

```

21 UBRROH = (F_CPU/(57600*16L)-1) >> 8;
22 UBRRIH = (F_CPU/(9600*16L)-1) >> 8;
23 UBRROL = (F_CPU/(57600*16L)-1);
24 UBRRL = (F_CPU/(9600*16L)-1);
25 // Enable Tx, Rx, Tx Interrupt, Rx Interrupt.
26 UCSR0B = _BV(RXEN0) | _BV(TXEN0) | _BV(TXCIE0);
27 UCSRIB = _BV(RXEN1) | _BV(TXEN1) | _BV(TXCIE1);
28 // 1 Start, 8 Data and 1 Stop bit.
29 UCSR0C = _BV(UCSZ00) | _BV(UCSZ01);
30 UCSRIC = _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     // USART Entry from MPU Decoder
40     state = nstate;
41     temp_byte = UDRL;
42     switch (state) {
43         case control:
44             PORRB |= _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             // PORRB |= _BV(PB1);
64             // nstate = control;
65             // }
66             PORRB &= ~_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 ROLLSYMBOL:
84             roll = high_byte << 8 | low_byte;
85             roll_update = 1;
86             break;
87         case YAWSYMBOL:
88             yaw = high_byte << 8 | low_byte;
89             yaw_update = 1;
90             break;
91         default:
92             break;
93         }
94         nstate = control;
95     }
96 }
97 }

98 // Remote Controller Communications
99 ISR(USART1_RX_vect)
100 {
101     temp_p_byte_rc = UDRL;
102     control_code = 0xC0 & temp_p_byte_rc;
103     data_byte = 0x3F & temp_p_byte_rc;
104     switch (control_code) {
105         case SERVOTILT:
106             if (data_byte != 0) {
107                 thrust <= 11;
108             }
109         break;
110     }
111     case SPITCH:
112         pitch_system.setpoint = data_byte;
113         break;
114     case SROLL:
115         roll_system.setpoint = data_byte;
116         break;
117     case SYAW:
118         yaw_system.setpoint = data_byte;
119         break;
120     }
121 }

1.2 Buffer
1.2.1 Header
1 /*_
2 Harry Beadle
3
4 Dj Thomas
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
21     uint8_t index;
22
23     } buffer8;
24
25     uint8_t buffer8_pop(buffer8* b);
26
27     void buffer8_add(buffer8* b, uint8_t c);
28
29     int buffer8_rdy(buffer8* b);
30
31     /* Set the value at the index location
32      of the buffer to the input value and
33      increment the index.
34      If the buffer[b->index] = c;
35      b->buffer[b->index]++;
36      b->index++;
37      // If we're at the end of the buffer,
38      // go back to the start.
39      // Return the stored value.
40
41     if (b->index == BUFFER_SIZE)
42         b->index = 0;
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
61         // If we're at the end of the buffer,
62         // go back to the start->
63         if (b->outdex == BUFFER_SIZE)
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+1] = c;
73
74             // If we're at the end of the buffer,
75             // go back to the start->
76             if (b->index == BUFFER_SIZE)
77                 b->index = 0;
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         }
85
86     1.3 Timer
87
88     1.3.1 Header
89
90     /*
91
92     3 Harry Beadle
93     4 D4 Thames
94     5 Timer (timer.h)
95
96     Headerfile for the timers outputting PWM to the
97     timer pins.
98
99     */
100
101 #ifndef _TIMER_H_
102 #define _TIMER_H_

```

```

14 #include <avr/io.h>
15 #include <avr/interrupt.h>
16
17 Timer (timer.c)
18
19 uint8_t n0 = 0;
20
21 uint8_t n1 = 0;
22
23 uint8_t min_cc;
24
25 uint8_t max_cc;
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

1.3.2 Source

1 /*
2 */

#typedef struct {
    uint8_t clear_cycle, cc_next;
    uint8_t clear_position, cp_next;
    uint8_t min_cc;
    uint8_t max_cc;
    uint8_t max_cp;
} rotor;

rotor Rotor1 = {
    .clear_cycle = 5,
    .clear_position = 215,
    .max_cc = 12,
    .max_cp = 40,
    .min_cc = 5,
    .min_cp = 215
};

rotor Rotor2 = {
    .clear_cycle = 5,
    .clear_position = 215,
    .max_cc = 12,
    .max_cp = 40,
    .min_cc = 5,
    .min_cp = 215
};

rotor Rotor3 = {
    .clear_cycle = 5,
    .clear_position = 215,
    .max_cc = 12,
    .max_cp = 40,
    .min_cc = 5,
    .min_cp = 215
};

rotor Rotor4 = {
    .clear_cycle = 5,
    .clear_position = 215,
    .max_cc = 12,
    .max_cp = 40,
    .min_cc = 5,
    .min_cp = 215
};

#endif

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 filtered. 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 {
    // Calculate the clear cycle and position, then
    // apply it to the given rotor.
    uint8_t cc = value >> 8;
    uint8_t cp = value & 255;
    if (cc > r->max_cc) {
        cc = r->max_cc;
        if (cp > r->max_cp) {
            cp = r->max_cp;
        }
    }
    if (cc < r->min_cc) {
        cc = r->min_cc;
        if (cp < r->min_cp) {
            cp = r->min_cp;
        }
    }
    r->cc_next = cc;
    r->cp_next = cp;
}

47 void init_timer(void)
48 {
    // Initialise Timer0
    // Fast PWM, TOP = 0xFF
    TCCR0A = 3;
    TCCR0A |= _BV(COM0A1) | _BV(COM0B1);
    // Prescaler: f/8
    TCCR0B = _BV(CS01);
    // Enable Overflow interrupt.
    TIMSK0 = _BV(TOIE0);
    TIFR0 = _BV(TOV0);

60
61
62
63
64
65
66
67
68
69
70

    // Initialise Timer2
    // Fast PWM, TOP = 0xFF
    TCC2A = 3;
    // Clear OC2{A, B} on Compare Match
    TCC2A |= _BV(COM2A1) | _BV(COM2B1);
}

```

```

65 // Prescaler: f/8
66 TCCR2B = _BV(CS21);
67 // Enable Overflow interrupt.
