# ELEC2205 D4 – Systems Design Exercise

Team Ganges
Ben Rowlinson, Lawrence Gray, Joel Trickey,
Joseph Hindmarsh and Mohammed Ibrahim
17 March 2017

## Contents

**1 Challenge Solution Statement** **2**
1.1 Challenges . . . 2
1.2 Specification . . . 3

**2 System Design** **4**
2.1 High Level Block Diagram . . . 4
2.2 UAV . . . 4
2.3 Communications . . . 4
2.4 Ground Control . . . 5

**3 Design Evaluation** **5**
3.1 Difficulty of Specification Attempted . . . 5
3.2 Quality of Electronic Design . . . 5
3.3 Ease of Use . . . 6
3.4 Creativity and Innovation . . . 6
3.5 Aesthetics . . . 6
3.6 Cost . . . 6
3.7 Reliability . . . 6

**4 Costing, Marketing and Conformance Marking** **6**
4.1 Costing . . . 6
4.2 Manufacturing Costs . . . 7
4.3 Person-Hours . . . 7
4.4 Conformance Marking and Manufacturing Set-Up . . . 8
4.5 Expected Profits . . . 8

**5 Final Product** **9**
5.1 Specification vs. Achievements . . . 9
5.2 Working of the Final Product . . . 11
5.3 Extensions . . . 11

**A Design Completion Form** **12**

**B Project Completion Form** **13**

**C Circuit Diagrams** **15**

**D Software Listings** **17**
D.1 PID_values_test_2.c . . . 17
D.2 PID_values_test_3.c . . . 21
D.3 test2_basestation_comms.c . . . 26
D.4 test3_basestation_comms.c . . . 34
D.5 test5_basestation_comms.c . . . 37
D.6 Uplink/drone_comms.c . . . 47
D.7 basestation_comms.c . . . 49
D.8 encode_test.c . . . 51
D.9 comms.h . . . 56
D.10 Combined_interface_comms/drone_comms.c . . . 57
D.11 test1_drone_comms.c . . . 60
D.12 test2_drone_comms.c . . . 62
D.13 test3_drone_comms.c . . . 65
D.14 drone_serial_comms.c . . . 69
D.15 ESCPID/AccelGyro/AccelGyro.ino . . . 77
D.16 ESCPID/AccelGyro/Definitions.h . . . 88
D.17 ESCPID/AccelGyro/PWM.cpp . . . 89
D.18 ESCPID/AccelGyro_TEST/utils.cpp . . . 91
D.19 ESCPID_TEST/AccelGyro/AccelGyro.ino . . . 92
D.20 ServoPID/AccelGyro/AccelGyro.ino . . . 101
D.21 leonardoPWM.ino . . . 109
D.22 leonardoPWM_input.ino . . . 110
D.23 i2c-lib-test/main.c . . . 118
D.24 onboard_serial_test_ard_3.ino . . . 119
D.25 onboard_serial_test_final.c . . . 121
D.26 SD_card_test_main.cpp . . . 123
D.27 apps.h . . . 125

**E Project Meeting Agendas and Minutes** **130**

**F Images** **134**

# 1 Challenge Solution Statement

## 1.1 Challenges

Same-day parcel delivery is a difficult proposition for many businesses, with it currently only being available in selected cities. This is due to needing a high enough concentration of orders to make sending couriers worthwhile for the company, which cannot be achieved reliably in less densely populated areas. To improve efficiency of small parcel deliveries, Team Ganges proudly presents its Project Itchen prototype – a UAV designed from the ground up for cargo transport. By opening the skies to Itchen for its deliveries, same-day delivery will no longer be restricted to cities and become a reality for customers across the country.// Many commercially available UAVs exist on the market already, however the key thing holding them back from adoption in the deliveries market is the question of cargo. The most common type of payload-carrying drone is a camera drone, which has the payload securely affixed to the drone chassis. This is excellent during flight, but is less optimal for package delivery, as the recipient would have to approach the live UAV to remove the package. To solve this challenge, Team Ganges will provide a mechanism to drop-off the package once it has reached its destination and is at a low enough altitude. This will reduce the turn-around time of the drone and allow it to drop-off the package when the recipient is not present.// The solution concept also addresses the problem of interference with other transmitters, through avoiding the use of a commercial transmitter in favour of a proprietary solution. This allows for the communications between the UAV and the Base Station to be encrypted, and the UAV identify true instructions from the base station over interference from other users and devices.//

## 1.2 Specification

- The UAV should use 4 2205 2300kV motors
- The motors must use an arming procedure
- The propellers shall be 5" in diameter with a pitch of 3"/rev
- Motor speeds shall be controlled by 1-2ms PWM ESCs (Electronic Speed Controllers)
- The UAV shall be powered by a single 3 Cell LiPo battery
- The battery capacity should be more than 1500mAh
- The Communications shall be in the ISM radio frequency band
- Telemetry data could be transmitted back to the base station
- Telemetry data could be stored locally on an SD Card
- Telemetry Transmission rate should be 1 packet/second
- The IMU should produce new data at least 100 times per second
- The UAV should be able to drop off cargo whilst in flight
- The UAV could be able to pick up cargo whilst in flight
- The communications could be encrypted
- Packet loss from the communications should be less than 1 in 100
- Communications speed should be at least 400 packets/second (One 10-bit word per packet)
- The PID gain constants could be updated over the communications uplink
- The UAV should be able to detect altitudes of between 15cm to 100cm
- The control loop should update at a rate of at least 100 times/second to match IMU
- The UAV should warn the user of low battery status
- The chassis should have a centre of mass within 5cm of the geometric centre
- UAV should recover from a 15 degree inclination

# 2 System Design

## 2.1 High Level Block Diagram

Appendix C gives a high-level block diagram of the final design of the system.

## 2.2 UAV

On-board the drone, there are two micro-controllers - an Arduino Leonardo and an Il Matto. The Leonardo handles the PID control to ensure stable flight according to the user's control inputs. It achieves this by receiving angle data from a IMU module over an I2C interface and controller inputs from the Il Matto using UART. After processing this data, it controls the flight of the UAV through four PWM channels to four ESCs, which in turn control four 2205 brushless motors using 3-phase AC power.
The Il Matto is responsible for receiving the user's inputs from the transceiver module and passing on potentiometer data to the Leonardo when requested. It also controls a servo using PWM to raise and lower the cargo hook as per the data received from switches on the ground. It also sends telemetry received from an IR sensor and a battery voltage circuit back to the ground through the transceivers. SD card is not present on the final design.
This system is powered by a 1550mAh 11.1V LiPo battery via a power distribution circuit that ensures the correct voltage is supplied to each component on-board the UAV.
For the chassis, a clear acrylic I-frame was laser-cut and glued together with Super Glue.
Using two micro-controllers to handle control and communications separately, allows the control to remain accurate by reducing computational strain, and a faster data rate from the ground to be achieved. The PWM outputs to the ESCs only use a very small range of timer values. This requires the timers to have a high precision to ensure fine control of the motors. Initially, an Il Matto was considered for the control but as this only supplies one 16-bit timer and two 8-bit timers, it was decided that this would not supply the required precision. The Leonardo on the other hand has two 16-bit timers with 4 outputs which allow sufficient control of its PWM outputs.
For communications, the Il Matto was used as we have experience of programming it, and it has the required hardware to interface with the various modules on the drone.

## 2.3 Communications

The ground control and the drone communicate with each other over a 433MHz FSK scheme using RFM12B-S2 transceiver modules. These modules are both controlled by Il Mattos using the SPI protocol and a micro-controller library. Potentiometer and switch data is sent to the drone and telemetry is sent back to the base.
The Spektrum DX6e remote control transmitter and receiver system was rejected at the design stage as it would require taking four PWM outputs from the receiver into the micro-controller which proved difficult in early testing.

### 2.4 Ground Control

On the ground, an Il Matto uses its built-in ADC to take four different analogue voltages from two X-Y potentiometers for user-controlled thrust, yaw, pitch, and roll. It sends this data to a transceiver for transmission to the drone. Switches taken as inputs to the Il-Matto, allow the user to control the cargo hook, toggle between PID and flight mode, and activate a kill switch for safety. A UART interface with a PC allows telemetry to be read and logged, and new K values for the PID controller to be input.
As an Il Matto was being used with the transceiver on the drone, the use of an Il Matto on the ground made software easier to develop.

## 3 Design Evaluation

### 3.1 Difficulty of Specification Attempted

At the start of this project we knew we wanted to choose a design which was challenging, but also achievable. We knew early on that we didn't want to accept the pre-built controller and instead build our own proprietary one. This helped us to achieve practical two-way communications between ground and air. We wanted to devolve as much of the responsibility for control from the pilot by using absolute angle measurements to provide feedback to the UAV. This poses a more difficult control problem and although we have a demonstrably working control system, we were not able to achieve full, independent, stable flight. Designing and building our own chassis out of acrylic sheets allowed us to come in on budget, but meant that the frame was extremely brittle and heavy, meaning testing the drone safely and non-destructively became a challenge. Our successful extra features; servo-actuated cargo acquisition, IR sensor telemetry, real-time PID tuning, communications encryption, battery voltage reading were only integrated once the more critical modules had been completed.

### 3.2 Quality of Electronic Design

We wanted separate micro-controllers for the on-board communications and control modules. This presented the considerable problem of synchronising communications between these two sub-systems. Utilising the on-chip DMP on the IMU allowed us to acquire pre-processed attitude data which massively reduced the computing strain on our micro-controller. This gives us parallel processing of the past and present IMU data thus giving us a finer resolution for control. This introduces the problem of synchronising these separate processes. Use of PWM controlled ESCs meant we needed 4 16-bit timer outputs from the control sub-system to finely adjust our motors. This meant upgrading to a more powerful micro-controller. The Power Distribution Board (PDB) meant we could power all the on-board components from the same supply and meant we could reliably power up. This did increase the weight of the drone.

