D4 Team Report

Team Thames

March 17, 2017

Contents

1 Aims

The aim of this project was to build an aerial cargo transporter, with the idea being that a company like Amazon could use the transporter to send packages to customers very quickly and reliably, with a minimisation on the cost of delivery. To make this possible, a quadcopter design was chosen. This design was selected to overcome one of the biggest challenges of the project maintaining stability when in flight. One of the requirements for the systems was to have telemetry of parameters such as height, pitch etc. This is advantageous, especially if the telemetry is broadcasted live, because the telemetry can be used for diagnostics either in the event of a crash or if the drone is not behaving as expected.

Most drones adopt one of two chassis styles the X-frame and the H-frame. For this project, the X-frame was chosen for its greater stability. The trade-off was that the H-frame is easier to manufacture and build, and is generally stronger. For additional stability, attempts were made to reduce both the weight of the drone, and the distance of the drones centre of mass from the base of the chassis.

The target weight of the drone was 0.5kg, which in combination with an expected thrust of 1.6kg, gives about 1.1kg as a maximum payload. Taking this into account, the recommended maximum payload is 0.5kg, as the drone should be able to lift this weight without putting excessive strain on the motors and speed controllers. Putting this strain on the drive system leads to high temperatures which can permanently affect their efficiency, and can even degrade their mountings.

Another important specification is the flight time of the drone. This depends on the load that drone is carrying, because increasing the load increases the thrust required for flight, which in turn increases the current draw of the motors and speed controllers. A lithium polymer battery was chosen for powering the drone because lipos give a constant voltage for a long time; the only risk is that lipos catch fire if charged incorrectly.

For the control aspect of the design, the telemetry was identified as an area where a stretch-goal could be attempted. In this case, it was decided that a Bluetooth link should be used. This link could potentially allow the user to see live data from the drone whilst in flight. Additionally, it could allow the user to make adjustments to the handling of the drone by tuning PID constants. If it works correctly, the procedure would be as simple as landing the drone, change the numbers and fly it again straight after, with the changed values taking immediate effect.

2 System Design

The block diagram for the project is in **Appendix C**. The controller is central to the design, as it takes inputs from the IMU, Bluetooth serial and remote controlled receiver, and converts this information into a PWM output for the speed controllers. The control is done by an ATmega664p from the Il Matto boards that were developed in first year laboratories. The function of the controller, besides converting its inputs into an output, is to change the interpretation of the signals from the receiver in such a way the drone is easier to fly. The control is based on damping, with the code for the controller allowing for the PID constants to be easily adjusted by the user.

The IMU decoder is also very important to the design, as it takes the Bluetooth, IMU and PPM data from the receiver, and gives an output to the controller using UART communication. The information from the IMU allows the controller to know its pitch, roll etc. so that it can adjust the thrust to the propellers to stabilise the drone. Taking the output of the PPM decoder also allows the IMU decoder to factor in the inputs from the pilot when calculating its output. The decoder also transmits the telemetry data from the IMU to the host using Bluetooth, which provides a data log for the user.

The PPM to Digital conversion is important for the implementation of the transmitter and receiver, as the output of the receiver needs to be decoded for the pilots input to be processed. The receivers output can either be processed as PWM or PPM; the latter was chosen here because it only required one output, although the data can only be processed at a quarter of the rate of the PWM outputs. However, the PWM requires 4 outputs/inputs to work. The PPM decoder, along with the controller and IMU decoder, are central to the design.

To get the drone off the ground, the drive module is needed. This consists of electronic speed controllers, motors, propellers and power distribution board. The speed controllers receive a PWM signal from the controller, which they convert into a three phase output for the motor. The speed controllers are very basic and have no intelligent control aspect to them. The power distribution board provides a suitable continuous voltage and helps to reduce the temperature of various other parts.

Last of the modules is the Bluetooth communications module. This is designed to allow live data logging from the drone, and allows the user to change damping constants and motor thrusts whilst in

use. This tuning data is sent to the IMU decoder using a UART communication, where it is then fed to the controller.

This architecture was developed based on what external components were needed, such as speed controllers and the receiver. Initially, the control section was seen as a black box, where inputs from the receiver and IMU would go in, be processed and outputted via the speed controllers. As the project developed and after researching chips, the control split into the PPM to digital conversion, IMU decoder, and the central controller. The way in which the system is designed means that no other system architectures were considered.

3 Design Evaluation

Overall, the design was relatively simple, with a single stretch goal. This is because it was agreed that it would be better to get something simple that works rather than be too ambitious and fail to get anything working. Even as it is, the difficulty is quite high due to the amount of communication that is present between the various modules. There is, however, scope to attempt other stretch goals; an audio feature would be the easiest to implement, although some sound would be lost due to the propellers.

In this project, it was important to have a good quality of electronic design, as the task is inherently difficult. Breaking the design into modules was very important, because it made it easier to allocate work, but also because it makes it easy to see how all of the modules interact. The design was also based on existing products, which makes debugging easier, because in the event of a module not working it is more likely to mean these products are being used incorrectly.

An important specification when making a product is the ease of use, because there is not much point in making a new product if it is too difficult to use. In the context of aircrafts, the best way to make the system easy to use is to reduce the start up procedure so that the user is able to pick up the drone and fly it straight away. If this is not achievable, the next best thing is to have such a small procedure that the instructions can be put onto the drone itself so that the instruction manual does not have to read for the drone to be used.

