1.2.2 Source

```
1   /*
2
3   Harry Beadle
4   D4 Thames
5   Buffer (buffer.c)
6
7   Generic rotational buffers of both 8-bit and 16-bit size.
8
9   */
10
11  #include "inc/buffer.h"
12
13  //////////////////
14  // 8-Bit Buffer //
15  //////////////////
16
17  uint8_t buffer8_pop(buffer8* b)
18  {
19          // Store the value at the outdex ready
20          // for return and increment outdex.
21          uint8_t rv = b->buffer[b->outdex];
22          b->outdex++;
23          // If we're at the end of the buffer,
24          // go back to the start.
25          if (b->outdex == BUFFER_SIZE)
26                  b->outdex = 0;
27          // Return the stored value.
28          return rv;
29  }
30
31  void buffer8_add(buffer8* b, uint8_t c)
32  {
33          // Set the value at the index location
34          // of the buffer to the input value and
35          // increment the index.
36          b->buffer[b->index] = c;
37          b->index++;
38          // If we're at the end of the buffer,
39          // go back to the start.
```



```
40          if (b->index == BUFFER_SIZE)
41                  b->index = 0;
42  }
43
44  int buffer8_rdy(buffer8* b)
45  {
46          // If the buffer is ready the index
47          // will not equal the outdex.
48          return b->index != b->outdex;
49  }
50
51  //////////////////
52  // 16-Bit Buffer //
53  //////////////////
54
55  uint16_t buffer16_pop(buffer16* b)
56  {
57          // Store the value at the outdex ready
58          // for return and increment outdex.
59          uint16_t rv = b->buffer[b->outdex++];
60          // If we're at the end of the buffer,
61          // go back to the start->
62          if (b->outdex == BUFFER_SIZE)
63                  b->outdex = 0;
64          // Return the stored value.
65          return rv;
66  }
67  void buffer16_add(buffer16* b, uint16_t c)
68  {
69          // Set the value at the index location
70          // of the buffer to the input value and
71          // increment the index.
72          b->buffer[b->index++] = c;
73          // If we're at the end of the buffer,
74          // go back to the start->
75          if (b->index == BUFFER_SIZE)
76                  b->index = 0;
77  }
78
79  int buffer16_rdy(buffer16* b)
80  {
81          // If the buffer is ready the index
82          // will not equal the outdex.
83          return b->index != b->outdex;
84  }
```

1.3 Timer

1.3.1 Header

```
1   /*
2
3   Harry Beadle
4   D4 Thames
5   Timer (timer.h)
6
7   Headerfile for the timers outputting PWM to the motors.
8
9   */
10
11  #ifndef _TIMER_H_
12          #define _TIMER_H_
13
```

```
14              #include <avr/io.h>
15              #include <avr/interrupt.h>
16
17              uint8_t n0 = 0;
18              uint8_t n2 = 0;
19
20              typedef struct {
21                      uint8_t clear_cycle, cc_next;
22                      uint8_t clear_position, cp_next;
23                      uint8_t min_cc;
24                      uint8_t min_cp;
25                      uint8_t max_cc;
26                      uint8_t max_cp;
27              } rotor;
28
29              rotor Rotor1 = {
30                      .clear_cycle = 5,
31                      .clear_position = 215,
32                      .max_cc = 12,
33                      .max_cp = 40,
34                      .min_cc = 5,
35                      .min_cp = 215
36              };
37              rotor Rotor2 = {
38                      .clear_cycle = 5,
39                      .clear_position = 215,
40                      .max_cc = 12,
41                      .max_cp = 40,
42                      .min_cc = 5,
43                      .min_cp = 215
44              };
45              rotor Rotor3 = {
46                      .clear_cycle = 5,
47                      .clear_position = 215,
48                      .max_cc = 12,
49                      .max_cp = 40,
50                      .min_cc = 5,
51                      .min_cp = 215
52              };
53              rotor Rotor4 = {
54                      .clear_cycle = 5,
55                      .clear_position = 215,
56                      .max_cc = 12,
57                      .max_cp = 40,
58                      .min_cc = 5,
59                      .min_cp = 215
60              };
61
62              void init_timer(void);
63              void update_rotor(rotor* r, uint16_t value);
64
65              ISR(TIMER0_OVF_vect);
66              ISR(TIMER2_OVF_vect);
67
68              #include "../timer.c"
69
70  #endif
```