### 3.3 Ease of Use

Our start-up sequence includes a number of safety features; IMU settling time, motor arming sequence, optional ESC calibration. The pilot interface is a standard UAV set-up, except that the throttle is centred at zero and not free-floating. The controller is ergonomic and can be run off batteries. The current lack of LEDs indicating the front makes piloting difficult at large distances.

### 3.4 Creativity and Innovation

We designed our own servo-actuated cargo hook to enable quick and easy pick-up and drop-off of the cargo. This adds weight and complexity to the electronics, but was easy to bolt onto the critical path design. We were able to read the altitude and battery level in real time using a serial terminal. This could be implemented using the Il Matto TFT screen so the Laptop isn't required. The ability to repair the chassis quickly and easily allowed for rapid testing, but this unbalanced the weight distribution of the drone and led to inconsistent test flights.

### 3.5 Aesthetics

Transparent acrylic chassis is less of an eye-sore, but equally makes piloting at a long distance difficult. Etching into the plastic to add our company name makes it easily identifiable as a brand. Our controller also matches the aesthetic style of the drone. The switches are placed at convenient locations for rapid killing of motors if necessary. The coat-hanger wire propeller guards are minimalistic but robust enough to deal with low velocity impacts. As this is still a prototype there are flying wires used but we used ribbon cable wherever possible to minimise crossed wires.

### 3.6 Cost

See section 4.1.

### 3.7 Reliability

The UAV powers up correctly and consistently on connection to the LiPo. The Communications system was heavily tested to ensure we could conduct test flights safely. We achieved tethered single axis stability through a number of tests. The current cargo acquisition hook doesn't fully extend and gets stuck in a particular position. We cannot reliably achieve stable flight, but this only requires tuning of the PID gains. Throughout our fully integrated testing, the electronics systems remained robust and very rarely experienced a failure. The chassis performed very well with regard to protecting the electronics from impacts.

## 4 Costing, Marketing and Conformance Marking

### 4.1 Costing

In the project proposal form, estimates were made towards the costings. Now that the project is done, more realistic estimates can be made. This cost differs from this original as more

components were included in the project than originally planned and some components broke in the testing process so the prices in Table 1 includes replacements for those parts.

| Component | Amount | Single Cost/£ | Bulk Cost/£ |
|---|---|---|---|
| 20A ESC | 4 | 23.95 | 23.95 |
| Clockwise 2204 2300KV motors | 2 | 11.78 | 11.78 |
| Counter-Clockwise 2204 2300KV motors | 2 | 11.78 | 11.78 |
| Thumb Joystick | 2 | 10.39 | 3.98 |
| Propeller Pack | 1 | 1.25 | 1.00 |
| Female XT60 battery connector | 1 | 1.00 | 1.00 |
| ESC Connector Pack | 1 | 3.50 | 1.99 |
| 1550MAh LiPo Battery | 1 | 14.99 | 12.86 |
| Arduino Leonardo | 1 | 18.50 | 5.64 |
| RFM12B-S2 transceiver modules | 2 | 12.00 | 3.78 |
| 5V Regulator | 1 | 1.63 | 0.79 |
| LiPo Charger | 1 | 14.99 | 14.99 |
| Atmel At-Mega 644p | 2 | 12.62 | 7.24 |
| Micro Servo | 1 | 7.70 | 1.93 |
| MPU-6050 Gyroscope Chip | 1 | 1.50 | 1.50 |
| Miscellaneous | 1 | 8.00 | 6.00 |
| A2 5mm acrylic sheet | 2 | 21.10 | 8.49 |
| A4 3mm acrylic sheet | 1 | 2.64 | 1.35 |
| Labour (£20/hour) | 2 | 0 | 40 |
| **Total** | | 179.32 | 160.05 |

Table 1: Table showing total costings for parts for prototyping and manufacturing

### 4.2 Manufacturing Costs

The bulk cost includes the labour costing for manufacturing the quad-copter. This was estimated to be 2 hours. The two hours broken down are as follows; A skilled worker will first laser cut the frame and glue it together which should take 30 minutes. They then must program all of the micro-controllers and calibrate the ESCs, this should take another 10 minutes. The circuits should take 40 minutes to connect together and mount. Finally, the last 40 minutes will be spent on testing and verifying that everything is in a working order.

### 4.3 Person-Hours

There are other costs that also need to be considered, the first being person-hours for the development. From the time sheet that we filled out throughout the project there was an estimated 700 hours that was spent on developing and building the quad-copter. At £75 an hour this comes to £52,500. This time can be split up into different categories being software development, board production and chassis manufacture, and development and debugging. The

breakdown for these categories is 300 hours, 50 hours and 350 hours respectively.

### 4.4 Conformance Marking and Manufacturing Set-Up

The extra costs that are involved with manufacturing a product are getting a CE mark at £2,000 and the manufacturing setup costs at £100,000.

### 4.5 Expected Profits

Finally, calculations were made to find out how many quad-copters needed to be sold in order to turn a profit. If they are sold at £300 which would be a profit margin of 46.7% at a 100% yield we would have to sell 1105 units to break even. In reality you will never get a 100% yield so at a worst case of 90% manufacturing yield we would need to sell 1228 units. These values are not too unreasonable as for a delivery company as, in order to get a large coverage of a country a company would need at least 2000 quad-copters which would be a very good start in order to turn a profit.

# 5 Final Product

## 5.1 Specification vs. Achievements

| Specifications | Achievements |
|---|---|
| Arming system for the motors to ensure safety compliance | We made 4 propeller guards from reusable coat hangers by reshaping it to the right size and dimension and were proven to be reliable and cost effective. |
| Motor speed controlled by 1-2ms PWM ESCs - $I_{max}$:20 A | Achieved higher resolution of motor speed control within 125-250us using high resolution timers from the Leonardo. |
| RF modules operated using SPI Interface | Achieved this since we were able to receive data that was transmitted from base station Il Matto to drone Il Matto. This proves that the SPI works reliably for sending and retrieving data between the Il Matto and the RF modules. |
| On-board file logging to SD card | Unable to achieve due to lack of time and coding issues with writing data to the SD card. |
| I2C interface at 100 samples per sec with Gyroscope and accelerometer(MPU6050) with built in DMP | Achieved reading data from this sensor at the specified rate. |
| Servo used for cargo acquisition- controlled by PWM | Achieved control of servo using PWM. |
| Laser-cut acrylic chassis and assembled using acrylic glue | Achieved successful construction for initial and final chassis designs. |
| I-style design for easier weight distribution and more carrying capacity | Achieved implementing this chassis design but in the final product, we added 'I' supports to the centre as well as to the arms to make the chassis more robust to object/ground collision. |
| Reprogram the PID controller on the fly by changing K values wirelessly for faster tuning | Program has been developed on the base station to send k values but has be modified for sending these values while the drone is on ground only. Also this method of re-tuning has not be used due to risk of a fault in wireless up-link and limited time for tuning of the PID controller. |
| Interfaced with the pilot using two X-Y joystick potentiometers, a bank of function switches, and a micro-controller | Achieved a remote controller module consisting of these components which has been designed, programmed and tested. |
| Sharp GP2Y0A41SK0F IR sensor providing altitude sensing for telemetry with accuracy from 15 to 100cm | Sensor has shown to work reliably for the specified range and also implemented receiving IR sensor data for telemetry. |

Table 2: Table comparing the Specification to the Achieved results

## 5.2 Working of the Final Product

The remote controller interface consist of joystick potentiometers which are read into the Il Matto using Analogue to Digital converter (ADC) interface. The values are then sent through SPI interface to the RF transceiver modules for wireless up-link of the joystick values to the drone. The RF transceiver on the drone then sends the received values to the Il Matto on the drone through the same SPI interface. The same Il Matto on the drone then sends these values to the Arduino Leonardo through UART interface. Finally the received joystick values in the Leonardo get translated to the corresponding PWM output signals for controlling the speed of the 4 motors. A stabilization controller program is used in the Leonardo which reads the data from the accelerometer/gyroscope sensor as feedback to ensure that the drone maintains stable flight. The serial cable in the remote controller is used to view telemetry of the battery voltage on the drone and the IR sensor data through a serial terminal on the PC, when one of the switches is in 'flight mode'. This switch also allows you to enter PID values for yaw, pitch and roll for tuning the stabilization program on the drone when in 'PID mode'.

## 5.3 Extensions

- One of the features that was not achieved was the logging of telemetry data on to the SD card which could have been achieved if we were provided more time for it. We are able to collect live data over the communications interface though.
- Instead of using the PC as a User Interface for telemetry and PID tuning, a TFT screen could have been implemented to make the User interface more localised to the remote.
- Being able to overcome the limitation of the User interface for entering different range of PID values and able to tune the stabilization controller on the drone during its flight.
- Adding IR sensors to the front and bottom of the drone to allow it to detect obstacles and control its motors accordingly to avoid collision.
- Implementing an auto landing feature for the drone to use the bottom IR sensor to land itself safely. The 4th switch on the remote was meant to enable this operation, where when switched on, the drone would reject control commands from the controller and would self land onto ground by accordingly adjusting its motor speed to ensure secure landing.