For this type of project, there are a lot of existing solutions and products already available. This means that it is difficult to be creative, because there is a good chance that a design already exists, or if it doesnt exist then it probably means it has failed to make a difference. For example, the chassis frame was chosen to be in the shape of an X-frame. This is a common shape for quadcopter frames, along with the H-frame; many other frames have been tried and tested and most failed to keep the drone stable. As such, there is very little on this quadcopter that has never been seen before on other products.

On a similar note to the above, the aesthetics are a low priority as well. It was seen as preferable to get a working drone than to make an aesthetically pleasing drone. However, if there is sufficient time at the end of the project, efforts will be made to make the aircraft easy on the eye. If this were to become an actual product, the aesthetics would be designed when the product works in order to make it more marketable.

It is expected that the drone will not cost much, with a price tag of 150 (not including VAT) giving an estimated 60 profit per unit sold. The cost is kept down by cutting costs on non-critical components such as propellers, whilst using decent quality parts for critical modules such as the motors. The cost of each unit would be 90, but reduction of costs in mass production would lower this value.

Reliability is important for all products and is often enhanced by thorough testing, and acting on the feedback received from the tests. The drone was split into separate, testable modules that could be put together. This meant that each module could tested thoroughly, resulting in a more reliable design.

4 Costing, Marketing and Conformance Testing

If this product were to actually be sold commercially, calculations would need to be made relating to how much was spent on the development of the product. For this, person-hours, conformance testing, and overheads should be considered. The overheads are estimated at 100,000 to pay for facilities for design, manufacturing, and testing; this makes up the majority of the cost. The person-hours are also expensive with an hourly rate of 75, and an estimated 450 hours spent on the design of the prototype, this equates to an additional 33750.

The cost of the prototypes components was 98, but with mass production we expect each unit will cost 70. This means that if each unit is sold for 150 plus VAT, the profit will be 80 per unit, meaning that 1720 number of units will need to be sold to cover the cost of the project.

In order to get an awareness of the product, there will need to be marketing exercises. These primarily consist of branding and early release discounts. A possible early discount would be to sell the first 20 units at half profit, so that an extra 20 units would need to be sold to cover the costs. Additionally, the product would need a website, branding and a logo. The logo could be designed in house for no additional cost, although it is a better idea to go to a professional graphic design agency. The logo alone would cost an estimated 250, with the branding costing somewhere in the region of 1000 and the website design costing another 500 as well (1).

Another way of getting market awareness is to sell the drones to large distributors at a bulk discount, so that can sell the units for a sensible profit. This would reduce profit per unit, but would increase the quantity of sales, which is another way to make profit. If units were sold at 110 plus VAT this would mean that 3440 units would need to be sold to cover the initial outlay. Given that it is unlikely that a distributor would buy many of a new product, as it would be a gamble for them; it is possible that a moderate number of units can be sold to multiple distributors.

For the product to be sold, it needs to pass conformance testing. Conformance testing is a standardised set of tests that determine whether a product is safe for reasonable conditions. In the context of electronic products, this will consist of exposing the casing to substances that are typically found in the house, such as water. They will also be tested with radiation (both exposure to radiation, and having their radiation output measured) and changes of voltage to simulate events like voltage spikes from motors. If all of these are passed, the product can then have a CE sticker put on it, which tells customers that the product is safe to use. This process costs approximately 2000.

Factoring in all of the above, the cost of getting the product off the ground would now be 137,500. With the same price tag of 150 plus VAT, the number of sales required for a profit would be 1720. If 2000 units were sold to a large distributor at a price of 110 plus VAT, the number of units would reduce to 720.

5 Final Product

Overall, the product did not work as well as expected. The drone was able to fly, but with very limited control. The PID control worked to some extent, meaning that each of the motors would adjust its thrust in response to changes of roll and pitch. Unfortunately, the process through which it did this was unable to give a smooth response, which would have resulted in the drone being very difficult to control. As such, our product did not meet the specifications of carrying weight or stable flight. With more time for development, it is reasonable to expect that the product would have operated as a quadcopter; the weight carrying aspect would require a lot more work than stable flight.

Figure 1: An image of the drone at submission with the electronics on the protoboard.

The battery powers the power distribution board, which provides an 11V output to each of the ESCs, which in turn power the motors. It also has a 3.3V regulator that powers everything else. The speed controllers take their input from the PID controller. The PID controller is the main brains of the system, taking inputs from the communications, and the Motion Processor Unit. The output of the MPU is filtered by the Arduino Mini, which sends the data to the PID controller in a UART format. The MPU is a 3 axis gyro that gives information on the roll and pitch of the drone at any point. It also has a magnetometer which allows it to output information relating to the yaw of the drone. The Bluetooth module transmits thrust, yaw, roll and pitch inputs from the pilot, so that the controller can change the output based on these inputs. In addition to this, the controller will attempt to maintain the

drone with all four propellers being level, and any inputs from the user will be slightly counteracted by the controller. There is also a hook on the underside of the chassis, which consists of four ends of string attached to each corner of the chassis. The string are twined together for increased strength. There is a hook on the ends of the string.