1.3.2 Source

```
1   /*
2
```



```
3   Harry Beadle
4   D4 Thames
5   Timer (timer.c)
6
7   Timers used for PWM output of the controller.
8   Note: due to hack used to get high precision
9   output from an 8-bit timer the output is
10  required to be low-pass filterd. This has been
11  tested with a 2K7 R and 6N8 C, and works with
12  our ESC.
13
14  Rotor   Timer   Match  Pin
15  ------  ------  -----  ----
16  Rotor1  Timer0  A      PB3
17  Rotor2  Timer0  B      PB4
18  Rotor3  Timer1  A      PD7
19  Rotor4  Timer1  B      PD6
20
21  */
22
23  #include "inc/timer.h"
24
25  void update_rotor(rotor* r, uint16_t value)
26  {
27          // Calculate the clear cycle and position, then
28          // apply it to the given rotor.
29          uint8_t cc = value >> 8;
30          uint8_t cp = value & 255;
31          if (cc > r->max_cc) {
32                  cc = r->max_cc;
33                  if (cp > r->max_cp) {
34                          cp = r->max_cp;
35                  }
36          }
37          if (cc < r->min_cc) {
38                  cc = r->min_cc;
39                  if (cp < r->min_cp) {
40                          cp = r->min_cp;
41                  }
42          }
43          r->cc_next = cc;
44          r->cp_next = cp;
45  }
46
47  void init_timer(void)
48  {
49          // Initalise Timer0
50          // Fast PWM, TOP = 0xFF
51          TCCR0A = 3;
52          // Clear OC0{A, B} on Compare Match
53          TCCR0A |= _BV(COM0A1) | _BV(COM0B1);
54          // Prescaler: f/8
55          TCCR0B = _BV(CS01);
56          // Enable Overflow interrupt.
57          TIMSK0 = _BV(TOIE0);
58          TIFR0 = _BV(TOV0);
59
60          // Initalise Timer2
61          // Fast PWM, TOP = 0xFF
62          TCCR2A = 3;
63          // Clear OC2{A, B} on Compare Match
64          TCCR2A |= _BV(COM2A1) | _BV(COM2B1);
```

```
65          // Prescaler: f/8
66          TCCR2B = _BV(CS21);
67          // Enable Overflow interrupt.
68          TIMSK2 = _BV(TOIE2);
69          TIFR2 = _BV(TOV2);
70
71          // Globally enable interrupts.
72          sei();
73
74          // Set output ports as pullup outputs.
75          DDRB = 0xFF;
76          PORTB = 0xFF;
77          DDRD |= (1<<6 | 1<<7);
78          PORTD |= (1<<6 | 1<<7);
79
80  }
81
82  ISR(TIMER0_OVF_vect, ISR_NOBLOCK)
83  {
84          n0++;
85          if (n0 == 117) {
86                  n0 = 0;
87                  Rotor1.clear_cycle = Rotor1.cc_next;
88                  Rotor1.clear_position = Rotor1.cp_next;
89                  Rotor2.clear_cycle = Rotor2.cc_next;
90                  Rotor2.clear_position = Rotor2.cp_next;
91          }
92
93          if (n0 == Rotor1.clear_cycle) {
94                  OCR0A = Rotor1.clear_position;
95          }
96          else if (n0 < Rotor1.clear_cycle) {
97                  OCR0A = 0xFF;
98          }
99          else {
100                 OCR0A = 0x00;
101         }
102
103         if (n0 == Rotor2.clear_cycle) {
104                 OCR0B = Rotor2.clear_position;
105         }
106         else if (n0 < Rotor2.clear_cycle) {
107                 OCR0B = 0xFF;
108         }
109         else {
110                 OCR0B = 0x00;
111         }
112 }
113
114 ISR(TIMER2_OVF_vect, ISR_NOBLOCK)
115 {
116         n2++;
117         if (n2 == 117) {
118                 n2 = 0;
119                 Rotor3.clear_cycle = Rotor3.cc_next;
120                 Rotor3.clear_position = Rotor3.cp_next;
121                 Rotor4.clear_cycle = Rotor4.cc_next;
122                 Rotor4.clear_position = Rotor4.cp_next;
123         }
124
125         if (n2 == Rotor3.clear_cycle) {
126                 OCR2A = Rotor3.clear_position;
```



```
127         }
128         else if (n2 < Rotor3.clear_cycle) {
129                 OCR2A = 0xFF;
130         }
131         else {
132                 OCR2A = 0x00;
133         }
134
135         if (n2 == Rotor4.clear_cycle) {
136                 OCR2B = Rotor4.clear_position;
137         }
138         else if (n2 < Rotor4.clear_cycle) {
139                 OCR2B = 0xFF;
140         }
141         else {
142                 OCR2B = 0x00;
143         }
144 }
```