68 TMR2 = _BV(TOIE2);
69 sei();
70 // Globally enable interrupts.
71
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 ISR(TIMER2_OVF_vect, ISR_NOBLOCK)
114 {
115     n2++;
116     if (n2 == 117) {
117         n2 = 0;
118         Rotor3.clear-cycle = Rotor3.cc.next;
119         Rotor3.clear-position = Rotor3.cp.next;
120         Rotor4.clear-cycle = Rotor4.cc.next;
121         Rotor4.clear-position = Rotor4.cp.next;
122     }
123
124     if (n2 == Rotor3.clear-cycle) {
125         OCR2A = Rotor3.clear-position;
126     }

```

```

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 * Harry Beadle
3 * D4 Thames
4 * Control (control.h)
5 * Control (control.c)
6 * PID Control
7 *
8 */
9
10 #ifndef _CONTROL_H_
11 #define _CONTROL_H_
12
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 * Harry Beadle
3 * D4 Thames
4 * Control (control.c)
5 * Control (control.h)
6 * PID Control
7 *
8 */
9
10 #include "inc/control.h"
11

```

```

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

```

#### 1.4.3 Systems

```

1     Harry Beadle
2     D4 Thames
3     /*
4      Harry Beadle
5      D4 Thames
6      Pitch Control (pitch.h)
7
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
35
36 /*
37     Harry Beadle
38     D4 Thames
39     Yaw Control (yaw.h)
40
41     /*
42     Yaw control system.
43
44     /*
45     Harry Beadle
46     D4 Thames
47     Yaw Control (yaw.h)
48
49     /*
50     Yaw control system.
51
52     /*
53     system yaw_system = {
54         .setpoint = 0,
55         .time-period = 0,
56
57         .o_max = 0,
58         .o_min = 0,
59
60         .k_p = 0.3,
61
62     };
63
64 #ifndef _YAW_H_
65 #define _YAW_H_
66
67 /*
68     system yaw_system = {
69         .setpoint = 0,
70         .time-period = 0,
71
72         .o_max = 0,
73         .o_min = 0,
74
75         .k_p = 0.3,
76
77     };
78
79 /*
80     Roll control system.
81
82     /*
83     system roll_system = {
84         .setpoint = 0x0000,
85         .time-period = 0.004,
86
87         .o_max = 10000,
88         .o_min = -10000,
89
90         .k_p = 5,
91         .k_i = 0,
92         .k_d = 0,
93
94         .i_max = 0,
95
96         .e_p = 0,
97         .e_i = 0,
98         .e_d = 0
99     };
100
101 /*
102     Harry Beadle
103     D4 Thames
104     Roll Control (roll.h)
105
106     /*
107     Roll control system.
108
109 /*
110     /*
111     system roll_system = {
112         .setpoint = 0x0000,
113         .time-period = 0.004,
114
115         .o_max = 10000,
116         .o_min = -10000,
117
118         .k_p = 5,
119         .k_i = 0,
120         .k_d = 0,
121
122         .i_max = 0,
123
124         .e_p = 0,
125         .e_i = 0,
126         .e_d = 0
127     };
128
129 /*
130     /*
131     system yaw_system = {
132         .setpoint = 0,
133         .time-period = 0,
134
135         .o_max = 0,
136         .o_min = 0,
137
138         .k_p = 0.3,
139
140     };
141
142 /*
143     /*
144     system yaw_system = {
145         .setpoint = 0,
146         .time-period = 0,
147
148         .o_max = 0,
149         .o_min = 0,
150
151         .k_p = 0.3,
152
153     };
154
155 /*
156     /*
157     system roll_system = {
158         .setpoint = 0x0000,
159         .time-period = 0.004,
160
161         .o_max = 10000,
162         .o_min = -10000,
163
164         .k_p = 5,
165         .k_i = 0,
166         .k_d = 0,
167
168         .i_max = 0,
169
170         .e_p = 0,
171         .e_i = 0,
172         .e_d = 0
173     };
174
175 /*
176     /*
177     system roll_system = {
178         .setpoint = 0x0000,
179         .time-period = 0.004,
180
181         .o_max = 10000,
182         .o_min = -10000,
183
184         .k_p = 5,
185         .k_i = 0,
186         .k_d = 0,
187
188         .i_max = 0,
189
190         .e_p = 0,
191         .e_i = 0,
192         .e_d = 0
193     };
194
195 /*
196     /*
197     system yaw_system = {
198         .setpoint = 0,
199         .time-period = 0,
200
201         .o_max = 0,
202         .o_min = 0,
203
204         .k_p = 0.3,
205
206     };
207
208 /*
209     /*
210     system yaw_system = {
211         .setpoint = 0,
212         .time-period = 0,
213
214         .o_max = 0,
215         .o_min = 0,
216
217         .k_p = 0.3,
218
219     };
220
221 /*
222     /*
223     system roll_system = {
224         .setpoint = 0x0000,
225         .time-period = 0.004,
226
227         .o_max = 10000,
228         .o_min = -10000,
229
230         .k_p = 5,
231         .k_i = 0,
232         .k_d = 0,
233
234         .i_max = 0,
235
236         .e_p = 0,
237         .e_i = 0,
238         .e_d = 0
239     };
240
241 /*
242     /*
243     system roll_system = {
244         .setpoint = 0x0000,
245         .time-period = 0.004,
246
247         .o_max = 10000,
248         .o_min = -10000,
249
250         .k_p = 5,
251         .k_i = 0,
252         .k_d = 0,
253
254         .i_max = 0,
255
256         .e_p = 0,
257         .e_i = 0,
258         .e_d = 0
259     };
260
261 /*
262     /*
263     system yaw_system = {
264         .setpoint = 0,
265         .time-period = 0,