If the drone were to be sold commercially, extension tasks would have been attempted to make the product more marketable and to give the customer better value for money. A good example of an extension would be to have live video streaming from the drone when in flight. This would allow the pilot to see the drone if it went out of eyeshot, or behind a behind a building for example. If this video was uploaded to the internet through a live stream, it would allow customers to see the progress of their delivery drone as it transports their order to them. Including a live internet stream would be quite difficult, but having a live video from the drone is a very achievable extension.

Another worthwhile extension would be to make the drone autonomous or at least semi autonomous. Full autonomy would require a lot of work on the control module, whereas semi autonomy would be more achievable. Semi autonomy would mean the drone would take off and/or land itself no control needed from the user. The take off would be easier than landing, because the take off could be programmed to give even thrust for a period of time so that the drone goes straight up. Landing would require the drone to know the area where its landing, which would require more sensors, each attached to the underside so that the landing area could be profiled.

An easier extension would be to have a distinctive lighting pattern. This would be useful for flying in the dark, both for allowing the drone to see and for the drone to be seen. Ultimately the lighting could just be LEDs of different colours forming differing patterns. This would be good if there was a small quantity of drones, whereas trying to uniquely identify thousands of drones would be much more difficult and would probably involve a unique code for each drone.

These extensions would be enough to give the product its own niche in the market, and each one would be achievable with more time and money. Of course, the first priority would be to make the drone work properly; the additional features would then follow on from that.

(1) 2B Designer. Available: http://www.graphicdesigner2b.co.uk/prices/ Last accessed 14th March 2017.

APPENDIX A: Design Completion Form

THAMES

	Component of system/Milestone	Supervisor	Time/Date	Comments (all/part/none working; protoboard/constructed)
X	Understand RC Rx output			
	Get basic motor movement via ESC	QM	12:30 6th	Signal generator → ~~square~~ wave → ESC → motor. pulse
X	Successful signal generator test of PPM Decoder			
X	Successful RC test of PPM Decoder			
✓	Successful serial telemetry test (transmit), in software			
✓	Successful serial telemetry test (receive), in software			
	Successful transmission and receive via Bluetooth, hardware			
	Get raw data out of the IMU	QM	12:30 6th	IMU → Arduino → serial → PC program - data decoded and displayed 200Hz ← on breadboard
	Establish communication (I2C or SPI)	QM	"	"
	Process measurements with DMP or external filter	QM	"	filter running on arduino Mahony filter from library
✓	Controller outputs a ~~PPM~~/PWM signal for ESCs	QM	10th 10:00	14 discrete levels from terminal → 11 Matlab → ESCs & motors.
X	Controller receives converted PPM data			
✓	Controller receives serial telemetry	QM	10th 10:00	see above. terminal → 11 matlab. used.
	Test power distribution board	QM	10.37 09.03.17	
✓	Test power distribution board with motors	QM	10th 10:00	Motors running from ESCs from battery.
	Flight/ ~~Stable Flight~~	SRG	14:10 13/3	FLIGHT OFF TABLE DEMONSTRATED BY VIDEO. CONTROL NEEDS SOME WORK
	Target flight time achieved (1/2/3 minutes)			
	Hooked on to and carried empty lunchbox			
	Lift target weight			

Milestones finalised by supervisor: S. GUNN ..... Signed ..... Date 13/3/17

Prototype hardware handed over to: ..... Signed ..... Date

Other items returned to Lab support hatch and checked by: ..... Signed ..... Date

APPENDIX B: Project Completion Form

Appendix E: Project Completion Form

Cost Estimates

Please give detailed calculations and estimates of the overall cost of your actual design below. Take care to include person-hour estimates for your software, board production and debugging, as well as your components and consumables. You should also estimate the production cost of your final unit (you may assume a large quantity are to be produced), the market price and determine how many need to be sold to be profitable. Account for any differences between the actual values and the values given in your original project proposal form.

Total cost for prototype £98

Estimated cost for manufacture £70 with mass discount of ~30%.

Person-hours 450 x 75 £33750

Total cost £137,500

Market value £150 + VAT → £180 £80 profit per unit → 1720 units sold to make a profit