1.4 Control

1.4.1 Header

```
1  /*
2
3  Harry Beadle
4  D4 Thames
5  Control (control.h)
6
7  PID Control
8
9  */
10
11 #ifndef _CONTROL_H_
12         #define _CONTROL_H_
13
14                 typedef struct {
15                         double setpoint, time_period;
16                         double o_max, o_min;
17                         double k_p, k_i, k_d;
18                         double i_max;
19                         double e_p, e_i, e_d;
20                 } system;
21
22                 double tick_control(uint16_t in_value, system* s);
23
24         #include "../control.c"
25
26 #endif
```

1.4.2 Source

```
1  /*
2
3  Harry Beadle
4  D4 Thames
5  Control (control.c)
6
7  PID Control
8
9  */
10
11 #include "inc/control.h"
```

```
12
13 double tick_control(uint16_t in_value, system* s)
14 {
15         // Right and left aligned inputs;
16         double input_r, input_l, input;
17         input_r = in_value;
18         input_l = 65535 - in_value;
19         if (input_r < input_l) input = input_r;
20         else input = -input_l;
21
22         // Calculate Errors on this tick.
23         double current_error = s->setpoint - input;
24         s->e_d = (current_error - s->e_p);
25         //s->e_i += s->k_i * current_error;
26         s->e_p = current_error;
27
28         // Handle Maximum Integral Error
29         //if (s->e_i > s->i_max || s->e_i < -s->i_max)
30         //      s->e_i = 0;
31
32         // Calculate Output
33         double output = 0;
34         output += s->k_p * s->e_p;
35         //output += s->e_i; // k_i already c.
36         output += s->k_d * s->e_d;
37
38         // Handle Maximum Output
39         if (output > s->o_max) output = s->o_max;
40         else if (output < s->o_min) output = s->o_min;
41
42         return output;
43 }
```

1.4.3 Systems

```
1
2
3  /*
4
5  Harry Beadle
6  D4 Thames
7  Pitch Control (pitch.h)
8
9  Pitch control system.
10
11 */
12
13 #ifndef _PITCH_H_
14         #define _PITCH_H_
15
16                 system pitch_system = {
17                         .setpoint = 0x0000,
18                         .time_period = 0.004,
19
20                         .o_max = 50,
21                         .o_min = -50,
22
23                         .k_p = 0,
24                         .k_i = 0,
25                         .k_d = 0,
26
27                         .i_max = 0,
```



```
28
29                         .e_p = 0,
30                         .e_i = 0,
31                         .e_d = 0
32                 };
33
34 #endif
```

```
1  /*
2
3  Harry Beadle
4  D4 Thames
5  Roll Control (roll.h)
6
7  Roll control system.
8
9  */
10
11 #ifndef _ROLL_H
12         #define _ROLL_H
13
14                 system roll_system = {
15                         .setpoint = 0x0000,
16                         .time_period = 0.004,
17
18                         .o_max =  10000,
19                         .o_min = -10000,
20
21                         .k_p = 5,
22                         .k_i = 0,
23                         .k_d = 0,
24
25                         .i_max = 0,
26
27                         .e_p = 0,
28                         .e_i = 0,
29                         .e_d = 0
30                 };
31
32 #endif
```

```
1  /*
2
3  Harry Beadle
4  D4 Thames
5  Yaw Control (yaw.h)
6
7  Yaw control system.
8
9  */
10
11 #ifndef _YAW_H_
12         #define _YAW_H_
13
14                 system yaw_system = {
15                         .setpoint = 0,
16                         .time_period = 0,
17
18                         .o_max = 0,
19                         .o_min = 0,
20
21                         .k_p = 0.3,
```

```
                        .k_i = 0.1,
                        .k_d = 0.2,

                        .i_max = 1,

                        .e_p = 0,
                        .e_i = 0,
                        .e_d = 0
                };

#endif
```