266
267         .o_max = 0,
268         .o_min = 0,
269
270         .k_p = 0.3,
271
272     };
273
274 /*
275     /*
276     system yaw_system = {
277         .setpoint = 0,
278         .time-period = 0,
279
280         .o_max = 0,
281         .o_min = 0,
282
283         .k_p = 0.3,
284
285     };
286
287 /*
288     /*
289     system yaw_system = {
290         .setpoint = 0,
291         .time-period = 0,
292
293         .o_max = 0,
294         .o_min = 0,
295
296         .k_p = 0.3,
297
298     };
299
300 /*
301     /*
302     system yaw_system = {
303         .setpoint = 0,
304         .time-period = 0,
305
306         .o_max = 0,
307         .o_min = 0,
308
309         .k_p = 0.3,
310
311     };
312
313 /*
314     /*
315     system yaw_system = {
316         .setpoint = 0,
317         .time-period = 0,
318
319         .o_max = 0,
320         .o_min = 0,
321
322         .k_p = 0.3,
323
324     };
325
326 /*
327     /*
328     system yaw_system = {
329         .setpoint = 0,
330         .time-period = 0,
331
332         .o_max = 0,
333         .o_min = 0,
334
335         .k_p = 0.3,
336
337     };
338
339 /*
340     /*
341     system yaw_system = {
342         .setpoint = 0,
343         .time-period = 0,
344
345         .o_max = 0,
346         .o_min = 0,
347
348         .k_p = 0.3,
349
350     };
351
352 /*
353     /*
354     system yaw_system = {
355         .setpoint = 0,
356         .time-period = 0,
357
358         .o_max = 0,
359         .o_min = 0,
360
361         .k_p = 0.3,
362
363     };
364
365 /*
366     /*
367     system yaw_system = {
368         .setpoint = 0,
369         .time-period = 0,
370
371         .o_max = 0,
372         .o_min = 0,
373
374         .k_p = 0.3,
375
376     };
377
378 /*
379     /*
380     system yaw_system = {
381         .setpoint = 0,
382         .time-period = 0,
383
384         .o_max = 0,
385         .o_min = 0,
386
387         .k_p = 0.3,
388
389     };
390
391 /*
392     /*
393     system yaw_system = {
394         .setpoint = 0,
395         .time-period = 0,
396
397         .o_max = 0,
398         .o_min = 0,
399
400         .k_p = 0.3,
401
402     };
403
404 /*
405     /*
406     system yaw_system = {
407         .setpoint = 0,
408         .time-period = 0,
409
410         .o_max = 0,
411         .o_min = 0,
412
413         .k_p = 0.3,
414
415     };
416
417 /*
418     /*
419     system yaw_system = {
420         .setpoint = 0,
421         .time-period = 0,
422
423         .o_max = 0,
424         .o_min = 0,
425
426         .k_p = 0.3,
427
428     };
429
430 /*
431     /*
432     system yaw_system = {
433         .setpoint = 0,
434         .time-period = 0,
435
436         .o_max = 0,
437         .o_min = 0,
438
439         .k_p = 0.3,
440
441     };
442
443 /*
444     /*
445     system yaw_system = {
446         .setpoint = 0,
447         .time-period = 0,
448
449         .o_max = 0,
450         .o_min = 0,
451
452         .k_p = 0.3,
453
454     };
455
456 /*
457     /*
458     system yaw_system = {
459         .setpoint = 0,
460         .time-period = 0,
461
462         .o_max = 0,
463         .o_min = 0,
464
465         .k_p = 0.3,
466
467     };
468
469 /*
470     /*
471     system yaw_system = {
472         .setpoint = 0,
473         .time-period = 0,
474
475         .o_max = 0,
476         .o_min = 0,
477
478         .k_p = 0.3,
479
480     };
481
482 /*
483     /*
484     system yaw_system = {
485         .setpoint = 0,
486         .time-period = 0,
487
488         .o_max = 0,
489         .o_min = 0,
490
491         .k_p = 0.3,
492
493     };
494
495 /*
496     /*
497     system yaw_system = {
498         .setpoint = 0,
499         .time-period = 0,
500
501         .o_max = 0,
502         .o_min = 0,
503
504         .k_p = 0.3,
505
506     };
507
508 /*
509     /*
510     system yaw_system = {
511         .setpoint = 0,
512         .time-period = 0,
513
514         .o_max = 0,
515         .o_min = 0,
516
517         .k_p = 0.3,
518
519     };
520
521 /*
522     /*
523     system yaw_system = {
524         .setpoint = 0,
525         .time-period = 0,
526
527         .o_max = 0,
528         .o_min = 0,
529
530         .k_p = 0.3,
531
532     };
533
534 /*
535     /*
536     system yaw_system = {
537         .setpoint = 0,
538         .time-period = 0,
539
540         .o_max = 0,
541         .o_min = 0,
542
543         .k_p = 0.3,
544
545     };
546
547 /*
548     /*
549     system yaw_system = {
550         .setpoint = 0,
551         .time-period = 0,
552
553         .o_max = 0,
554         .o_min = 0,
555
556         .k_p = 0.3,
557
558     };
559
560 /*
561     /*
562     system yaw_system = {
563         .setpoint = 0,
564         .time-period = 0,
565
566         .o_max = 0,
567         .o_min = 0,
568
569         .k_p = 0.3,
570
571     };
572
573 /*
574     /*
575     system yaw_system = {
576         .setpoint = 0,
577         .time-period = 0,
578
579         .o_max = 0,
580         .o_min = 0,
581
582         .k_p = 0.3,
583
584     };
585
586 /*
587     /*
588     system yaw_system = {