Differences: More person-hours

More expensive propellers, motors, power distribution board

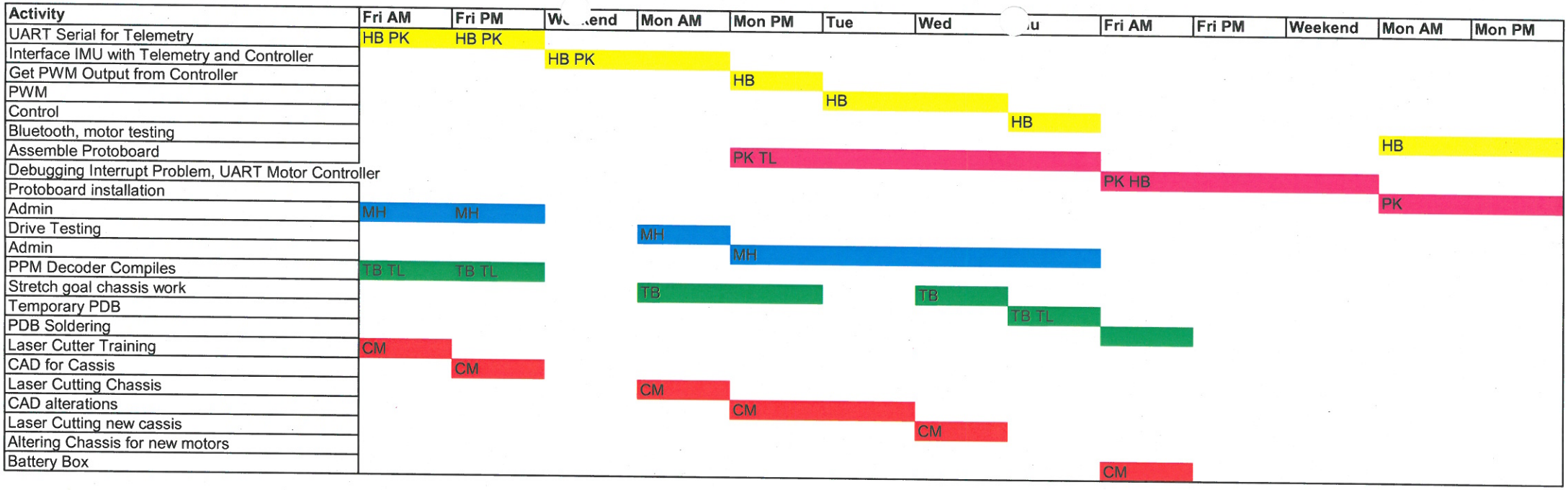
Design Changes

Briefly summarise any design changes your team had to make to the original design proposal, in order to get your system to work. Do not go into vast detail, as it is anticipated that this will be done by the individuals responsible for these components of the design in the formal report.

- Removal of PPM-to-Digital Decoder due to Spectrum not arming, replaced with bluetooth module.
- Moved IMU interface to an Arduino Mini to remove compute stress from the main controller.
- Added a MSP microcontroller to fix issue with data integrity because of a high interrupt ~~rate~~ rate on the controller. (Motor control)*
- Holes from chassis removed due to restrictions on the laser cutter hardware, used tape for mounting instead.

* MSP used to control motors via UART, also a better choice due to 6 COM outputs from 2 * 16-bit timers vs 4 COM on 2 8-bit timers on the mbed.

Actual


Activity
Fri AM
Fri PM
Weekend
Mon AM
Mon PM
Tue
Wed
Fri AM
Fri PM
Weekend
Mon AM
Mon PM
UART Serial for Telemetry
Interface IMU with Telemetry and Controller
Get PWM Output from Controller
PWM
Control
Bluetooth, motor testing
Assemble Protoboard
Debugging Interrupt Problem, UART Motor Controller
Protoboard installation
Admin
Drive Testing
Admin
PPM Decoder Compiles
Stretch goal chassis work
Temporary PDB
PDB Soldering
Laser Cutter Training
CAD for Cassis
Laser Cutting Chassis
CAD alterations
Laser Cutting new cassis
Altering Chassis for new motors
Battery Box
HB PK
HB PK
HB PK
HB
HB
HB
HB
PK TL
PK HB
PK
MH
MH
MH
MH
TB TL
TB TL
TB
TB
TB TL
CM
CM
CM
CM
CM
CM

Discrepancy in Project Activities

Comment on any major differences between the planned and actual project activities.

Transmitter and receiver did not arrive

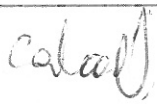
Unexpected activities had to be completed

Broken arm on chassis

Kept to plan reasonably well

Assessment of Effort

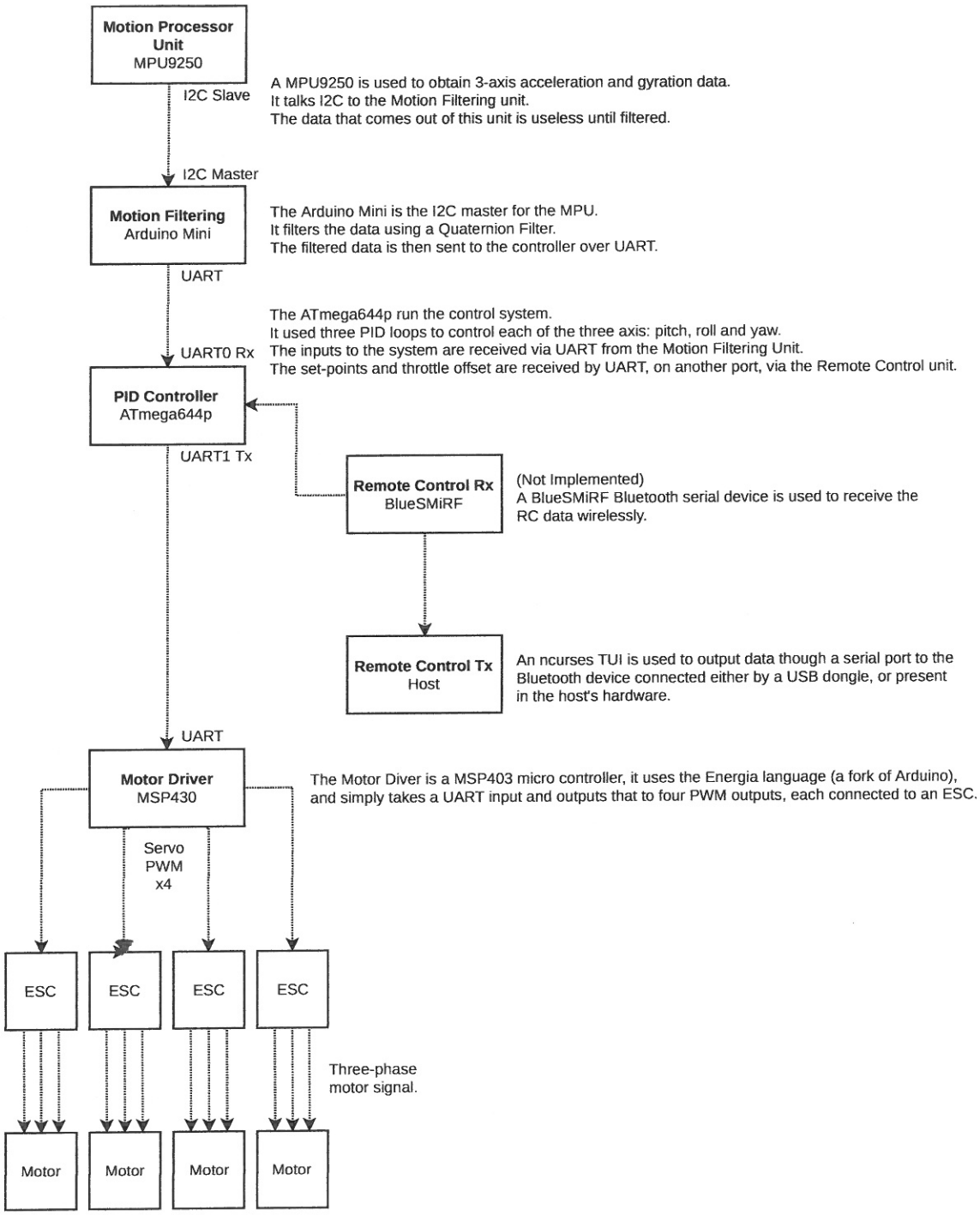
The table below will be used as an indication how team marks should be allocated across the team.