1.5 Drone

1.5.1 Header

```
/*

Harry Beadle
D4 Thames
Drone (drone.h)

Headerfile combineing all other modules into a complete system.

*/

#ifndef _DRONE_H_
        #define _DRONE_H_

                #include <avr/io.h>

                // Rotor Communications Symbols
                #define S_R1 0xF4
                #define S_R2 0xF5
                #define S_R3 0xF6
                #define S_R4 0xF7

                // Main Control Loop
                uint16_t thrust = 50000;
                uint16_t pitch = 0;
                uint16_t roll = 0;
                uint16_t yaw = 0;
                uint8_t pitch_update = 0;
                uint8_t roll_update = 0;
                uint8_t yaw_update = 0;
                double pitch_adjust, roll_adjust, yaw_adjust;

                // Include System Specifications
                #include "control.h"
                #include "pitch.h"
                #include "roll.h"
                #include "yaw.h"

                // Include
                #include "rc-symbols.h"
                #include "comms.h"
                #include "buffer.h"

                int main(void);

#endif
```

1.5.2 Source


```
/*

Harry Beadle
D4 Thames
Drone (drone.c)

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

Rotor numbers and directions:

\ /  \ /   /\    1 and 4 are Clockwise
 1    2    /\    2 and 3 are Anticlockwise
/ \==/ \   /\
  ====     /\    1 = thrust + pitch + roll + yaw
\ /==\ /   /\    2 = thrust + pitch - roll - yaw
 3    4    /\    3 = thrust - pitch + roll - yaw
/ \  / \   /\    4 = thrust - pitch - roll + yaw

Control systems are utilised to control the pitch roll and yaw of the
drone. The output of the controller is in servo-compliant PWM.

Rotor   Timer   Match  Pin
------  ------  -----  ----
Rotor1  Timer0  A      PB3
Rotor2  Timer0  B      PB4
Rotor3  Timer2  B      PD6
Rotor4  Timer2  A      PD7

*/

#include "inc/drone.h"

void update_rotor(uint8_t r, uint16_t t)
{
        // Send the Symbol -> High Byte -> Low Byte
        while (!(UCSR1A & _BV(UDRE1)));
        UDR1 = r;
        while (!(UCSR1A & _BV(UDRE1)));
        UDR1 = t >> 8;
        while (!(UCSR1A & _BV(UDRE1)));
        UDR1 = t;
}

int main(void)
{
        // Initalise
        init_comms();
        // Gloally Enable interrupts.
        sei();

        // Loop Forever
        while (1) {
                // Update Control Systems
                pitch_adjust = tick_control(pitch, &pitch_system);
                roll_adjust = tick_control(roll, &roll_system);
                yaw_adjust = tick_control(yaw, &yaw_system);
                // Update Rotors Speeds
                update_rotor(S_R1, thrust + pitch_adjust + roll_adjust + yaw_adjust);
                update_rotor(S_R2, thrust + pitch_adjust - roll_adjust - yaw_adjust);
                update_rotor(S_R3, thrust - pitch_adjust + roll_adjust - yaw_adjust);
                update_rotor(S_R4, thrust - pitch_adjust - roll_adjust + yaw_adjust);
        }
```

```
63 }
```

2 ThamesHost

2.1 Remote Control

2.1.1 Header

```
1  #ifndef _REMOTE_H_
2          #define _REMOTE_H_
3
4                  #include <curses.h>
5                  #include <stdio.h>
6                  #include <stdint.h>
7
8                  #include "spec.h"
9
10                 #define FILENAME "output.hex"
11
12                 #define WIH 80
13                 #define HGT 24
14                 #define PAD 3
15                 #define EMT (WIH-(5*PAD))/4
16
17                 char inchar;
18                 uint8_t thrust, pitch, roll, yaw;
19
20 #endif
```