589         .setpoint = 0,
590         .time-period = 0,
591
592         .o_max = 0,
593         .o_min = 0,
594
595         .k_p = 0.3,
596
597     };
598
599 /*
600     /*
601     system yaw_system = {
602         .setpoint = 0,
603         .time-period = 0,
604
605         .o_max = 0,
606         .o_min = 0,
607
608         .k_p = 0.3,
609
610     };
611
612 /*
613     /*
614     system yaw_system = {
615         .setpoint = 0,
616         .time-period = 0,
617
618         .o_max = 0,
619         .o_min = 0,
620
621         .k_p = 0.3,
622
623     };
624
625 /*
626     /*
627     system yaw_system = {
628         .setpoint = 0,
629         .time-period = 0,
630
631         .o_max = 0,
632         .o_min = 0,
633
634         .k_p = 0.3,
635
636     };
637
638 /*
639     /*
640     system yaw_system = {
641         .setpoint = 0,
642         .time-period = 0,
643
644         .o_max = 0,
645         .o_min = 0,
646
647         .k_p = 0.3,
648
649     };
650
651 /*
652     /*
653     system yaw_system = {
654         .setpoint = 0,
655         .time-period = 0,
656
657         .o_max = 0,
658         .o_min = 0,
659
660         .k_p = 0.3,
661
662     };
663
664 /*
665     /*
666     system yaw_system = {
667         .setpoint = 0,
668         .time-period = 0,
669
670         .o_max = 0,
671         .o_min = 0,
672
673         .k_p = 0.3,
674
675     };
676
677 /*
678     /*
679     system yaw_system = {
680         .setpoint = 0,
681         .time-period = 0,
682
683         .o_max = 0,
684         .o_min = 0,
685
686         .k_p = 0.3,
687
688     };
689
690 /*
691     /*
692     system yaw_system = {
693         .setpoint = 0,
694         .time-period = 0,
695
696         .o_max = 0,
697         .o_min = 0,
698
699         .k_p = 0.3,
700
701     };
702
703 /*
704     /*
705     system yaw_system = {
706         .setpoint = 0,
707         .time-period = 0,
708
709         .o_max = 0,
710         .o_min = 0,
711
712         .k_p = 0.3,
713
714     };
715
716 /*
717     /*
718     system yaw_system = {
719         .setpoint = 0,
720         .time-period = 0,
721
722         .o_max = 0,
723         .o_min = 0,
724
725         .k_p = 0.3,
726
727     };
728
729 /*
730     /*
731     system yaw_system = {
732         .setpoint = 0,
733         .time-period = 0,
734
735         .o_max = 0,
736         .o_min = 0,
737
738         .k_p = 0.3,
739
740     };
741
742 /*
743     /*
744     system yaw_system = {
745         .setpoint = 0,
746         .time-period = 0,
747
748         .o_max = 0,
749         .o_min = 0,
750
751         .k_p = 0.3,
752
753     };
754
755 /*
756     /*
757     system yaw_system = {
758         .setpoint = 0,
759         .time-period = 0,
760
761         .o_max = 0,
762         .o_min = 0,
763
764         .k_p = 0.3,
765
766     };
767
768 /*
769     /*
770     system yaw_system = {
771         .setpoint = 0,
772         .time-period = 0,
773
774         .o_max = 0,
775         .o_min = 0,
776
777         .k_p = 0.3,
778
779     };
780
781 /*
782     /*
783     system yaw_system = {
784         .setpoint = 0,
785         .time-period = 0,
786
787         .o_max = 0,
788         .o_min = 0,
789
790         .k_p = 0.3,
791
792     };
793
794 /*
795     /*
796     system yaw_system = {
797         .setpoint = 0,
798         .time-period = 0,
799
800         .o_max = 0,
801         .o_min = 0,
802
803         .k_p = 0.3,
804
805     };
806
807 /*
808     /*
809     system yaw_system = {
810         .setpoint = 0,
811         .time-period = 0,
812
813         .o_max = 0,
814         .o_min = 0,
815
816         .k_p = 0.3,
817
818     };
819
820 /*
821     /*
822     system yaw_system = {
823         .setpoint = 0,
824         .time-period = 0,
825
826         .o_max = 0,
827         .o_min = 0,
828
829         .k_p = 0.3,
830
831     };
832
833 /*
834     /*
835     system yaw_system = {
836         .setpoint = 0,
837         .time-period = 0,
838
839         .o_max = 0,
840         .o_min = 0,
841
842         .k_p = 0.3,
843
844     };
845
846 /*
847     /*
848     system yaw_system = {
849         .setpoint = 0,
850         .time-period = 0,
851
852         .o_max = 0,
853         .o_min = 0,
854
855         .k_p = 0.3,
856
857     };
858
859 /*
860     /*
861     system yaw_system = {
862         .setpoint = 0,
863         .time-period = 0,
864
865         .o_max = 0,
866         .o_min = 0,
867
868         .k_p = 0.3,
869
870     };
871
872 /*
873     /*
874     system yaw_system = {
875         .setpoint = 0,
876         .time-period = 0,
877
878         .o_max = 0,
879         .o_min = 0,
880
881         .k_p = 0.3,
882
883     };
884
885 /*
886     /*
887     system yaw_system = {
888         .setpoint = 0,
889         .time-period = 0,
890
891         .o_max = 0,
892         .o_min = 0,
893
894         .k_p = 0.3,
895
896     };
897
898 /*
899     /*
900     system yaw_system = {
901         .setpoint = 0,
902         .time-period = 0,
903
904         .o_max = 0,
905         .o_min = 0,
906
907         .k_p = 0.3,