# A Design Completion Form

GANGES (TWO - SIDED FORM)

**Appendix D: Design Completion Form – Team Ganges**

| Component of system/Milestone | Supervisor | Time/ Date | Comments (all/part/none working; protoboard/constructed) |
|---|---|---|---|
| Receive Log Data from Arduino on II Matto | | | 1 log/s |
| Write data from II Matto to SD Card/ Read Data from SD card in computer | | | |
| Collect gyro data and Battery level and write to SD card via II Matto | | | |
| Read battery voltage into II Matto | RM | 2pm 13th | 11.1V -> 3.3V using Potential Divider, into 10-bit ADC channel Simulated using a pot connected to Vcc, rather than battery |
| Actuate Servo using PWM signal | RM | 2pm 13th | 50Hz with 1-2ms pulse A bit shaky but the electronics works |
| Servo controlled by ground control switches | RM | 2pm 13th | Toggle between two positions Done |
| Verify operation of Power Distribution Board | RM | 4pm 13th | 11.1V to each ESC, required power to all on-board microcontrollers everything is powered |
| Transfer Pot values from base transmitter to arduino via IIMatto | RM | 08-03-17 | |
| Achieve lead to end transmission of distance values to vary motors | RM | 11:15 08/03/17 | |
| Stable single axis flight | [illegible] | 1590 13th | Tethered to keep roll, stable in pitch for hover |

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 D: Design Completion Form – Team Ganges**

| Component of system/Milestone | Supervisor | Time/ Date | Comments (all/part/none working; protoboard/constructed) |
|---|---|---|---|
| UAV takes off for a short time with stable flight | | | 10 seconds of stable flight @ 20cm |
| UAV sustains stable flight and lands in a small target area | | | Within a 40 x 40cm area |
| Complete system is integrated and can lift a cargo of ____ g | gvm | 4pm 6th | Mass without cargo _____ g, Dimensions without cargo_____ cm acrylic frame assembled |
| Power management of complete UAV | RM | 4pm 13th | _____ mA , at _____ V = _____ mW. Batteries should last for _____ hrs. It is powered |
| Read angles from gyro over I2C | RM | 3pm 6th | 100 samples/s ✓ arduino reading from gyro on breadboard. |
| Correct PWM outputs from Arduino over four channels | GVM | 3pm 6th | 10% max duty cycle, 50Hz, Check with scope Sorted out PWM channels from arduino. Seen on scope, listening to 1/3/4 J106 |
| PID testing with servo and gyro input | RM | 3pm 6th | Stable for -70 -> 70 deg breadboard, arduino, servo - PID rotations level tilt |
| Transmit instruction packets from II Matto to Arduino | gvm | 6th 16:30 | @ 115200 baud 100 packets per second II matto sends packet containing thrust, pitch, yaw & roll to arduino, check loopr. Shown on scope. |
| Calibrate the ESCs so they power up consistently | RM | 10:45 08-03-17 | Verify using scope |
| ESCs interfaced with Arduino to independently control motors | RM | 10:45 08-03-17 | |
| Transmitting data from II Matto using RFM12b-S2 | RM | 3pm 6th | see below |
| Receive transmission on II Matto from RFM12b-S2 | RM | 3pm 5th | ~100 packets per second II Matto -> RFM -> RFM -> II Matto -> Hyperterminal |
| Interface one II Matto with 2 transceivers for bi-directional RF comms | RM | 4pm 13th | Bidirectional comms UAV <-> controller |
| Achieve two-way communications between two II Mattos | RM | 2pm 13th | On board receive at 100 packets/s with transmission back at 1 packet/s two II Mattos connected to Hyper terminal |
| 4 10-Bit ADC Potentiometer readings from the controller to the II Matto. | gvm | 2pm 6th | 2 pot joysticks -> II Matto -> serial terminal. Some crosstalk between |
| | gvm | 16:30 6th | the two channels (significant). Crosstalk fixed. |
| UI from PC sends K values to the ground comms II Matto | gvm | 11:00 10th | Ground comms sufficient between tx'ing pot flight controls, or PID adjust through terminal |
| PID k values transferred from the PC to the drone via the RF comms | gvm | 11:00 10th | drone receiving controller pot data or PID updates via RFM12 J2. PC -> (UART) -> Base II Matto -> (SPI) -> Radio link -> (SPI) -> Drone II Matto |

Milestones finalised by supervisor: Steve Quinn ..... Signed ..... Date
Prototype hardware handed over to: ..... Signed ..... Date
Other items returned to Lab support hatch and checked by: ..... Signed ..... Date

# B Project Completion Form

Cost estimates

The final estimated costings are outlined in section 4 of the main team report.

Our estimate at the beginning of the project of the total development and manufacturing costs was £135,143.55. The actual total came to £154,679.32. This £19,535.77 difference is largely due to the man hours being significantly more than expected (700 vs 440). Development and testing was a more time-consuming task than we thought initially.

Design Changes

The design remained largely unchanged from that which was laid out in the Project Proposal Form. A couple of features, SD card logging and remote PID tuning, could not be implemented fully due to time-constraints from more critical systems.

Discrepancy in Project activities

Development and testing of the communications took longer than anticipated and as such the tesing of the RFM12B-S2 modules was also done on Monday. Encryption was a less time-critical feature and testing of it was done after the two-way communications was comprehensively tested.

Battery voltage telemetry and operation of PDB were not completed.

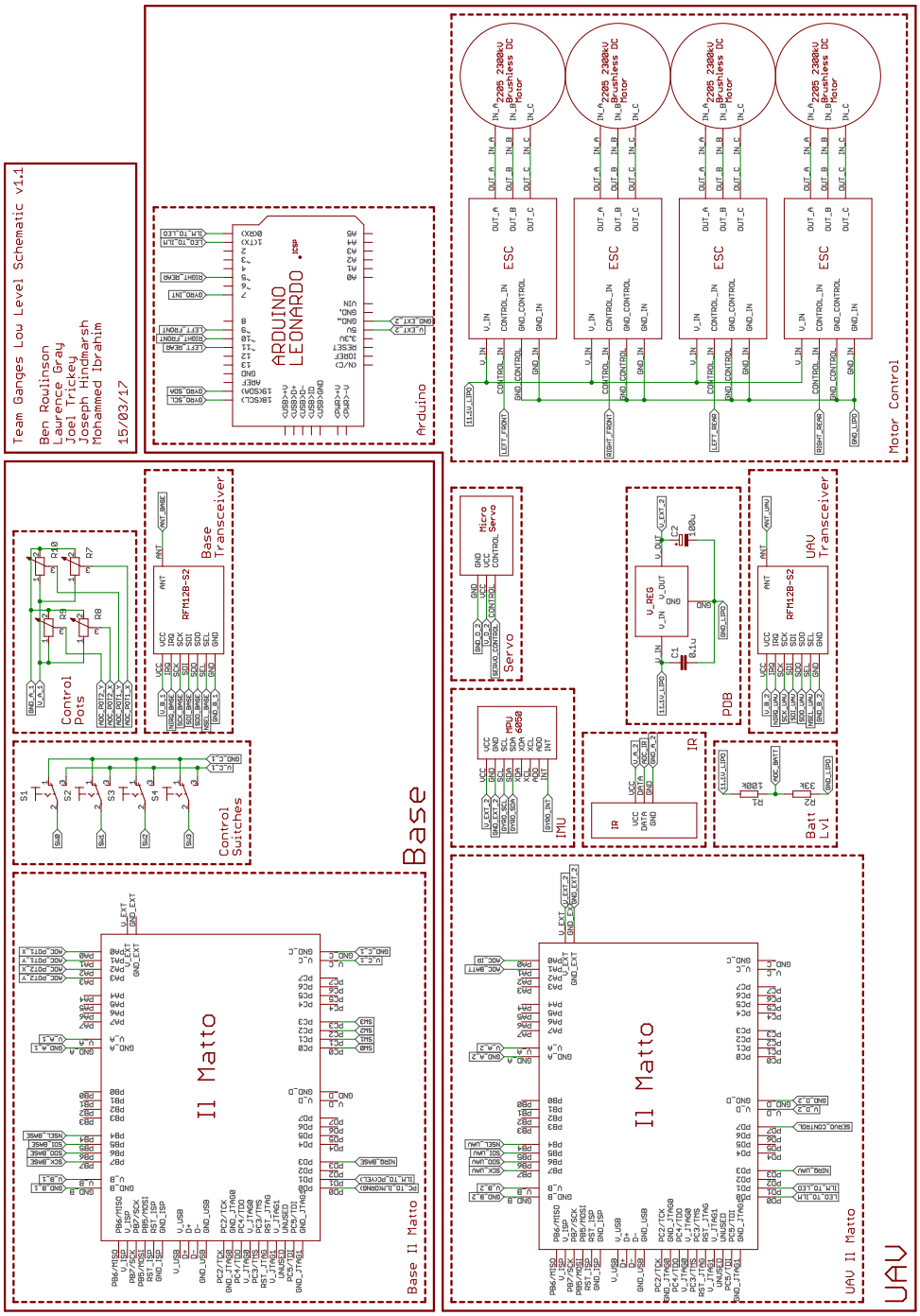
The rest of our activities were undertaken just as planned.