2.1.2 Source

```
1  /*
2
3  Harry Beadle
4  D4 Thames
5  Bluetooth (serial) Remote Control (remote.c)
6
7  Backup remote for if the Spektrum remote doesn't arrive in time.
8
9  */
10
11 #include "remote.h"
12
13 int main(void)
14 {
15         // Initalise the Screen
16         initscr();
17         noecho();
18         cbreak();
19         nodelay(stdscr, TRUE);
20
21         // Open the serial port
22         FILE* port = fopen(FILENAME, "w");
23
24         // Draw the Basic Screen
25         // mvprintw(0, PAD, "Thrust 0x00");
26         // mvprintw(0, PAD*2 + EMT, "Yaw 0x00");
27         // mvprintw(0, PAD*3 + EMT*2, "Pitch 0x00");
28         // mvprintw(0, PAD*4 + EMT*3, "Roll 0x00");
29         mvprintw(1, PAD-1, "Drone Remote Control");
30         mvprintw(3, PAD, "Thrust 0x00");
31         mvprintw(4, PAD, "Yaw   0x00");
32         mvprintw(5, PAD, "Pitch  0x00");
```



```
33         mvprintw(6, PAD, "Roll   0x00");
34
35         do {
36                 inchar = getch();
37                 switch (inchar) {
38                         case 'w':
39                                 // Increace Thrust
40                                 if (thrust != 63) {
41                                         thrust++;
42                                         fputc(S_THROTTLE & thrust, port);
43                                         mvprintw(3, PAD, "Thrust 0x%02X", thrust);
44                                 }
45                                 break;
46                         case 's':
47                                 // Decreace Thrust
48                                 if (thrust != 0) {
49                                         thrust--;
50                                         fputc(S_THROTTLE & thrust, port);
51                                         mvprintw(3, PAD, "Thrust 0x%02X", thrust);
52                                 }
53                                 break;
54                         case 'a':
55                                 // Yaw right
56                                 if (yaw != 0) {
57                                         yaw--;
58                                         fputc(S_YAW & yaw, port);
59                                         mvprintw(4, PAD, "Yaw    0x%02X", yaw);
60                                 }
61                                 break;
62                         case 'd':
63                                 // Yaw left
64                                 if (yaw != 63) {
65                                         yaw++;
66                                         fputc(S_YAW & yaw, port);
67                                         mvprintw(4, PAD, "Yaw    0x%02X", yaw);
68                                 }
69                                 break;
70                         case 'i':
71                                 // Pitch forward
72                                 if (pitch != 63) {
73                                         pitch++;
74                                         fputc(S_PITCH & pitch, port);
75                                         mvprintw(5, PAD, "Pitch  0x%02X", pitch);
76                                 }
77                                 break;
78                         case 'k':
79                                 // Pitch back
80                                 if (pitch != 0) {
81                                         pitch--;
82                                         fputc(S_PITCH & pitch, port);
83                                         mvprintw(5, PAD, "Pitch  0x%02X", pitch);
84                                 }
85                                 break;
86                         case 'j':
87                                 // Roll left
88                                 if (roll != 0) {
89                                         roll--;
90                                         fputc(S_ROLL & roll, port);
91                                         mvprintw(6, PAD, "Roll   0x%02X", roll);
92                                 }
93                                 break;
94                         case 'l':
```

```
                                // Roll right
                                if (roll != 63) {
                                        roll++;
                                        fputc(S_ROLL & roll, port);
                                        mvprintw(6, PAD, "Roll___0x%02X", roll);
                                }
                                break;
                        case 'c':
                                // Cut thrust (safety measure)
                                thrust = 0;
                                fputc(S_THROTTLE & thrust, port);
                                mvprintw(3, PAD, "Thrust_cut!", thrust);
                                break;
                        default:
                                break;
                }
                fflush(port);
        } while (inchar != 'q');

        return 0;
}
```