908
909     };
910
911 /*
912     /*
913     system yaw_system = {
914         .setpoint = 0,
915         .time-period = 0,
916
917         .o_max = 0,
918         .o_min = 0,
919
920         .k_p = 0.3,
921
922     };
923
924 /*
925     /*
926     system yaw_system = {
927         .setpoint = 0,
928         .time-period = 0,
929
930         .o_max = 0,
931         .o_min = 0,
932
933         .k_p = 0.3,
934
935     };
936
937 /*
938     /*
939     system yaw_system = {
940         .setpoint = 0,
941         .time-period = 0,
942
943         .o_max = 0,
944         .o_min = 0,
945
946         .k_p = 0.3,
947
948     };
949
950 /*
951     /*
952     system yaw_system = {
953         .setpoint = 0,
954         .time-period = 0,
955
956         .o_max = 0,
957         .o_min = 0,
958
959         .k_p = 0.3,
960
961     };
962
963 /*
964     /*
965     system yaw_system = {
966         .setpoint = 0,
967         .time-period = 0,
968
969         .o_max = 0,
970         .o_min = 0,
971
972         .k_p = 0.3,
973
974     };
975
976 /*
977     /*
978     system yaw_system = {
979         .setpoint = 0,
980         .time-period = 0,
981
982         .o_max = 0,
983         .o_min = 0,
984
985         .k_p = 0.3,
986
987     };
988
989 /*
990     /*
991     system yaw_system = {
992         .setpoint = 0,
993         .time-period = 0,
994
995         .o_max = 0,
996         .o_min = 0,
997
998         .k_p = 0.3,
999
1000    };
1001
1002 /*
1003     /*
1004     system yaw_system = {
1005         .setpoint = 0,
1006         .time-period = 0,
1007
1008         .o_max = 0,
1009         .o_min = 0,
1010
1011         .k_p = 0.3,
1012
1013    };
1014
1015 /*
1016     /*
1017     system yaw_system = {
1018         .setpoint = 0,
1019         .time-period = 0,
1020
1021         .o_max = 0,
1022         .o_min = 0,
1023
1024         .k_p = 0.3,
1025
1026    };
1027
1028 /*
1029     /*
1030     system yaw_system = {
1031         .setpoint = 0,
1032         .time-period = 0,
1033
1034         .o_max = 0,
1035         .o_min = 0,
1036
1037         .k_p = 0.3,
1038
1039    };
1040
1041 /*
1042     /*
1043     system yaw_system = {
1044         .setpoint = 0,
1045         .time-period = 0,
1046
1047         .o_max = 0,
1048         .o_min = 0,
1049
1050         .k_p = 0.3,
1051
1052    };
1053
1054 /*
1055     /*
1056     system yaw_system = {
1057         .setpoint = 0,
1058         .time-period = 0,
1059
1060         .o_max = 0,
1061         .o_min = 0,
1062
1063         .k_p = 0.3,
1064
1065    };
1066
1067 /*
1068     /*
1069     system yaw_system = {
1070         .setpoint = 0,
1071         .time-period = 0,
1072
1073         .o_max = 0,
1074         .o_min = 0,
1075
1076         .k_p = 0.3,
1077
1078    };
1079
1080 /*
1081     /*
1082     system yaw_system = {
1083         .setpoint = 0,
1084         .time-period = 0,
1085
1086         .o_max = 0,
1087         .o_min = 0,
1088
1089         .k_p = 0.3,
1090
1091    };
1092
1093 /*
1094     /*
1095     system yaw_system = {
1096         .setpoint = 0,
1097         .time-period = 0,
1098
1099         .o_max = 0,
1100         .o_min = 0,
1101
1102         .k_p = 0.3,
1103
1104    };
1105
1106 /*
1107     /*
1108     system yaw_system = {
1109         .setpoint = 0,
1110         .time-period = 0,
1111
1112         .o_max = 0,
1113         .o_min = 0,
1114
1115         .k_p = 0.3,
1116
1117    };
1118
1119 /*
1120     /*
1121     system yaw_system = {
1122         .setpoint = 0,
1123         .time-period = 0,
1124
1125         .o_max = 0,
1126         .o_min = 0,
1127
1128         .k_p = 0.3,
1129
1130    };
1131
1132 /*
1133     /*
1134     system yaw_system = {
1135         .setpoint = 0,
1136         .time-period = 0,
1137
1138         .o_max = 0,
1139         .o_min = 0,
1140
1141         .k_p = 0.3,
1142
1143    };
1144
1145 /*
1146     /*
1147     system yaw_system = {
1148         .setpoint = 0,
1149         .time-period = 0,
1150
1151         .o_max = 0,
1152         .o_min = 0,
1153
1154         .k_p = 0.3,
1155
1156    };
1157
1158 /*
1159     /*
1160     system yaw_system = {
1161         .setpoint = 0,
1162         .time-period = 0,
1163
1164         .o_max = 0,
1165         .o_min = 0,
1166
1167         .k_p = 0.3,
1168
1169    };
1170
1171 /*
1172     /*
1173     system yaw_system = {
1174         .setpoint = 0,
1175         .time-period = 0,
1176
1177         .o_max = 0,
1178         .o_min = 0,
1179
1180         .k_p = 0.3,
1181
1182    };
1183
1184 /*
1185     /*
1186     system yaw_system = {
1187         .setpoint = 0,
1188         .time-period = 0,
1189
1190         .o_max = 0,
1191         .o_min = 0,
1192
1193         .k_p = 0.3,
1194
1195    };
1196
1197 /*
1198     /*
1199     system yaw_system = {
1200         .setpoint = 0,
1201         .time-period = 0,
1202
1203         .o_max = 0,
1204         .o_min = 0,
1205
1206         .k_p = 0.3,
1207
1208    };
1209
1210 /*
1211     /*
1212     system yaw_system = {
1213         .setpoint = 0,
1214         .time-period = 0,
1215