**Actual Project Activities**

| Activity | Initials | Fri am | Fri pm | Mon am | Mon pm | Tue | Wed | Thu | Fri am | Fri pm | Mon am | Mon pm |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Testing RFM12B-S2 modules using II Matto | JT | X | X | X | X | | | | | | | |
| Testing RFM12B-S2 modules using II Matto | MI | X | X | X | X | | | | | | | |
| Sending ADC values using RFM12B-S2 | MI | | | X | X | | | | | | | |
| Reading Gyro data over I2C to Arduino | JH | X | X | X | X | | | | | | | |
| Testing encryption system for Communications | JT | | | | | X | X | | | | | |
| End to End transmission of motor values | JT | | | | | X | X | | | | | |
| Testing PID controller using gyro and servo | LG | X | X | | | | | | | | | |
| Manufacture and Assembly of the Chassis | LG | X | X | X | X | | | | | | X | X |
| Manufacture and Assembly of the Chassis | BR | X | X | | | | | | | | | |
| Integrating PID controller with external desired throttle | BR | | | | | X | X | | | | | |
| Test data rate of the Gyro | JH | | | X | | | | | | | | |
| Test IR sensor over a range of distances | BR | | | | X | | | | | | | |
| ~~PID testing with servo and gyro input~~ | LH | | | | | | | | | | | |
| UI for real-time PID tuning | MI | | | | | | X | X | | | | |
| Test ESCs and motors | LG | | | | | X | X | X | | | | |
| Test two-way communications | JT | | | | | X | X | | | | | |
| Tune communications code for optimal data transfer | JT | | | | | | | X | | | | |
| Test piloting system – XY pots and switches | MI | | | X | X | X | | | | | | |

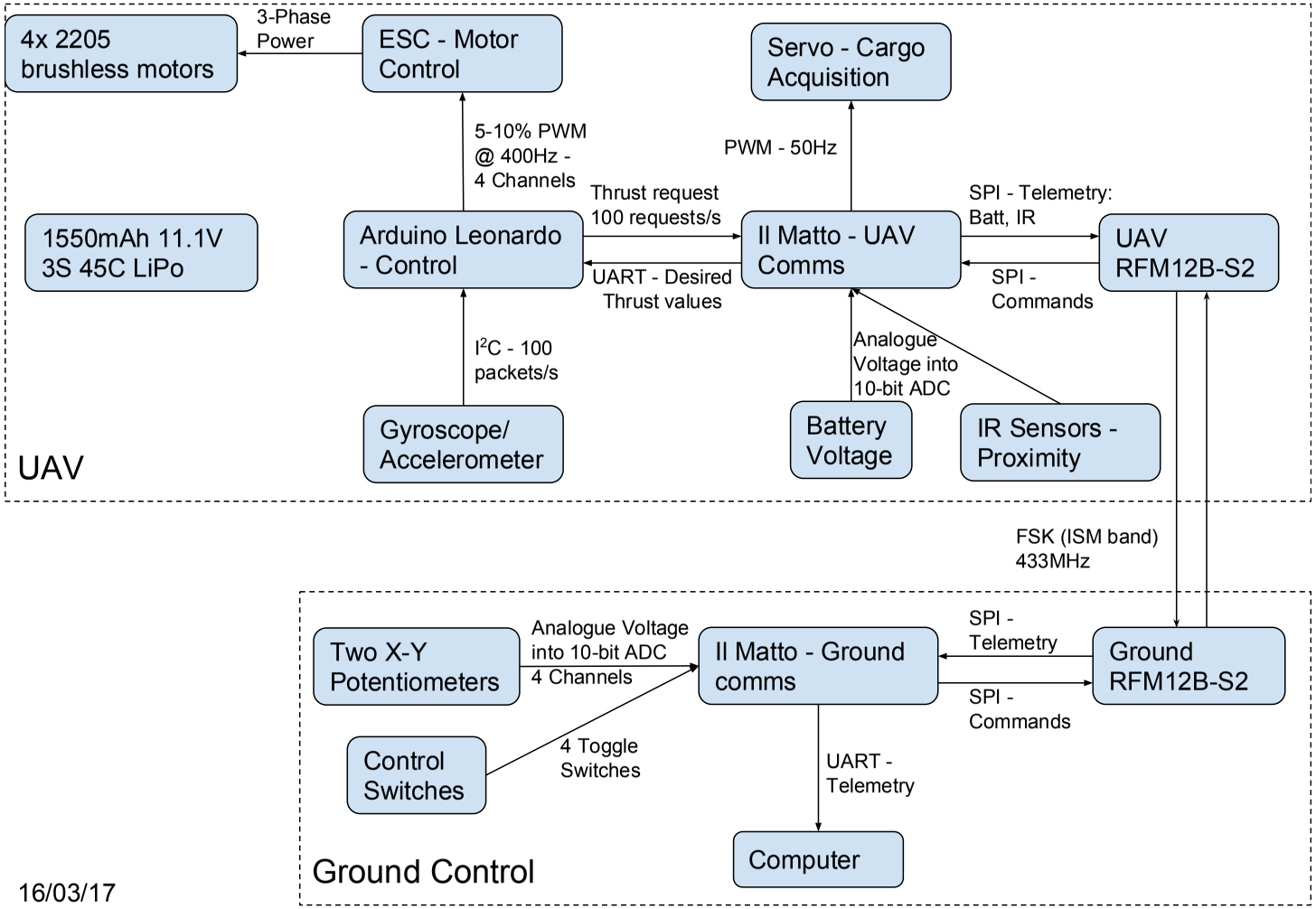
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ıne PID controller | BR | | | | | | | X | X | X | X | X |
| ıne PID controller | LG | | | | | | | X | X | X | X | X |
| ead in battery voltage | BR | | | | | | X | | | | | |
| ower distribution | BR | | | | | | X | X | | | | |
| esting Serial communications between Arduino and II atto | ~~JH~~ BR | | | X | X | | | | | | | |
| 10-Bit ADC Potentiometer readings from the controller the II Matto. | MI | | X | X | | | | | | | | |
| l from PC sends K values to the ground comms II Matto | MI | | | | | | X | X | | | | |
| D k values transferred from the PC to the drone via the comms | MI | | | | | | | | X | | | |
| eceive Log Data from Arduino on II Matto | JT | | | | | | X | | | | | |
| ~~ead battery voltage into II Matto~~ | BR | | | | | | | | | | | |
| ~~erify operation of Power Distribution Board~~ | BR | | | | | | | | | | | |
| ot values from base to Arduino via RF | BR | | | | | X | X | | | | | |
| ontrol Servo using PWM | JH | | | | X | X | | | | | | |
| ability in single-axis testing | LH | | | | | | | | X | X | X | X |
| ability in single-axis testing | BR | | | | | | | | X | X | X | X |

# C Circuit Diagrams




15/03/2017 14:48 C:\Users\Ben Rowlinson\Documents\GitHub\D4-Ganges\Schematic\onboard_v1.1.sch (Sheet: 1/1)

D4 - Ganges High Level Block Schematic - rev5




16/03/17

# D Software Listings

## D.1 PID_values_test_2.c

```c
// Code for entering PID constants values on putty and then
// step 1: get them ton display on screen
// step 2: Once step 1 is done get it to display then figure out how to convert them to 8 bit values that show a resolution between 0 and 1
// and display them
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>
//#include "debug.h"
#define BAUD 9600                                   // define baud
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)            // set baud rate value for UBRR
#define CHAR_MAX 6
//********** from servo.c

#define PWM_DUTY_MAX 240
#define PWM_DUTY_MIN 0
#define PWM_PRESCALER 8UL
#define PWM_FREQUENCY 50
#define PWM_OFFSET 0
//**********

/*Includes usart.h header file which defines different functions for USART. USART header file version is 1.1*/
void adc_init()//[1]
{

 // In ADCSRA Enable ADC (set ADEN) and prescaler of 64
 ADCSRA |= _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);
}
void init_pwm(void)
{
    /* TIMER 2 */
    DDRD |= _BV(PD6); /* PWM out */
    DDRD |= _BV(PD7); /* inv. PWM out */


    TCCR2A = _BV(WGM20) | /* fast PWM/MAX */
      _BV(COM2A1); /* A output */
    TCCR2B = _BV(CS21)  |
             _BV(CS22);   /* 1/256 prescaling */
}
uint16_t adc_read(int n)//[1]
{
 ADMUX = n;// represents PA2
 // start conversion
 ADCSRA |= _BV(ADSC);
```

```c
 // wait for conversion to complete
 //while(!(ADCSRA & _BV(ADIF))){};
 while(ADCSRA & _BV(ADSC));
 ADC = (ADCH << 8) | ADCL;// [1]
 return ADC;
}
void pwm_duty(uint8_t duty)// from servo.c
{
 duty = duty > PWM_DUTY_MAX ? PWM_DUTY_MAX : duty;
 duty = duty < PWM_DUTY_MIN ? PWM_DUTY_MIN : duty;
     //printf("\nPWM=%3u  ==>  ", duty);
 OCR2A = duty;
}
void init_uart1()// initialize UART
{
  //1. set the baud rate, lets configure to 9600;
 // set the baud rate registers Ref: [1],[2]
 UBRR0H = BAUDRATE >> 8;// UBRRnH is 8 bits left
 UBRR0L = BAUDRATE;

  //2. setting up data packet: 8 bits ,no parity 1 stop bit
  // setting 8 bits got to UCSCR register Ref:[3], pg 185 of data sheet

 UCSR0C = _BV(UCSZ00) | _BV(UCSZ01); // 8 bits, USBS1 = 0 for 1 stop bit

  // note: havnt set up the stop bit in Ref [2] slides
 // 3. from Ref[2] we now enable Transmission and receive n UCSRnB register
 UCSR0B = _BV(TXEN0) | _BV(RXEN0) | _BV(RXCIE0);// enable rx interrupt

}
// transmit data function to transmit to the screen
void uart_transmit( char data)
{
 while(!( UCSR0A &  _BV(UDRE0) ) ); //  data register enable bit is 1 if tx buffer is empy
 // if its 1 we load data onto UDR- Uart Data Register(buffer)
 UDR0 = data;
}
// reveive data function to receive values entered on screen
// char uart_receive()
// {
//  if (!(UCSR0A &  _BV(RXC0) ))// if there is unread data in the receive whihc needs to be read
//   return UDR0;
//  else
//   return NULL;
// }
void send_string(char *str)
{
 int i;
 for( i = 0; str[i]; i++) uart_transmit(str[i]);
}//**************
char k[CHAR_MAX];
```

```c
volatile uint8_t counter = 0;// itss uint8_t for size saving
volatile uint8_t pid_enter_check = 0; // used to ensure number(eg: 10.230) is not passed to k[] when asked to enter p or i or d
volatile char pid;
char buff[10];
ISR(USART0_RX_vect)
{
 char temp[CHAR_MAX];
 temp[counter] = UDR0;
 if (temp[counter] == 'p' || temp[counter] == 'i' || temp[counter] == 'd')
 {
  pid = temp[counter];
  uart_transmit(pid);// print it to see what we enter
 }
 else if (temp[counter] == 'x')
 {
  counter = 0;//reset the counter to zero so that u can start re-writing to the k[] array
  send_string("\n\r Re-enter K value: ");
 }
 else if ( (pid_enter_check == 1) && (temp[counter] > 47 || temp[counter] < 58) )
 {
  uart_transmit(temp[counter]);
  send_string("\n\r p or i or d pls!: ");
  counter = 0;// ensure buffer isnt filled with these numbers by re-setting this counter
  pid_enter_check = 0;
 }
 //  ASCII : 47 < x < 75 corresponds to integers 0-9 and x = 46 whihc is a 'dot' which we EXCLUDE in this condition to DEAL with the "other characters" apart fron 'p' or 'i' or 'd'
 else if ( (temp[counter] < 47 || temp[counter] > 57) && (temp[counter] != 46) ) // if what you entered are letters like 'l' or 'z' etc. when asked to enter p or i or d and
 {
  send_string("\n\r Please enter p or i or d: "); // ask to re enter p or i or d
  counter = 0;// BUT if numbers were ented when asked to enter p or i or d
  // re-write the the k[] buffer to get rid of those numbers entered
 }
 else // if numbers or dots were entered
  {
   k[counter] = temp[counter];// can store digits or dots and incremet counter
   uart_transmit(k[counter]);// print it to see what we enter
   counter++;// incremet to get next digit
  }
    // do an else in case its an invalid character entery other than i or p or d

 // ki[counter] = UDR0;// read the
 // //send_string("\n\rin ISR");
 // uart_transmit(ki[counter]);
 // counter++;
 // //sprintf(buff,"%d",counter);
 // //send_string(buff);
```

```
}

int main()
{
 init_uart1();
 init_pwm();
 adc_init();
 //init_debug_uart0();
 sei();// enable global interrupt
 char ch[60];
 float f, f_temp;
 uint16_t ten_bit;
 send_string("\n\r Enter a K type (p or i or d): ");
 pid_enter_check = 1;// always set this variable to 1 when asked to enter p or i or d
 uint16_t adc_value;
 uint8_t pwm_value;

 while(1)
 {

  //receive_string(ki);
  //ki[i] = uart_receive();
  adc_value = adc_read(0);
  pwm_value = (uint8_t) (adc_value/4) + PWM_OFFSET;
  pwm_duty(pwm_value);
  // we are having a servio as way of checking if this code doesnt block the flow of this while loop significantly
  if (pid == 'p' || pid == 'i' || pid == 'd')
   {
    send_string("\n\r Enter k value (press x to re-enter): ");
    // BEFORE NULLING IT transmit it!!!!!!!!!!!!!!!
    //**************
    //**************
    pid = NULL;
   }

  else if (counter == CHAR_MAX)// char bufer is ready to transmit
  {
   counter = 0;// back to zero
   f = atof(k);// convert it to float
   f_temp = f*100; //
   if (f_temp > 1023)// check if number not withing range
   {
    send_string("\n\r Error: k value is not in range!");
    send_string("\n\r Enter a K type (p or i or d): ");
    continue;
   }
   ten_bit = (uint16_t) (f_temp+0.5);
   sprintf(ch,"\n\r 10-bit dec value: %d",ten_bit);
   send_string(ch);
   // transmit its
```

```
    send_string("\n\r Enter a K type (p or i or d): ");
    pid_enter_check = 1;// always set this variable to 1 when asked to enter p or i or d
   }
   else
    continue;


  }
}
```