Name	Signature	% of effort
Matt Hunter		15
Harry Beadle		20
Petros Kargologiannis		20
Trong Luong		15
Tim Bills		15
Callum Marshall		15

If the breakdown is not equal please provide a short explanation below:

Harry and Petros were focused on software

APPENDIX C: Circuit Diagrams


Motion Processor Unit
MPU9250
I2C Slave
A MPU9250 is used to obtain 3-axis acceleration and gyration data.
It talks I2C to the Motion Filtering unit.
The data that comes out of this unit is useless until filtered.
I2C Master
Motion Filtering
Arduino Mini
The Arduino Mini is the I2C master for the MPU.
It filters the data using a Quaternion Filter.
The filtered data is then sent to the controller over UART.
UART
The ATmega644p run the control system.
It used three PID loops to control each of the three axis: pitch, roll and yaw.
The inputs to the system are received via UART from the Motion Filtering Unit.
The set-points and throttle offset are received by UART, on another port, via the Remote Control unit.
UART0 Rx
PID Controller
ATmega644p
UART1 Tx
Remote Control Rx
BlueSMiRF
(Not Implemented)
A BlueSMiRF Bluetooth serial device is used to receive the RC data wirelessly.
Remote Control Tx
Host
An ncurses TUI is used to output data though a serial port to the Bluetooth device connected either by a USB dongle, or present in the host's hardware.
UART
Motor Driver
MSP430
The Motor Diver is a MSP403 micro controller, it uses the Energia language (a fork of Arduino), and simply takes a UART input and outputs that to four PWM outputs, each connected to an ESC.
Servo
PWM
x4
ESC
ESC
ESC
ESC
Three-phase
motor signal.
Motor
Motor
Motor
Motor

APPENDIX D: Software Listings

D4 Code Listings

Team Thames

March 17, 2017

Contents

1 ThamesControl

1.1 Communications

1.1.1 Header

```
1  /*
2
3  Harry Beadle
4  D4 Thames
5  Communications (comms.h)
6
7  Headerfile for UART ISRs and initalisation.
8
9  */
10
11 #ifndef _COMMS_H_
12         #define _COMMS_H_
13
14         #include <avr/io.h>
15         #include <avr/interrupt.h>
16
17         #include "rc-symbols.h"
18         #include "mpu-symbols.h"
19         #include "drone.h"
20
21         #define BAUD 9600
22
23         // MPU Decoder Communications
24         uint8_t temp_byte, control_byte, low_byte, high_byte, bad_control;
25         ISR(USART0_RX_vect);
26
27         // Remote Controller Communications
28         enum {control, high, low} state, nstate;
29         uint8_t temp_byte_rc, control_code, data_byte;
30         ISR(USART1_RX_vect);
31
32         #include "../comms.c"
33
34 #endif
```