1216         .o_max = 0,
1217         .o_min = 0,
1218
1219         .k_p = 0.3,
1220
1221    };
1222
1223 /*
1224     /*
1225     system yaw_system = {
1226         .setpoint = 0,
1227         .time-period = 0,
1228
1229         .o_max = 0,
1230         .o_min = 0,
1231
1232         .k_p = 0.3,
1233
1234    };
1235
1236 /*
1237     /*
1238     system yaw_system = {
1239         .setpoint = 0,
1240         .time-period = 0,
1241
1242         .o_max = 0,
1243         .o_min = 0,
1244
1245         .k_p = 0.3,
1246
1247    };
1248
1249 /*
1250     /*
1251     system yaw_system = {
1252         .setpoint = 0,
1253         .time-period = 0,
1254
1255         .o_max = 0,
1256         .o_min = 0,
1257
1258         .k_p = 0.3,
1259
1260    };
1261
1262 /*
1263     /*
1264     system yaw_system = {
1265         .setpoint = 0,
1266         .time-period = 0,
1267
1268         .o_max = 0,
1269         .o_min = 0,
1270
1271         .k_p = 0.3,
1272
1273    };
1274
1275 /*
1276     /*
1277     system yaw_system = {
1278         .setpoint = 0,
1279         .time-period = 0,
1280
1281         .o_max = 0,
1282         .o_min = 0,
1283
1284         .k_p = 0.3,
1285
1286    };
1287
1288 /*
1289     /*
1290     system yaw_system = {
1291         .setpoint = 0,
1292         .time-period = 0,
1293
1294         .o_max = 0,
1295         .o_min = 0,
1296
1297         .k_p = 0.3,
1298
1299    };
1300
1301 /*
1302     /*
1303     system yaw_system = {
1304         .setpoint = 0,
1305         .time-period = 0,
1306
1307         .o_max = 0,
1308         .o_min = 0,
1309
1310         .k_p = 0.3,
1311
1312    };
1313
1314 /*
1315     /*
1316     system yaw_system = {
1317         .setpoint = 0,
1318         .time-period = 0,
1319
1320         .o_max = 0,
1321         .o_min = 0,
1322
1323         .k_p = 0.3,
1324
1325    };
1326
1327 /*
1328     /*
1329     system yaw_system = {
1330         .setpoint = 0,
1331         .time-period = 0,
1332
1333         .o_max = 0,
1334         .o_min = 0,
1335
1336         .k_p = 0.3,
1337
1338    };
1339
1340 /*
1341     /*
1342     system yaw_system = {
1343         .setpoint = 0,
1344         .time-period = 0,
1345
1346         .o_max = 0,
1347         .o_min = 0,
1348
1349         .k_p = 0.3,
1350
1351    };
1352
1353 /*
1354     /*
1355     system yaw_system = {
1356         .setpoint = 0,
1357         .time-period = 0,
1358
1359         .o_max = 0,
1360         .o_min = 0,
1361
1362         .k_p = 0.3,
1363
1364    };
1365
1366 /*
1367     /*
1368     system yaw_system = {
1369         .setpoint = 0,
1370         .time-period = 0,
1371
1372         .o_max = 0,
1373         .o_min = 0,
1374
1375         .k_p = 0.3,
1376
1377    };
1378
1379 /*
1380     /*
1381     system yaw_system = {
1382         .setpoint = 0,
1383         .time-period = 0,
1384
```

```

22 .k_i = 0.1,
23 .k_d = 0.2,
24
25 .i_max = 1,
26
27 .e_P = 0,
28 .e_I = 0,
29 .e_D = 0
30 }
31
32 #endif
33
34 // 1.5 Drone
35 // 1.5.1 Header
36
37 Headerfile combining all other modules into a complete system.
38
39 /*

40 3 Harry Beadle
41 4 D4 Thames
42 5 Drone (drone.h)
43
44
45 #ifndef DRONE_H_
46
47 #include <avr/io.h>
48
49 */
50
51
52 // Rotor Communications Symbols
53
54 #define S_R1 0xF4
55 #define S_R2 0xF5
56 #define S_R3 0xF6
57 #define S_R4 0xF7
58
59 // Main Control Loop
60
61 uint16_t thrust = 50000;
62 uint16_t pitch = 0;
63 uint16_t roll = 0;
64 uint16_t yaw = 0;
65
66 uint8_t pitch_update = 0;
67 uint8_t roll_update = 0;
68 uint8_t yaw_update = 0;
69
70 double pitch_adjust, roll_adjust, yaw_adjust;
71
72 // Include System Specifications
73 // #include "control.h"
74 // #include "pitch.h"
75 // #include "roll.h"
76 // #include "yaw.h"
77
78 // Include
79 // #include "rc-symbols.h"
80 // #include "comms.h"
81 // #include "buffer.h"
82
83 int main(void)
84 {
85
86     // Update Control Systems
87     pitch_adjust = tick_control(pitch, &pitch_system);
88     roll_adjust = tick_control(roll, &roll_system);
89     yaw_adjust = tick_control(yaw, &yaw_system);
90
91     // Update Rotors Speeds
92     update_rotor(S_R1, thrust + pitch_adjust + roll_adjust + yaw_adjust);
93     update_rotor(S_R2, thrust + pitch_adjust - roll_adjust - yaw_adjust);
94     update_rotor(S_R3, thrust - pitch_adjust + roll_adjust - yaw_adjust);
95     update_rotor(S_R4, thrust - pitch_adjust - roll_adjust + yaw_adjust);
96
97 }
98
99 // 1.5.2 Source

```

1 /\*  
2 \*  
3 *Harry Beadle*  
4 *D4 Thames*  
5 *Drone (drone.c)*  
6 Combines the function of all other modules into a complete system.  
7  
8 Rotor numbers and directions:  
9  
10 \ / \ / 1 and 4 are Clockwise  
11 1 ==/ \ ^ and 3 are Anticlockwise  
12 / \ ==/ \ /  
13 \ ==/ \ / 1 = thrust + pitch + roll + yaw  
14 \ ==/ \ / 2 = thrust + pitch - roll - yaw  
15 \ ==/ \ / 3 = thrust - pitch + roll + yaw  
16 \ ==/ \ / 4 = thrust - pitch - roll + yaw  
17 / \ / \ / 4 = thrust - pitch - roll + yaw  
18  
19 Control systems are utilised to control the pitch roll and yaw of the  
20 drone. The output of the controller is in servo-compliant PWM.  