## D.2 PID_values_test_3.c

```
// Code for entering PID constants values on putty and then
// step 1: get them ton display on screen
// step 2: Once step 1 is done get it to display then figure out how to convert them to 8 bit values that show a resolution between 0 and 1
// Note : compared to PID_values_test_2 we need to add ask user to input pid values for yaw pitch and roll
// and display them
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>
//#include "debug.h"
#define BAUD 57600//9600                                    // define baud
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)            // set baud rate value for UBRR
#define CHAR_MAX 6
//********** from servo.c

#define PWM_DUTY_MAX 240
#define PWM_DUTY_MIN 0
#define PWM_PRESCALER 8UL
#define PWM_FREQUENCY 50
#define PWM_OFFSET 0
//**********

/*Includes usart.h header file which defines different functions for USART. USART header file version is 1.1*/
void adc_init()//[1]
{

 // In ADCSRA Enable ADC (set ADEN) and prescaler of 64
 ADCSRA |= _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);
}
void init_pwm(void)
{
    /* TIMER 2 */
    DDRD |= _BV(PD6); /* PWM out */
    DDRD |= _BV(PD7); /* inv. PWM out */
```

```c


    TCCR2A = _BV(WGM20) | /* fast PWM/MAX */
      _BV(COM2A1); /* A output */
    TCCR2B = _BV(CS21)  |
             _BV(CS22);   /* 1/256 prescaling */
}
uint16_t adc_read(int n)//[1]
{
 ADMUX = n;// represents PA2
 // start conversion
 ADCSRA |= _BV(ADSC);
 // wait for conversion to complete
 //while(!(ADCSRA & _BV(ADIF))){};
 while(ADCSRA & _BV(ADSC));
 ADC = (ADCH << 8) | ADCL;// [1]
 return ADC;
}
void pwm_duty(uint8_t duty)// from servo.c
{
 duty = duty > PWM_DUTY_MAX ? PWM_DUTY_MAX : duty;
 duty = duty < PWM_DUTY_MIN ? PWM_DUTY_MIN : duty;
     //printf("\nPWM=%3u  ==>  ", duty);
 OCR2A = duty;
}
void init_uart1()// initialize UART
{
  //1. set the baud rate, lets configure to 9600;
 // set the baud rate registers Ref: [1],[2]
 UBRR0H = BAUDRATE >> 8;// UBRRnH is 8 bits left
 UBRR0L = BAUDRATE;

  //2. setting up data packet: 8 bits ,no parity 1 stop bit
  // setting 8 bits got to UCSCR register Ref:[3], pg 185 of data sheet

 UCSR0C = _BV(UCSZ00) | _BV(UCSZ01); // 8 bits, USBS1 = 0 for 1 stop bit

  // note: havnt set up the stop bit in Ref [2] slides
 // 3. from Ref[2] we now enable Transmission and receive n UCSRnB register
 UCSR0B = _BV(TXEN0) | _BV(RXEN0) | _BV(RXCIE0);// enable rx interrupt

}
// transmit data function to transmit to the screen
void uart_transmit( char data)
{
 while(!( UCSR0A &  _BV(UDRE0) ) ); //  data register enable bit is 1 if tx
     buffer is empy
 // if its 1 we load data onto UDR- Uart Data Register(buffer)
 UDR0 = data;
}
// reveive data function to receive values entered on screen
// char uart_receive()
// {
```

```
//   if (!(UCSR0A & _BV(RXC0) ))// if there is unread data in the receive whihc needs to be read
//    return UDR0;
//   else
//    return NULL;
// }
void send_string(char *str)
{
 int i;
 for( i = 0; str[i]; i++) uart_transmit(str[i]);
}//***************
char k[CHAR_MAX];
volatile uint8_t counter = 0;// itss uint8_t for size saving
volatile uint8_t pid_enter_check = 0; // used to ensure number(eg: 10.230) is not passed to k[] when asked to enter p or i or d
volatile uint8_t tyh_enter_check = 0;
volatile char pid, tyh, pid_copy, tyh_copy;
char buff[10];
ISR(USART0_RX_vect)
{
 char temp[CHAR_MAX];
 temp[counter] = UDR0;
 // if user entrs t/y or h when asked to enter p or i or d then skip this if statement
 // and go to the 5th if statement
 //          |
 //          V
 if ( (pid_enter_check == 0) && (temp[counter] == 'r' || temp[counter] == 'y' || temp[counter] == 'h'))
 {
  tyh_enter_check = 0;// disbale checker as the user has entered thrust or yaw or pitch as requested
  tyh = temp[counter];
  uart_transmit(tyh);
 }
 else if ((tyh_enter_check == 1) && !(temp[counter] == 'r' || temp[counter] == 'y' || temp[counter] == 'h'))//(temp[counter] > 47 || temp[counter] < 58)
 {
  uart_transmit(temp[counter]);
  send_string("\n\r Roll(r) or Yaw(y) or Pitch(h) pls!: ");
  counter = 0;// ensure buffer isnt filled with entered numbers by re-setting this counter
 }
 else if (temp[counter] == 'p' || temp[counter] == 'i' || temp[counter] == 'd')
 {
  pid_enter_check = 0;// disbale checker as the user has entered p or i or d as requested
  pid = temp[counter];
  uart_transmit(pid);// print it to see what we enter
 }
 else if (temp[counter] == 'x')
 {
```

```
  counter = 0;//reset the counter to zero so that u can start re-writing to the k[] array
  send_string("\n\r Re-enter roll(r)/yaw(y)/pitch(h) : ");
  // reset the entry checkers to reset the entry of the details
  tyh_enter_check = 1;// to get to the 2nd if condition

 }
 else if ((pid_enter_check == 1) && !(temp[counter] == 'p' || temp[counter] == 'i' || temp[counter] == 'd'))//(temp[counter] > 47 || temp[counter] < 58)*/ )// incase u entered value when asked to enter p or i or d
 {
  uart_transmit(temp[counter]);
  send_string("\n\r Pls p or i or d!: ");
  counter = 0;// ensure buffer isnt filled with entered numbers by re-setting this counter
 }
 else // if numbers or dots were entered
 {
  k[counter] = temp[counter];// can store digits or dots and incremet counter
  uart_transmit(k[counter]);// print it to see what we enter
  counter++;// incremet to get next digit
 }
 //  ASCII : 47 < x < 57 corresponds to integers 0-9 and x = 46 which is a 'dot' which we EXCLUDE in this condition to DEAL with the "other characters" apart fron 'p' or 'i' or 'd'
 // else if ( (temp[counter] < 47 || temp[counter] > 57) && (temp[counter] != 46) )// if what you entered are letters like 'l' or 'z' etc. when asked to enter p or i or d and
 // {
 //  send_string("\n\r Please enter p or i or d: "); // ask to re enter p or i or d
 //  counter = 0;// BUT if numbers were ented when asked to enter p or i or d
 //  // re-write the the k[] buffer to get rid of those numbers entered
 // }

}

int main()
{
 init_uart1();
 init_pwm();
 adc_init();
 //init_debug_uart0();
 sei();// enable global interrupt
 char ch[60];
 float f, f_temp;
 uint16_t ten_bit;
 send_string("\n\rEnter roll(r)/yaw(y)/pitch(h): ");
 tyh_enter_check = 1;
 uint16_t adc_value;
 uint8_t pwm_value;

 while(1)
```

```
{

 //receive_string(ki);
 //ki[i] = uart_receive();
 adc_value = adc_read(0);
 pwm_value = (uint8_t) (adc_value/4) + PWM_OFFSET;
 pwm_duty(pwm_value);
 // we are having a servio as way of checking if this code doesnt block the flow of this while loop significantly
 if (tyh == 'r' || tyh == 'y' || tyh == 'h')
 {
  send_string("\n\r Enter a K type (p or i or d): ");
  pid_enter_check = 1;// always set this variable to 1 when asked to enter p or i or d
  tyh_copy = tyh;// we need a copy of tyh to display what we have to the user
  tyh = NULL;// definitely clear this as this condition will keep getting satisfied!

 }

 else if (pid == 'p' || pid == 'i' || pid == 'd')
  {
   send_string("\n\r Enter k value of this type: ");
   pid_copy = pid;
   pid = NULL;// definitely clear this as this condition will keep getting satisfied!
  }

 else if (counter == CHAR_MAX)// char bufer is ready to transmit
 {
  counter = 0;// back to zero
  f = atof(k);// convert it to float
  f_temp = f*100; //
  if (f_temp > 1023)// check if number not withing range
  {
   send_string("\n\r Error: k value is not in range!");
   send_string("\n\r Enter a K type (p or i or d): ");
   continue;
  }
  ten_bit = (uint16_t) (f_temp+0.5);
  sprintf(ch,"\n\r 10-bit dec value for k type: %c of %c: %d",pid_copy,tyh_copy,ten_bit);
  send_string(ch);
  // transmit transmit the data for pid_copy,tyh_copy and ten_bi

  send_string("\n\rEnter Thrust(t)/yaw(y)/pitch(h): ");
  tyh_enter_check = 1;
 }
 else
  continue;


}
```

```
}
```