2.2 Telemetry

2.2.1 Header

```
// Compile with -lncurses

#ifndef _THAMESHOST_H_
        #define _THAMESHOST_H_

                #include <ncurses.h>
                #include <stdio.h>
                #include <unistd.h>
                #include "tel-comms-spec.h"

                #define PORT_LOCATION "/dev/random"

                #define TEXT_OFFSET 1

                typedef struct {
                        double p, i, d;
                } PID;

                enum {telemetry, tuning} mode;

                char input_byte, ui_input;
                int control_code;
                float input_data;

                void initialise_screen(void);
                int main(void);

#endif
```

2.2.2 Source

```
#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:
```

```
69                      // Get control code and data from input
70                      if (!(input_byte & 0x80)) {
71                              control_code = (input_byte & 0xE0) >> 5;
72                              input_data = (input_byte & 0x1F);
73                      } else {
74                              control_code = (input_byte & 0xF0) >> 4;
75                              input_data = (input_byte & 0x0F);
76                      }
77                      switch (control_code) {
78                              // Remote Control Input
79                              case S_THROTTLE:
80                                      mvprintw( 3, TEXT_OFFSET, "Throttle %f", in
81                                      clrtoeol();
82                                      break;
83                              case S_YAW:
84                                      mvprintw( 4, TEXT_OFFSET, "Yaw      %f", in
85                                      clrtoeol();
86                                      break;
87                              case S_PITCH:
88                                      mvprintw( 5, TEXT_OFFSET, "Pitch    %f", in
89                                      clrtoeol();
90                                      break;
91                              case S_ROLL:
92                                      mvprintw( 6, TEXT_OFFSET, "Roll     %f", in
93                                      clrtoeol();
94                                      break;
95                              // Linear Acceleration
96                              case S_LINX:
97                                      mvprintw( 9, TEXT_OFFSET, "lX %f", input_da
98                                      clrtoeol();
99                                      break;
100                             case S_LINY:
101                                     mvprintw(10, TEXT_OFFSET, "lY %f", input_da
102                                     clrtoeol();
103                                     break;
104                             case S_LINZ:
105                                     mvprintw(11, TEXT_OFFSET, "lZ %f", input_da
106                                     clrtoeol();
107                                     break;
108                             // Angular Acceleration
109                             case S_ANGX:
110                                     mvprintw(14, TEXT_OFFSET, "aX %f", input_da
111                                     clrtoeol();
112                                     break;
113                             case S_ANGY:
114                                     mvprintw(15, TEXT_OFFSET, "aY %f", input_da
115                                     clrtoeol();
116                                     break;
117                             case S_ANGZ:
118                                     mvprintw(16, TEXT_OFFSET, "aZ %f", input_da
119                                     clrtoeol();
120                                     break;
121                             // Start/Stop Signals
122                             case S_STARTF:
123                                     // Unexpected Start Signal
124                                     mvprintw(18, TEXT_OFFSET, "E Unexpected sta
125                                     clrtoeol();
126                                     break;
127                             case S_STOPF:
128                                     // Flight has stopped
129                                     // Change to tuning mode
130                                     mvprintw(18, TEXT_OFFSET, "M Flight Stopped
```



```
131                                     mode = tuning;
132                                     clrtoeol();
133                                     write_base_tun();
134                                     break;
135                             default:
136                                     mvprintw(18, TEXT_OFFSET, "E Unexpected sym
137                                     clrtoeol();
138                                     break;
139                     }
140                     break;
141             case tuning:
142                     ui_input = wgetch(stdscr);
143                     switch (ui_input) {
144                             case 'q':
145                                     // Quit Tuning Mode
146                                     // TODO Send End Symbol
147                                     write_base_tel();
148                                     mode = telemetry;
149                                     break;
150                             case 'p':
151                                     // Set p coefficient
152                                     break;
153                             case 'r':
154                                     // Set i coefficient
155                                     break;
156                             case 'y':
157                                     // Set i error threshold.
158                                     break;
159                             default:
160                                     break;
161                     }
162                     break;
163             }
164             refresh();
165             usleep(1000000);
166     }
167     return 0; // Unreachable
168 }
```