21 Rotor Timer Match Pin  
22 Rotor1 Timer0 A PB3  
23 Rotor2 Timer0 B PB4  
24 Rotor3 Timer2 B PD6  
25 Rotor4 Timer2 A PD7  
26  
27  
28 \*/  
29  
30 #include "inc/drone.h"  
31
32 void update\_rotor(uint8\_t r, uint16\_t t)  
33 {
34 // Send the Symbol -> High Byte -> Low Byte  
35 while (!(UCSRA & \_BV(UDRE1))){  
36 UDRL = r;  
37 UDRL = t;  
38 while (!(UCSRA & \_BV(UDRE1))){  
39 UDRL = t >> 8;  
40 while (!(UCSRA & \_BV(UDRE1))){  
41 UDRL = t;  
42 }  
43 }  
44 int main(void)  
45 {  
46 // Initialise  
47 init\_comms();  
48 // Globally Enable interrupts.  
49 sei();  
50
51 // **Loop Forever.**  
52 while (1){  
53 // **Update Control Systems**  
54 pitch\_adjust = tick\_control(pitch, &pitch\_system);  
55 roll\_adjust = tick\_control(roll, &roll\_system);  
56 yaw\_adjust = tick\_control(yaw, &yaw\_system);  
57
58 // **Update Rotors Speeds**  
59 update\_rotor(S\_R1, thrust + pitch\_adjust + roll\_adjust + yaw\_adjust);  
60 update\_rotor(S\_R2, thrust + pitch\_adjust - roll\_adjust - yaw\_adjust);  
61 update\_rotor(S\_R3, thrust - pitch\_adjust + roll\_adjust - yaw\_adjust);  
62 update\_rotor(S\_R4, thrust - pitch\_adjust - roll\_adjust + yaw\_adjust);  
63 }
64 }
65 }

```

63 }
2 ThameHost
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 WTH 80
13 #define HGT 24
14 #define PAD 3
15 #define EMT (WTH-(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 Themes
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     // Initialise the Screen
16     initscr();
17     noecho();
18     cbreak();
19     nodelay();
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
34     do {
35         inchar = getch();
36
37         switch (inchar) {
38             case 'w':
39                 // Increase Thrust
40                 if (thrust != 63) {
41                     thrust++;
42                     fputchar(STHRROTL & thrust, port);
43                     myprintw(3, PAD, "Thrust_0x%02X", thrust);
44                 }
45                 break;
46             case 's':
47                 // Decrease Thrust
48                 if (thrust != 0) {
49                     thrust--;
50                     fputchar(STHRROTL & thrust, port);
51                     myprintw(3, PAD, "Thrust_0x%02X", thrust);
52                 }
53                 break;
54             case 'a':
55                 // Yaw right
56                 if (yaw != 0) {
57                     yaw--;
58                     fputchar(SYAW & yaw, port);
59                     myprintw(4, PAD, "Yaw_0x%02X", yaw);
60                 }
61                 break;
62             case 'd':
63                 // Yaw left
64                 if (yaw != 63) {
65                     yaw++;
66                     fputchar(SYAW & yaw, port);
67                     myprintw(4, PAD, "Yaw_0x%02X", yaw);
68                 }
69                 break;
70             case 'i':
71                 // Pitch forward
72                 if (pitch != 63) {
73                     pitch++;
74                     fputchar(SPITCH & pitch, port);
75                     myprintw(5, PAD, "Pitch_0x%02X", pitch);
76                 }
77                 break;
78             case 'k':
79                 // Pitch back
80                 if (pitch != 0) {
81                     pitch--;
82                     fputchar(SPITCH & pitch, port);
83                     myprintw(5, PAD, "Pitch_0x%02X", pitch);
84                 }
85                 break;
86             case 'j':
87                 // Roll left
88                 if (roll != 0) {
89                     roll--;
90                     fputchar(SROLL & roll, port);
91                     myprintw(6, PAD, "Roll_0x%02X", roll);
92                 }
93                 break;
94             case 'l':
95                 break;
96         }
97     }
98 }
```

```

95 // Roll right
96 if (roll == 63) {
97   roll++;
98   fpuc(SROLL & roll, port);
99   mvprintw(6, PAD, "Roll---0x%02X", roll);
100 }
101 break;
102 }
103 case 'c':
104   // Cut thrust (safety measure)
105   thrust = 0;
106   fpuc(STHROTTLE & thrust, port);
107   mvprintw(3, PAD, "Thrust_cut!", thrust);
108   break;
109 default:
110   break;
111   fflush(port);
112 } while (inchar != 'q');
113
114 }
115 }

2.2 Telemetry

2.2.1 Header

1 // Compile with -lncurses
2
3 #ifndef _THAMESHOST_H_
4 #define _THAMESHOST_H_
5
6 #include <lncurses.h>
7 #include <stdio.h>
8 #include <unistd.h>
9 #include "tel-comms-spec.h"
10
11 #define PORTLOCATION "/dev/random"
12
13 #define TEXTOFFSET 1
14
15 typedef struct {
16   double p, i, d;
17 } PID;
18
19 enum { telemetry, tuning} mode;
20
21 char input_byte, ui_input;
22 int control_code;
23 float input_data;
24
25 void initialise_screen(void);
26
27
28#endif

2.2.2 Source

1 #include "thameshost.h"
2
3 void initialise_screen(void)
4 {
5   initscr();
6   noecho();

```

```

7   cbreak();
8   nodelay(stdscr, TRUE);
9 }
10
11 PID* write
12
13 void write_base_tun(void)
14 {
15   clear();
16   attron(A_BOLD);
17   mvprintw(0, TEXT_OFFSET, "Tuning_System");
18   mvprintw(1, TEXT_OFFSET, "Flight_not_available_while_tuning.");
19   attron(A_BOLD);
20   mvprintw(3, TEXT_OFFSET, "p---Edit_Pitch_System");
21   mvprintw(4, TEXT_OFFSET, "r---Edit_Roll_System");
22   mvprintw(5, TEXT_OFFSET, "y---Edit_Yaw_System");
23 }
24
25 void write_base_tel(void)
26 {
27   clear();
28   attron(A_BOLD);