### D.3 test2_basestation_comms.c

```
// This code is now going to involve taking the copy of basestation_comms.c and adding the adc bits from base_station_controller.c

//  date finished : 8/3/17

// Arthur: Mohammed Ibrahim and Joel

// D4-Ganges!!!!!

// This code can:

//- send adc values and cargo on/off during flight mode (switch is on flight mode)
// - send each k type values for each channel as the user enters them(switch is on PID/ UI mode)
// UI is more robust, can re-enter enter details again (starting again from entering yaw/pitch/roll)
// -


#include <avr/io.h>

#include <stdio.h>// for sprintf()

#include <avr/interrupt.h>

#include <util/delay.h>

#include <math.h>

#include <stdlib.h>

#include "rfm12.h"

#include "basestation_comms.h"

//***************
char k[CHAR_MAX];
volatile uint8_t counter = 0;// itss uint8_t for size saving
volatile uint8_t pid_enter_check = 0; // used to ensure number(eg: 10.230) is not passed to k[] when asked to enter p or i or d
volatile uint8_t tyh_enter_check = 0;
volatile char pid, tyh, pid_copy, tyh_copy;
char buff[10];
ISR(USART0_RX_vect)
{
 char temp[CHAR_MAX];
 temp[counter] = UDR0;
```

```
// if user entrs t/y or h when asked to enter p or i or d then skip this if statement
// and go to the 5th if statement
//          |
//          V
if ( (pid_enter_check == 0) && (temp[counter] == 'r' || temp[counter] == 'y' || temp[counter] == 'h'))
{
 tyh_enter_check = 0;// disbale checker as the user has entered thrust or yaw or pitch as requested
 //send_string("\n\r FLAGGG");
 tyh = temp[counter];
 uart_transmit(tyh);
}
else if ((tyh_enter_check == 1) && !(temp[counter] == 'r' || temp[counter] == 'y' || temp[counter] == 'h'))//(temp[counter] > 47 || temp[counter] < 58)
{
 uart_transmit(temp[counter]);
 send_string("\n\r Roll(r) or Yaw(y) or Pitch(h) pls!: ");
 counter = 0;// ensure buffer isnt filled with entered numbers by re-setting this counter
}
else if (temp[counter] == 'p' || temp[counter] == 'i' || temp[counter] == 'd')
{
 pid_enter_check = 0;// disbale checker as the user has entered p or i or d as requested
 pid = temp[counter];
 uart_transmit(pid);// print it to see what we enter
}
else if (temp[counter] == 'x')
{
 counter = 0;//reset the counter to zero so that u can start re-writing to the k[] array
 send_string("\n\r Re-enter roll(r)/yaw(y)/pitch(h) : ");
 // reset the entry checkers to reset the entry of the details
 tyh_enter_check = 1;// to get to the 2nd if condition

}
else if ((pid_enter_check == 1) && !(temp[counter] == 'p' || temp[counter] == 'i' || temp[counter] == 'd'))//(temp[counter] > 47 || temp[counter] < 58)*/ )// incase u entered value when asked to enter p or i or d
{
 uart_transmit(temp[counter]);
 send_string("\n\r Pls p or i or d!: ");
 counter = 0;// ensure buffer isnt filled with entered numbers by re-setting this counter
}
else // if numbers or dots were entered
{
 k[counter] = temp[counter];// can store digits or dots and incremet counter
 uart_transmit(k[counter]);// print it to see what we enter
 counter++;// incremet to get next digit
}
```

```
 //  ASCII : 47 < x < 57 corresponds to integers 0-9 and x = 46 which is a 'dot' which we EXCLUDE in this condition to DEAL with the "other characters" apart fron 'p' or 'i' or 'd'
 // else if ( (temp[counter] < 47 || temp[counter] > 57) && (temp[counter] != 46) )// if what you entered are letters like 'l' or 'z' etc. when asked to enter p or i or d and
 // {
 //  send_string("\n\r Please enter p or i or d: "); // ask to re enter p or i or d
 //  counter = 0;// BUT if numbers were ented when asked to enter p or i or d
 //  // re-write the the k[] buffer to get rid of those numbers entered
 // }

}
//********************************************** from PID_control_test_3

int main(void)

{

 // Initialise rfm12, adc, uart and interrupts

 rfm12_init();

 adc_init();

 init_uart1();

 sei();
 //*************************************
 #if ENABLE_UI
 char ui[60];
 float f, f_temp;
 uint16_t ten_bit;
 send_string("\n\rEnter roll(r)/yaw(y)/pitch(h): ");
 tyh_enter_check = 1;
 //uint16_t adc_value;
 //uint8_t pwm_value;
 #endif
 //****************************************
 #if UPLINK_TEST

  uint16_t testdata;

  testdata = 0;

 #endif



 // ADC channels
    #if ENABLE_CONTROLS
 uint16_t thrust, yaw, pitch, roll;
```

```
// telemetry values
uint8_t packettype;
uint16_t data;
// Button for cargo
DDRA &= ~_BV(PA6);// enable input
PORTA |= _BV(PA6);// enable pull ups
DDRA &= ~_BV(PA7);// enable input
PORTA |= _BV(PA7);// enable pull ups

DDRD |= _BV(PD6);// SETto output
PORTD |= _BV(PD6);// SET TO HIGH
DDRD |= _BV(PD7);// SETto output
PORTD |= _BV(PD7);// SET TO HIGH
// button states
uint8_t button_present, button_previous;
uint8_t flight_present, flight_previous;
flight_previous = 0;
button_previous = 0;

char ch[40];
#endif
Send_data(OP_BUTTON, flight_present = (PINA & _BV(PA7))? BTN_FLIGHT_MODE:
    BTN_PID_MODE);
while (1)

{

 flight_present = (PINA & _BV(PA7))? 1:0;

 if ((flight_present == 1) && (flight_previous == 0))
 {
  Send_data(OP_BUTTON,BTN_FLIGHT_MODE);
  _delay_ms(2);
  Send_data(OP_BUTTON,BTN_FLIGHT_MODE);
  _delay_ms(2);
  rfm12_tick();
 }
 else if ((flight_present == 0) && (flight_previous == 1))
 {
  // we are having a servio as way of checking if this code doesnt block the
      flow of this while loop significantly
  Send_data(OP_BUTTON,BTN_PID_MODE);
  _delay_ms(2);
  Send_data(OP_BUTTON,BTN_PID_MODE);
  _delay_ms(2);
  rfm12_tick();
  send_string("\n\rEnter roll(r)/yaw(y)/pitch(h): ");
 }
 flight_previous = flight_present;
  // check the rx buffer
 if (flight_present)
 {
```

```
    if (rfm12_rx_status() == STATUS_COMPLETE)
    {
     packettype = rfm12_rx_type();
     data = *rfm12_rx_buffer();

     Retrieve_data(&packettype, &data);
     sprintf(ch, "\n\rPacket type: %u and Data: %u", packettype, data);
     send_string(ch);
     rfm12_rx_clear();
    }
    //else
     //send_string("\n\r No telemetry :(");

    thrust = adc_read(PIN_THRUST);

    Send_data(OP_THRUST, thrust);
    _delay_ms(2);
    Send_data(OP_THRUST, thrust);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!!

    yaw = adc_read(PIN_YAW);


    Send_data(OP_YAW, yaw);
    _delay_ms(2);
    Send_data(OP_YAW, yaw);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!!


    pitch = adc_read(PIN_PITCH);


    Send_data(OP_PITCH, pitch);
    _delay_ms(2);
    Send_data(OP_PITCH, pitch);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!!

    roll = adc_read(PIN_ROLL);


    Send_data(OP_ROLL, roll);
    _delay_ms(2);
    Send_data(OP_ROLL, roll);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
    button_present = (PINA & _BV(PA6)) ? 1:0;
    //sprintf(ch,"\n\rbutton_present = %u", button_present);
    //send_string(ch);
    if ((button_present == 1) && (button_previous == 0))
    {
     send_string("\n\rCargo hook up!");
     Send_data(OP_BUTTON, BTN_CARGO_HOOK_UP);
     _delay_ms(2);
     Send_data(OP_BUTTON, BTN_CARGO_HOOK_UP);
     _delay_ms(2);
     rfm12_tick();

    }
    else if ((button_present == 0) && (button_previous == 1))
    {
     send_string("\n\rCargo hook down!");
     Send_data(OP_BUTTON, BTN_CARGO_HOOK_DOWN);
     _delay_ms(2);
     Send_data(OP_BUTTON, BTN_CARGO_HOOK_DOWN);
     _delay_ms(2);
     rfm12_tick();
    }
    button_previous = button_present;
   }
   else if (!flight_present)
   {
    if (tyh == 'r' || tyh == 'y' || tyh == 'h')
    {
     send_string("\n\r Enter a K type (p or i or d): ");
     pid_enter_check = 1;// always set this variable to 1 when asked to enter p or i or d
     tyh_copy = tyh;// we need a copy of tyh to display what we have to the user
     tyh = NULL;// definitely clear this as this condition will keep getting satisfied!

    }

    else if (pid == 'p' || pid == 'i' || pid == 'd')
     {
      send_string("\n\r Enter k value of this type: ");
      pid_copy = pid;
      pid = NULL;// definitely clear this as this condition will keep getting satisfied!
     }

    else if (counter == CHAR_MAX)// char bufer is ready to transmit
    {
     counter = 0;// back to zero
     f = atof(k);// convert it to float
     f_temp = f*100; //
     if (f_temp > 1023)// check if number not withing range
     {
```

```
      send_string("\n\r Error: k value is not in range!");
      send_string("\n\r Enter a K type (p or i or d): ");
      continue;
    }
    ten_bit = (uint16_t) (f_temp+0.5);
    sprintf(ui,"\n\r 10-bit dec value for k type: %c of %c: %d",pid_copy,tyh_copy
        ,ten_bit);
    send_string(ui);
    // transmit the data for pid_copy,tyh_copy and ten_bi
    switch(tyh_copy)
    {
     case 'r':   if (pid_copy == 'p')
         {
          Send_data(OP_KP_ROLL,ten_bit);
          _delay_ms(2);
          // Send_data(OP_KP_ROLL,ten_bit);
          // _delay_ms(2);
          rfm12_tick();
         }
        else if (pid_copy == 'i')
        {
         Send_data(OP_KI_ROLL,ten_bit);
         _delay_ms(2);
         // Send_data(OP_KI_ROLL,ten_bit);
         // _delay_ms(2);
         rfm12_tick();
        }
        else if (pid_copy == 'd')
        {
         Send_data(OP_KD_ROLL,ten_bit);
         _delay_ms(2);
         // Send_data(OP_KD_ROLL,ten_bit);
         // _delay_ms(2);
         rfm12_tick();
        }
        break;
     case 'y':   if (pid_copy == 'p')
         {
          Send_data(OP_KP_YAW,ten_bit);
          _delay_ms(2);
          // Send_data(OP_KP_YAW,ten_bit);
          // _delay_ms(2);
          rfm12_tick();
         }
        else if (pid_copy == 'i')
        {
         Send_data(OP_KI_YAW,ten_bit);
         _delay_ms(2);
         // Send_data(OP_KI_YAW,ten_bit);
         // _delay_ms(2);
         rfm12_tick();
        }
        else if (pid_copy == 'd')
```

```
        {
         Send_data(OP_KD_YAW, ten_bit);
         _delay_ms(2);
         // Send_data(OP_KD_YAW, ten_bit);
         // _delay_ms(2);
         rfm12_tick();
        }
        break;
    case 'h':   if (pid_copy == 'p')
         {
          Send_data(OP_KP_PITCH, ten_bit);
          _delay_ms(2);
          // Send_data(OP_KP_PITCH, ten_bit);
          // _delay_ms(2);
          rfm12_tick();
         }
        else if (pid_copy == 'i')
        {
         send_string("\n\rFLAG!");
         Send_data(OP_KI_PITCH, ten_bit);
         _delay_ms(2);
         // Send_data(OP_KI_PITCH, ten_bit);
         // _delay_ms(2);
         rfm12_tick();
        }
        else if (pid_copy == 'd')
        {
         Send_data(OP_KD_PITCH, ten_bit);
         _delay_ms(2);
         // Send_data(OP_KD_PITCH, ten_bit);
         // _delay_ms(2);
         rfm12_tick();
        }
        break;
    }
    // clear pid_copy and tyh type as they get transmitted again
    pid_copy = NULL;
    tyh_copy = NULL;
    send_string("\n\rEnter roll(r)/yaw(y)/pitch(h): ");
    tyh_enter_check = 1;
    counter = 0;// to stop receivng these datas event houghtb i havnt entered anything
   }
   else
    continue;
  }


 }



 // Allows program to stop
```

```

 while (1) {};

}
```