1.1.2 Source

```
1  /*
2
3  Harry Beadle
4  D4 Thames
5  Communications (comms.c)
6
7  Input                    Perf    Pin
8  ------------------------ ------- -------
9  UART from MPU Decoder    UART0   PD0
10 UART from Controller     UART1   PD2
11 UART to Motor Controller UART1   PD3
12
13 */
14
15 #include "inc/comms.h"
16
17 void init_comms(void)
18 {
19         // Initialise UART0 and UART1
20         // Set Badu Rate.
```

```
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>
```

```
#include <util/delay.h>

#ifndef F_CPU
#define F_CPU 12000000UL
#endif
#define BDRATE_BAUD 9600

#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 1
#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


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 servo values AKA central positions
uint16_t stat[NUM_PULSE_PER_FRAME];

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;
}
void init_stdio2uart0(void)
{
        // configure UART0 baud rate, one start bit, 8-bit, no parity and one stop bit
        UBRR0H = (F_CPU/(BDRATE_BAUD*16L)-1) >> 8;
        UBRR0L = (F_CPU/(BDRATE_BAUD*16L)-1);
        UCSR0B = _BV(RXEN0)|_BV(TXEN0);
        UCSR0C = _BV(UCSZ00)|_BV(UCSZ01);

}
void binaryoutput(const uint16_t Outputnumber)
{
        DDRA = 0xFF;
        uint8_t remainer = 0;
        uint16_t tmp = Outputnumber;
        while (tmp > 1) {
                remainer = tmp % 10;
                tmp = tmp/10 - remainer;
                if (remainer == 1) {
                        PORTA |= _BV(PINA1);
                        _delay_ms(1);
                } else {
                        PORTA |= ~_BV(PINA1);
                        _delay_ms(1);
```



```
                }
        }
}
void Transfer_numb(const uint16_t Controldata)
{
        // output the 5 bits data through UDR0
        while (!(UCSR0A & _BV(UDRE0)));
        UDR0 = Controldata;
}
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;

        // Initiate the servo values with neutral values
        for (i=0;i< NUM_PULSE_PER_FRAME;i++) {
                inPW[i] = NEUTRAL_PULSE_LEN;
        }
        // Power and noise reduction
        PRR = _BV(PRTIM2)|_BV(PRTIM0)|_BV(PRSPI)|_BV(PRUSART0)|_BV(PRADC);

        DDRD = 0x00; //PD6 as input (ICP1)

        ICR1 = 0;

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


        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;
                                validSync = 1;
                                inPulseCount = 0;
                        } else if (validSync && ValidPulseLen(PulseLen) && (inPulseCount <
                                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
                for (i=0; i<NUM_PULSE_PER_FRAME;i++) {
```

```
                        Transfer_numb(digitalData[i]);
                } // sending 5-bit binary through UDR0
                for (i=0; i<NUM_PULSE_PER_FRAME;i++) {
                        binaryoutput(digitalData[i]);
                }

        }
}
```

5 Serial Telemetry Proccessor

```
/*

Harry Beadle
D4 Thames
Telemetry (telemetry.c)

UART0 Tx Bluetooth Host
UART0 Rx Bluetooth Host

UART1 Rx PPM Decoder (RC Data)

*/

#include "inc/telemetry.h"

ISR(USART0_RX_vect)
{
        // UART0 Rx from Bluetooth Host
        // Add control tuning data to control buffer.
        buffer8_add(&control_buffer, UDR0);
}

ISR(USART1_RX_vect)
{
        // UART1 RX from PPM Decoder
        // Add remote control data to output buffer.
        buffer8_add(&output_buffer, UDR1);
}

ISR(SPI_STC_vect)
{
        // Pull PB4 Low
        PORTB &= ~_BV(PB4);
        // Put Rx'd byte in rx buffer
        uint8_t spi_data = SPDR;
        if (spi_data) {
                buffer8_add(&spi_rx_buffer, SPDR);
        }
}

int main(void)
{
        // Initialise SPI
        DDRB = _BV(PB4) | _BV(PB5) | _BV(PB7);
        SPCR = _BV(SPE) | _BV(SPIE) | _BV(MSTR) | _BV(SPR0);

        // Initalise UART0 and UART1
        // Set Baud Rate.
        UBRR0H = (F_CPU/(BAUD*16L)-1) >> 8;
        UBRR1H = (F_CPU/(BAUD*16L)-1) >> 8;
        UBRR0L = (F_CPU/( BAUD*16L)-1);
        UBRR1L = (F_CPU/( BAUD*16L)-1);
        // Enable Tx, Rx, Tx Interupt, Rx Interupt.
        UCSR0B = _BV(RXEN0) | _BV(RXCIE0) | _BV(TXEN0) | _BV(TXCIE0);
        UCSR1B = _BV(RXEN1) | _BV(RXCIE1) | _BV(TXEN1) | _BV(TXCIE1);
        // 1 Start, 8 Data and 1 Stop bit.
        UCSR0C = _BV(UCSZ00) | _BV(UCSZ01);
        UCSR1C = _BV(UCSZ10) | _BV(UCSZ11);
        // Gloally Enable interrupts.
        sei();

        // Initalise the SPI device
        spi_write(WRITE & MPUREG_USER_CTRL, BIT_I2C_IF_DIS);
        spi_write(WRITE & MPUREG_INT_PIN_CFG, 0x22);
        spi_write(WRITE & MPUREG_PWR_MGMT_1, BIT_H_RESET);
        spi_write(WRITE & MPUREG_PWR_MGMT_1, 0x01);
        spi_write(READ & MPUREG_WHOAMI, 0x00);

        // Loop Forever
        while (1) {
                // Wait for the output buffer to be ready.
                if (buffer8_rdy(&output_buffer)) {
                        // Wait wile the UART is busy.
                        while (!(UCSR0A & _BV(UDRE0)));
                        // Output the contents of the buffer.
                        UDR0 = buffer8_pop(&output_buffer);
                }
                if (buffer8_rdy(&spi_rx_buffer)) {
                        while (!(UCSR0A & _BV(UDRE0)));
                        UDR0 = buffer8_pop(&spi_rx_buffer);
                }
                if (!(SPSR & _BV(SPIF)) && buffer8_rdy(&spi_tx_buffer)) {
                        PORTB |= _BV(PB4);
                        SPDR = buffer8_pop(&spi_tx_buffer);
                }
        }
}
```