29   mvprintw(0, TEXT_OFFSET, "Telemetry_System");
30   attron(A_BOLD);
31   attron(A_UNDERLINE);
32   mvprintw(2, TEXT_OFFSET, "Remote_Control_Inputs");
33   attron(A_UNDERLINE);
34   mvprintw(3, TEXT_OFFSET, "Throttle");
35   mvprintw(4, TEXT_OFFSET, "Yaw");
36   mvprintw(5, TEXT_OFFSET, "Pitch");
37   mvprintw(6, TEXT_OFFSET, "Roll");
38   attron(A_UNDERLINE);
39   mvprintw(8, TEXT_OFFSET, "Linear_Acceleration");
40   attron(A_UNDERLINE);
41   mvprintw(9, TEXT_OFFSET, "IX");
42   mvprintw(10, TEXT_OFFSET, "IY");
43   mvprintw(11, TEXT_OFFSET, "IZ");
44   attron(A_UNDERLINE);
45   mvprintw(13, TEXT_OFFSET, "Angular_Acceleration");
46   attron(A_UNDERLINE);
47   mvprintw(14, TEXT_OFFSET, "aX");
48   mvprintw(15, TEXT_OFFSET, "aY");
49   mvprintw(16, TEXT_OFFSET, "aZ");
50 }
51
52 int main(void)
53 {
54   // Open File, Initialise System
55   FILE* serial_port = fopen(PORTLOCATION, "r");
56   mode = telemetry;
57
58   // Initialise Screen
59   initialise_screen();
60   write_base_tel();
61
62   while (1) {
63     // Read from file
64     input_byte = fgetc(serial_port);
65
66     // Deal with input
67     switch (mode) {
68       case telemetry:

```

```

69 // Get control code and data from input
70 if (!(input.byte & 0x80)) {
71     control_code = (input.byte & 0xED) >> 5;
72     input.data = (input.byte & 0x1F);
73 } else {
74     control_code = (input.byte & 0xFF) >> 4;
75     input.data = (input.byte & 0x0F);
76 }
77 switch (control_code) {
78     // Remote Control Input
79     case STIRROTTE:
80         mvprintw( 3, TEXT_OFFSET, "Throttle %f", in
81             control_code );
82         break;
83     case SYAW:
84         mvprintw( 4, TEXT_OFFSET, "Yaw~~~~~%f", in
85             control_code );
86         break;
87     case SPITCH:
88         mvprintw( 5, TEXT_OFFSET, "Pitch~~~~%f", in
89             control_code );
90         break;
91     case SROLL:
92         mvprintw( 6, TEXT_OFFSET, "Roll~~~~%f", in
93             control_code );
94         break;
95     // Linear Acceleration
96     case SLINX:
97         mvprintw( 9, TEXT_OFFSET, "IX.%f", input_da
98             control_code );
99         break;
100    case SLINY:
101        mvprintw(10, TEXT_OFFSET, "IY.%f", input_da
102            control_code );
103        break;
104    case SLINZ:
105        mvprintw(11, TEXT_OFFSET, "IZ.%f", input_da
106            control_code );
107        break;
108    case SANGX:
109        mvprintw(14, TEXT_OFFSET, "aX.%f", input_da
110            control_code );
111        break;
112    case SANGY:
113        mvprintw(15, TEXT_OFFSET, "aY.%f", input_da
114            control_code );
115        break;
116    case SANZZ:
117        mvprintw(16, TEXT_OFFSET, "aZ.%f", input_da
118            control_code );
119        break;
120    case SSTARTF:
121        mvprintw(17, TEXT_OFFSET, "E_Unexpected_st
122        control_code );
123        break;
124    case SSTOPF:
125        mvprintw(18, TEXT_OFFSET, "E_Flight_Stopp
126        control_code );
127        break;
128    // Flight has stopped
129    // Change to tuning mode
130    mvprintw(18, TEXT_OFFSET, "ML_Flight_Stopp

```

```

131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168 }

3 Motor Controller
3.0.1 Energia
1 /*
2
3 Harry Beadle, Petros Karydogiannis
4 D4 Thame
5 SPI Motor Controller (spi-motor-controller.c)
6
7 */
8 #include <Servo.h>
9
10 // Debugging will break the output!!!
11 // #define DEBUG
12
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 } packet;
24
25 #define S_R1 0xF4
26 #define S_R2 0xF5
27 #define S_R3 0xF6
28 #define S_R4 0xF7
29
30 void tick_comms(void)
31 {
32     if (!Serial.available()) return;
33     state = nstate;
34     uint8_t data = Serial.read();
35     switch (state) {
36         case control:
37             if ((data >= 0xF4) && (data <= 0xF7)) {
38                 #ifdef DEBUG
39                     Serial.print("Valid_Byte_");
40                     Serial.println(data, HEX);
41                 #endif
42             }
43             packet.control = data;
44             nstate = high;
45         case high:
46             else {
47                 nstate = control;
48             }
49             break;
50         case low:
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             nstate = control;
58     }
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
83     }
84     break;
85 }
86
87 void setup() {
88     // Start Serial
89     // Serial.begin(9600);
90
91     // Attach the motors to their respective output pins.
92     // Attach the motors to their respective output pins.
93     Rotor1.attach(P1_4);
94     Rotor2.attach(P1_5);
95     Rotor3.attach(P1_6);
96     Rotor4.attach(P2_5);
97
98     // Arm Motors
99     Pos1 = 1000;
100    Pos2 = 1000;
101    Pos3 = 1000;
102    Pos4 = 1000;
103
104    Rotor1.writeMicroseconds(1000);
105    Rotor2.writeMicroseconds(2000);
106    Rotor3.writeMicroseconds(2000);
107    Rotor4.writeMicroseconds(2000);
108
109    delay(5200);
110
111    Rotor1.writeMicroseconds(1000);
112    Rotor2.writeMicroseconds(1000);
113    Rotor3.writeMicroseconds(1000);
114    Rotor4.writeMicroseconds(1000);
115
116    delay(8200);
117
118    void loop() {
119        tick_comms();
120        if (Update1) {
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    }
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200

```

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>
```