## D.4 test3_basestation_comms.c

```
// This code is now going to involve taking the copy of basestation_comms.c and adding the adc bits from base_station_controller.c

//  date: 1/3/17

// Arthur: Mohammed Ibrahim and Joel

// D4-Ganges!!!!!

// This code is a aprt of test 3 I mentioned in my pink book to do downlink every second while doing uplink

#include <avr/io.h>

#include <stdio.h>// for sprintf()

#include <avr/interrupt.h>

#include <util/delay.h>

#include <math.h>


#include "rfm12.h"

#include "basestation_comms.h"


int main(void)

{

 // Initialise rfm12, adc, uart and interrupts

 rfm12_init();

 adc_init();

 init_uart1();

 sei();


```

```
#if UPLINK_TEST
 uint16_t testdata;
 testdata = 0;
#endif



// ADC channels
uint16_t thrust, yaw, pitch, roll;

// telemetry values
uint8_t packettype;
uint16_t data;
// Button for cargo
DDRA &= ~_BV(PA6);// enable input
PORTA |= _BV(PA6);// enable pull ups
DDRD |= _BV(PD6);// SETto output
PORTD |= _BV(PD6);// SET TO HIGH
// button states
uint8_t button_present, button_previous;
button_previous = 0;

char ch[40];
while (1)
{

 //rfm12_tick();

 #if UPLINK_TEST
  Send_data(OP_ROLL, testdata);
  //sprintf(ch, "\n\rTestdata = %d", testdata);
  //send_string(ch);

  testdata++;

  if (testdata == 1023) break;

  _delay_ms(2);
  Send_data(OP_ROLL, testdata);
  _delay_ms(2);

 #endif

 #if ENABLE_CONTROLS

  // check the rx buffer
  if (rfm12_rx_status() == STATUS_COMPLETE)
  {
   packettype = rfm12_rx_type();
   data = *rfm12_rx_buffer();
```

```
    Retrieve_data(&packettype, &data);
    sprintf(ch, "\n\rPacket type: %u and Data: %u", packettype, data);
    send_string(ch);
    rfm12_rx_clear();
   }
   //else
    //send_string("\n\r No telemetry :(");

   thrust = adc_read(PIN_THRUST);

   if (thrust < 1020)
   {
    Send_data(OP_THRUST, thrust);
    _delay_ms(2);
    Send_data(OP_THRUST, thrust);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!
   }

   yaw = adc_read(PIN_YAW);


   if (yaw < 1020)
   {
    Send_data(OP_YAW, yaw);
    _delay_ms(2);
    Send_data(OP_YAW, yaw);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!
   }


   pitch = adc_read(PIN_PITCH);

   if (pitch < 1020)
   {
    Send_data(OP_PITCH, pitch);
    _delay_ms(2);
    Send_data(OP_PITCH, pitch);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!
   }

   roll = adc_read(PIN_ROLL);

   if (roll < 1020)
   {
    Send_data(OP_ROLL, roll);
    _delay_ms(2);
```

```
    Send_data(OP_ROLL, roll);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!
   }

   sprintf(ch, "\n\rThrust %u yaw %u pitch %u roll %u", thrust, yaw, pitch, roll);
   send_string(ch);

   button_present = (PINA & _BV(PA6)) ? 1:0;
   //sprintf(ch,"\n\rbutton_present = %u", button_present);
   //send_string(ch);
   if ((button_present == 1) && (button_previous == 0))
   {
    send_string("\n\rCargo hook up!");
    Send_data(OP_BUTTON, BTN_CARGO_HOOK_UP);
    _delay_ms(2);
    Send_data(OP_BUTTON, BTN_CARGO_HOOK_UP);
    _delay_ms(2);
    rfm12_tick();

   }
   else if ((button_present == 0) && (button_previous == 1))
   {
    send_string("\n\rCargo hook down!");
    Send_data(OP_BUTTON, BTN_CARGO_HOOK_DOWN);
    _delay_ms(2);
    Send_data(OP_BUTTON, BTN_CARGO_HOOK_DOWN);
    _delay_ms(2);
    rfm12_tick();
   }
   button_previous = button_present;
  #endif

 }



 // Allows program to stop

 while (1) {};

}
```

### D.5 test5_basestation_comms.c

```
// This code is now going to involve taking the copy of basestation_comms.c and adding the adc bits from base_station_controller.c

//  date finished : 8/3/17