6 MPU Decoder

```
/* MPU9250 Basic Example Code
 by: Kris Winer
 date: April 1, 2014
 license: Beerware - Use this code however you'd like. If you
 find it useful you can buy me a beer some time.
 Modified by Brent Wilkins July 19, 2016

 Demonstrate basic MPU-9250 functionality including parameterizing the register
 addresses, initializing the sensor, getting properly scaled accelerometer,
 gyroscope, and magnetometer data out. Addition of 9 DoF sensor fusion using open source
 Madgwick and Mahony filter algorithms. Sketch runs on the 3.3 V 8 MHz Pro Mini
 and the Teensy 3.1.

 SDA and SCL should have external pull-up resistors (to 3.3V).
 10k resistors are on the EMSENSR-9250 breakout board.

 Hardware setup:
 MPU9250 Breakout ------------ Arduino
 VDD ------------------------- 3.3V
 VDDI ------------------------ 3.3V
 SDA ------------------------- A4
 SCL ------------------------- A5
 GND ------------------------- GND
```

```
 */

#include "quaternionFilters.h"
#include "MPU9250.h"

#define AHRS true         // Set to false for basic data read
#define SerialDebug true  // Set to true to get Serial output for debugging

// Pin definitions
int intPin = 12;  // These can be changed, 2 and 3 are the Arduinos ext int pins
int myLed  = 13;  // Set up pin 13 led for toggling

MPU9250 myIMU;

void setup()
{
  Wire.begin();
  Serial.begin(57600);

  // Set up the interrupt pin, its set as active high, push-pull
  pinMode(intPin, INPUT);
  digitalWrite(intPin, LOW);
  pinMode(myLed, OUTPUT);
  digitalWrite(myLed, HIGH);


  // Read the WHO_AM_I register, this is a good test of communication
  byte c = myIMU.readByte(MPU9250_ADDRESS, WHO_AM_I_MPU9250);
  if (c == 0x73)
  {
    // Start by performing self test
    myIMU.MPU9250SelfTest(myIMU.SelfTest);

    // Calibrate gyro and accelerometers, load biases in bias registers
    myIMU.calibrateMPU9250(myIMU.gyroBias, myIMU.accelBias);

    myIMU.initMPU9250();
    // Initialize device for active mode read of acclerometer, gyroscope, and
    // temperature

    // Read the WHO_AM_I register of the magnetometer, this is a good test of
    // communication
    byte d = myIMU.readByte(AK8963_ADDRESS, WHO_AM_I_AK8963);

    // Get magnetometer calibration from AK8963 ROM
    myIMU.initAK8963(myIMU.magCalibration);

    // Initialize device for active mode read of magnetometer
    Serial.println("AK8963 initialized for active data mode....");


  }
  if (c == 0x73) //Check if communication is established
  else
  {
    while(1) ; // Loop forever if communication doesn't happen
  }
}

uint16_t pitch_send, roll_send, yaw_send;

void loop()
```



```
{
  // If intPin goes high, all data registers have new data
  // On interrupt, check if data ready interrupt
  if (myIMU.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
  {
    myIMU.readAccelData(myIMU.accelCount);  // Read the x/y/z adc values
    myIMU.getAres();

    // Now we'll calculate the accleration value into actual g's
    // This depends on scale being set
    myIMU.ax = (float)myIMU.accelCount[0]*myIMU.aRes; // - accelBias[0];
    myIMU.ay = (float)myIMU.accelCount[1]*myIMU.aRes; // - accelBias[1];
    myIMU.az = (float)myIMU.accelCount[2]*myIMU.aRes; // - accelBias[2];

    myIMU.readGyroData(myIMU.gyroCount);  // Read the x/y/z adc values
    myIMU.getGres();

    // Calculate the gyro value into actual degrees per second
    // This depends on scale being set
    myIMU.gx = (float)myIMU.gyroCount[0]*myIMU.gRes;
    myIMU.gy = (float)myIMU.gyroCount[1]*myIMU.gRes;
    myIMU.gz = (float)myIMU.gyroCount[2]*myIMU.gRes;

    myIMU.readMagData(myIMU.magCount);  // Read the x/y/z adc values
    myIMU.getMres();
    // User environmental x-axis correction in milliGauss, should be
    // automatically calculated
    myIMU.magbias[0] = +470.;
    // User environmental x-axis correction in milliGauss TODO axis??
    myIMU.magbias[1] = +120.;
    // User environmental x-axis correction in milliGauss
    myIMU.magbias[2] = +125.;

    // Calculate the magnetometer values in milliGauss
    // Include factory calibration per data sheet and user environmental
    // corrections
    // Get actual magnetometer value, this depends on scale being set
    myIMU.mx = (float)myIMU.magCount[0]*myIMU.mRes*myIMU.magCalibration[0] -
               myIMU.magbias[0];
    myIMU.my = (float)myIMU.magCount[1]*myIMU.mRes*myIMU.magCalibration[1] -
               myIMU.magbias[1];
    myIMU.mz = (float)myIMU.magCount[2]*myIMU.mRes*myIMU.magCalibration[2] -
               myIMU.magbias[2];
  } // if (readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)

  // Must be called before updating quaternions!
  myIMU.updateTime();

  // Sensors x (y)-axis of the accelerometer is aligned with the y (x)-axis of
  // the magnetometer; the magnetometer z-axis (+ down) is opposite to z-axis
  // (+ up) of accelerometer and gyro! We have to make some allowance for this
  // orientationmismatch in feeding the output to the quaternion filter. For the
  // MPU-9250, we have chosen a magnetic rotation that keeps the sensor forward
  // along the x-axis just like in the LSM9DS0 sensor. This rotation can be
  // modified to allow any convenient orientation convention. This is ok by
  // aircraft orientation standards! Pass gyro rate as rad/s
//  MadgwickQuaternionUpdate(ax, ay, az, gx*PI/180.0f, gy*PI/180.0f, gz*PI/180.0f,
my,  mx, mz);
  MahonyQuaternionUpdate(myIMU.ax, myIMU.ay, myIMU.az, myIMU.gx*DEG_TO_RAD,
                         myIMU.gy*DEG_TO_RAD, myIMU.gz*DEG_TO_RAD, myIMU.my,
                         myIMU.mx, myIMU.mz, myIMU.deltat);