3 Motor Controller

3.0.1 Energia

```
1  /*
2
3  Harry Beadle, Petros Karydogiannis
4  D4 Thames
5  SPI Motor Controller (spi-motor-controller.c)
6
7  */
8
9  #include <Servo.h>
10
11 // Debugging will break the output!!!
12 //#define DEBUG
13
14 Servo Rotor1, Rotor2, Rotor3, Rotor4;
15 bool Update1, Update2, Update3, Update4;
16 int Pos1, Pos2, Pos3, Pos4;
17
18 enum {control, high, low} state, nstate;
19
```

```
20  struct {
21    unsigned int control, high, low;
22    unsigned int data;
23    unsigned int angle;
24  } packet;
25
26  #define S_R1 0xF4
27  #define S_R2 0xF5
28  #define S_R3 0xF6
29  #define S_R4 0xF7
30
31  void tick_comms(void)
32  {
33    if (!Serial.available()) return;
34    state = nstate;
35    uint8_t data = Serial.read();
36    switch (state) {
37      case control:
38        if ((data >= 0xF4) && (data <= 0xF7)) {
39          #ifdef DEBUG
40            Serial.print("Valid Byte ");
41            Serial.println(data, HEX);
42          #endif
43          packet.control  = data;
44          nstate = high;
45        }
46        else {
47          nstate = control;
48        }
49        break;
50      case high:
51        packet.high = data;
52        nstate = low;
53        break;
54      case low:
55        packet.low = data;
56        packet.data = (int) (packet.high << 8 | packet.low) & 0x0000FFFF;
57        packet.angle = map(packet.data, 0, 65536, 1000, 2000);
58        nstate = control;
59  #ifdef DEBUG
60        Serial.print("Update ");
61        Serial.print(packet.control, HEX);
62        Serial.print(" to ");
63        Serial.print(packet.angle);
64        Serial.print(" with ");
65        Serial.println(packet.data, HEX);
66  #endif
67        switch (packet.control) {
68          case S_R1:
69            Pos1 = packet.angle;
70            Update1 = true;
71            break;
72          case S_R2:
73            Pos2 = packet.angle;
74            Update2 = true;
75            break;
76          case S_R3:
77            Pos3 = packet.angle;
78            Update3 = true;
79            break;
80          case S_R4:
81            Pos4 = packet.angle;
```



```
82            Update4 = true;
83            break;
84        }
85        break;
86    }
87  }
88
89  void setup() {
90    // Start Serial
91    Serial.begin(9600);
92
93
94    // Attach the motors to their respective output pins.
95    Rotor1.attach(P1_4);
96    Rotor2.attach(P1_5);
97    Rotor3.attach(P1_6);
98    Rotor4.attach(P2_5);
99
100   // Arm Motors
101   Pos1 = 1000;
102   Pos2 = 1000;
103   Pos3 = 1000;
104   Pos4 = 1000;
105   Rotor1.writeMicroseconds(2000);
106   Rotor2.writeMicroseconds(2000);
107   Rotor3.writeMicroseconds(2000);
108   Rotor4.writeMicroseconds(2000);
109   delay(5200);
110   Rotor1.writeMicroseconds(1000);
111   Rotor2.writeMicroseconds(1000);
112   Rotor3.writeMicroseconds(1000);
113   Rotor4.writeMicroseconds(1000);
114   delay(8200);
115 }
116
117 void loop() {
118   tick_comms();
119   if (Update1) {
120     Rotor1.writeMicroseconds(Pos1);
121     Update1 = !Update1;
122   }
123   if (Update2) {
124     Rotor2.writeMicroseconds(Pos2);
125     Update2 = !Update2;
126   }
127   if (Update3) {
128     Rotor3.writeMicroseconds(Pos3);
129     Update3 = !Update3;
130   }
131   if (Update4) {
132     Rotor4.writeMicroseconds(Pos4);
133     Update4 = !Update4;
134   }
135 }
```

4 PPM to Digital Converter

```
1  /** Ideas on PPM decoder is from
2  http://diydrones.com/profiles/blogs/decoding-multichannel-ppm
3  */
4
5  #include <avr/io.h>
```