```

```
147         myIMU.yaw   = atan2(2.0f * (*(getQ()+1) * *(getQ()+2) + *getQ() *
148                       *(getQ()+3)), *getQ() * *getQ() + *(getQ()+1) * *(getQ()+1)
149                       - *(getQ()+2) * *(getQ()+2) - *(getQ()+3) * *(getQ()+3));
150         myIMU.pitch = -asin(2.0f * (*(getQ()+1) * *(getQ()+3) - *getQ() *
151                       *(getQ()+2)));
152         myIMU.roll  = atan2(2.0f * (*getQ() * *(getQ()+1) + *(getQ()+2) *
153                       *(getQ()+3)), *getQ() * *getQ() - *(getQ()+1) * *(getQ()+1)
154                       - *(getQ()+2) * *(getQ()+2) + *(getQ()+3) * *(getQ()+3));
155         myIMU.pitch *= RAD_TO_DEG;
156         myIMU.yaw   *= RAD_TO_DEG;
157
158         myIMU.yaw   -= 8.5;
159         myIMU.roll  *= RAD_TO_DEG;
160
161         pitch_send = (uint16_t) map(myIMU.pitch, -180, 180, 0, 65536);
162         roll_send = (uint16_t) map(myIMU.roll, -180, 180, 0, 65536);
163         yaw_send = (uint16_t) map(myIMU.yaw, -180, 180, 0, 65536);
164
165         #define PITCH_S 0xF1
166         #define ROLL_S  0xF2
167         #define YAW_S   0xF3
168
169         Serial.write(PITCH_S);
170         Serial.write((char) pitch_send >> 8);
171         Serial.write((char) pitch_send);
172         Serial.write(ROLL_S);
173         Serial.write((char) roll_send >> 8);
174         Serial.write((char) roll_send);
175         Serial.write(YAW_S);
176         Serial.write((char) yaw_send >> 8);
177         Serial.write((char) yaw_send);
178
179         myIMU.count = millis();
180         myIMU.sumCount = 0;
181         myIMU.sum = 0;
182 }
```

APPENDIX E: Project Meeting Agendas & Minutes

D4 First meeting 23/2:

https://github.com/orgs/d4thames/projects/1

D4 Second meeting 24/2:

- How the RC talks to to mcu
- UART DUPLEXING
- Materials balsarwood (plexi is brittle)
- aliminum brackets
- Motor Mounting
- Landing Gear
- Max Size 80cm diameter
- more than 6-axis sensor
- power

"Quadcopter, also known as quadrotor, is a helicopter with four rotors.

The rotors are directed upwards and they are placed in a square formation with equal distance from the center of mass of the quadcopter."

hover thrust

where to put the ranger??? needs to work with and without load

---

27/2 Lab Session

-Sensors: Grove - IMU 9DOF v2.0

price: 10.99

accelerometer/gyro/mag/processing

I2C

http://ow.ly/hYvC309ogP0

Sharp GP2Y0A21YK0F Reflective Sensor

price: 7.10

analogue

http://ow.ly/sea8309ogMU

Total: 18.09

Design Clinic

- 1 person define interfaces between different modules
- memory
- instrumentation of the subsystems so that they can be exercised on their own
- number of processing cycles needed to create stability
- PWM to SPR
- digital receiver
- maybe preprocess sensor data

To do:

- figure out the size of data coming out of IMU SPI (done)
- syslog ppm to digital decoder
- check if there is internal clock on IMU (done)
- bill of Materials (done)

To do:

- Project proposal form
- Cost estimates (handbook p.25)
- Prototyping (mockups)

$$$ Limit Pitch Roll Yaw range to a number of degrees

Is there a way to calculate the load?

Check if atmga 32 and 644 have the same pinouts

Meeting Thur 2/3

- Friday
    - •Test PPM-Digital
    - •Test Control
- Tuesday

- Investigate R/C System !!!ME!!!

Geoff Meeting

- MPU- Filering

Power Circuit