```

```
// Arthur: Mohammed Ibrahim and Joel

// D4-Ganges!!!!!

// This code can:

//- send adc values and cargo on/off during flight mode (switch is on flight mode)
// - send each k type values for each channel as the user enters them(switch is on PID/ UI mode)
// UI is more robust, can re-enter enter details again (starting again from entering yaw/pitch/roll)
// Extra feature but might potentially be an important feature to increase data rate:
// During flight mode, send adc values only when there is a change in the channels to hopefully improve datarate (ACHIEVED!!!!!!!!!!!!!!!)

// Same as test4 but aiming to add these features :
// Important feature: get inputs from two more swithces and transmit them(ONE OF THEM DONE)
// do Telemetry interpretation ie do case statement to see what the received data represents: - 3 angle values and voltage of the(DONE!!)


#include <avr/io.h>
#include <stdio.h>// for sprintf()
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>
#include <stdlib.h>

#include "rfm12.h"
#include "basestation_comms.h"

//***************
char k[CHAR_MAX];
volatile uint8_t counter = 0;// itss uint8_t for size saving
volatile uint8_t pid_enter_check = 0; // used to ensure number(eg: 10.230) is not passed to k[] when asked to enter p or i or d
volatile uint8_t tyh_enter_check = 0;
volatile char pid, tyh, pid_copy, tyh_copy;
char buff[10];
ISR(USART0_RX_vect)
{
 char temp[CHAR_MAX];
 temp[counter] = UDR0;
 // if user entrs t/y or h when asked to enter p or i or d then skip this if statement
 // and go to the 5th if statement
 //          |
 //          V
 if ( (pid_enter_check == 0) && (temp[counter] == 'r' || temp[counter] == 'y' || temp[counter] == 'h'))
```

```
{
 tyh_enter_check = 0;// disbale checker as the user has entered thrust or yaw or pitch as requested
 //send_string("\n\r FLAGGG");
 tyh = temp[counter];
 uart_transmit(tyh);
}
else if ((tyh_enter_check == 1) && !(temp[counter] == 'r' || temp[counter] == 'y' || temp[counter] == 'h'))//(temp[counter] > 47 || temp[counter] < 58)
{
 uart_transmit(temp[counter]);
 send_string("\n\r Roll(r) or Yaw(y) or Pitch(h) pls!: ");
 counter = 0;// ensure buffer isnt filled with entered numbers by re-setting this counter
}
else if (temp[counter] == 'p' || temp[counter] == 'i' || temp[counter] == 'd')
{
 pid_enter_check = 0;// disbale checker as the user has entered p or i or d as requested
 pid = temp[counter];
 uart_transmit(pid);// print it to see what we enter
}
else if (temp[counter] == 'x')
{
 counter = 0;//reset the counter to zero so that u can start re-writing to the k[] array
 send_string("\n\r Re-enter roll(r)/yaw(y)/pitch(h) : ");
 // reset the entry checkers to reset the entry of the details
 tyh_enter_check = 1;// to get to the 2nd if condition

}
else if ((pid_enter_check == 1) && !(temp[counter] == 'p' || temp[counter] == 'i' || temp[counter] == 'd'))//(temp[counter] > 47 || temp[counter] < 58)*/ )// incase u entered value when asked to enter p or i or d
{
 uart_transmit(temp[counter]);
 send_string("\n\r Pls p or i or d!: ");
 counter = 0;// ensure buffer isnt filled with entered numbers by re-setting this counter
}
else // if numbers or dots were entered
{
 k[counter] = temp[counter];// can store digits or dots and incremet counter
 uart_transmit(k[counter]);// print it to see what we enter
 counter++;// incremet to get next digit
}


}
//********************************************** from PID_control_test_3

int main(void)

```

```
{

 // Initialise rfm12, adc, uart and interrupts

 rfm12_init();

 adc_init();

 init_uart1();

 sei();
 //************************************
 #if ENABLE_UI
 char ui[60];
 float f, f_temp;
 uint16_t ten_bit;
 send_string("\n\rEnter roll(r)/yaw(y)/pitch(h): ");
 tyh_enter_check = 1;
 //uint16_t adc_value;
 //uint8_t pwm_value;
 #endif
 //****************************************
 #if UPLINK_TEST

  uint16_t testdata;

  testdata = 0;

 #endif



 // ADC channels
    #if ENABLE_CONTROLS
 uint16_t thrust_previous, yaw_previous, pitch_previous, roll_previous;
 thrust_previous = 0;
 yaw_previous = 0;
 pitch_previous = 0;
 roll_previous = 0;
 uint16_t thrust_present, yaw_present, pitch_present, roll_present;

 // telemetry values
 uint8_t packettype;
 uint16_t data;
 // Button for cargo
 DDRA &= ~_BV(PA6);// enable input
 PORTA |= _BV(PA6);// enable pull ups
 // flight/pid siwtch
 DDRA &= ~_BV(PA7);// enable input
 PORTA |= _BV(PA7);// enable pull ups
 // kill switch
 DDRA &= ~_BV(PA4);// enable input
 PORTA |= _BV(PA4);// enable pull ups
```

```
DDRD |= _BV(PD6);// SETto output
PORTD |= _BV(PD6);// SET TO HIGH
// DDRD |= _BV(PD7);// SETto output
// PORTD |= _BV(PD7);// SET TO HIGH
// button states
uint8_t button_present, button_previous;
uint8_t flight_present, flight_previous;
uint8_t kill_present, kill_previous;
flight_previous = 0;
button_previous = 0;
kill_previous = 0;
char ch[40], telemetry[40];
float angle, voltage;
#endif
Send_data(OP_BUTTON, flight_present = (PINA & _BV(PA7))? BTN_FLIGHT_MODE:
    BTN_PID_MODE);// sending an initial mode to the receiver
while (1)

{

 flight_present = (PINA & _BV(PA7))? 1:0;

 if ((flight_present == 1) && (flight_previous == 0))
 {
  Send_data(OP_BUTTON,BTN_FLIGHT_MODE);
  _delay_ms(2);
  Send_data(OP_BUTTON,BTN_FLIGHT_MODE);
  _delay_ms(2);
  rfm12_tick();
 }
 else if ((flight_present == 0) && (flight_previous == 1))
 {
  // we are having a servio as way of checking if this code doesnt block the
      flow of this while loop significantly
  Send_data(OP_BUTTON,BTN_PID_MODE);
  _delay_ms(2);
  Send_data(OP_BUTTON,BTN_PID_MODE);
  _delay_ms(2);
  rfm12_tick();
  send_string("\n\rEnter roll(r)/yaw(y)/pitch(h): ");
 }
 flight_previous = flight_present;
  // check the rx buffer

 if (flight_present)
 {

  if (rfm12_rx_status() == STATUS_COMPLETE)
  {
   packettype = rfm12_rx_type();
   data = *rfm12_rx_buffer();
```

```
    Retrieve_data(&packettype, &data);
    //sprintf(ch, "\n\rPacket type: %u and Data: %u", packettype, data);
    //send_string(ch);
    rfm12_rx_clear();
    switch(packettype)
    {
     case OP_ANGLE_X : // angle needs to be converted from its decimal value to float
        //angle = ;
        sprintf(telemetry,"Gyro angle x: %f", angle);
        send_string(telemetry);
       break;
     case OP_ANGLE_Y: // angle needs to be converted from its decimal value to float
        //angle = ;
        sprintf(telemetry,"Gyro angle y: %f", angle);
        send_string(telemetry);
       break;
     case OP_ANGLE_Z: // angle needs to be converted from its decimal value to float
        //angle = ;
        sprintf(telemetry,"Gyro angle z: %f", angle);
        send_string(telemetry);
       break;
     case OP_BATTERY_LEVEL:  // angle needs to be converted from its decimal value to float
        voltage = (float)(3.3/1024)*data;// r1 = 100k and r2 is 33k
        voltage = (float)(133/33)*voltage;
        sprintf(telemetry,"\n\rBattery voltage: %f", voltage);
        send_string(telemetry);
       break;
     case OP_IR_SENSOR:
      voltage = to_distance(data);
      sprintf(telemetry, "\n\rIR sensor: %f", voltage);
      send_string(telemetry);
      break;
    }
   }
   //else
    //send_string("\n\r No telemetry :(");

   thrust_present = (kill_present) ? 512 : adc_read(PIN_THRUST);
   if (thrust_present != thrust_previous && thrust_present < 1020)
   {
    Send_data(OP_THRUST, thrust_present);
    _delay_ms(2);
    Send_data(OP_THRUST, thrust_present);
    _delay_ms(2);
    rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!!
    thrust_previous = thrust_present;
   }
```

```
245
246     yaw_present = (kill_present) ? 512 : adc_read(PIN_YAW);
247     if (yaw_present != yaw_previous && yaw_present < 1020)
248     {
249      Send_data(OP_YAW, yaw_present);
250      _delay_ms(2);
251      Send_data(OP_YAW, yaw_present);
252      _delay_ms(2);
253      rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE
         SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!
254      yaw_previous = yaw_present;
255     }
256
257     pitch_present = (kill_present) ? 512 : adc_read(PIN_PITCH);
258
259     if (pitch_present != pitch_previous && pitch_present < 1020)
260     {
261      Send_data(OP_PITCH, pitch_present);
262      _delay_ms(2);
263      Send_data(OP_PITCH, pitch_present);
264      _delay_ms(2);
265      rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE
         SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!
266      pitch_previous = pitch_present;
267     }
268
269     roll_present = (kill_present) ? 512 : adc_read(PIN_ROLL);
270
271     if (roll_present != roll_previous && roll_present < 1020)
272     {
273      Send_data(OP_ROLL, roll_present);
274      _delay_ms(2);
275      Send_data(OP_ROLL, roll_present);
276      _delay_ms(2);
277      rfm12_tick();// make sure you 'TICK' EVERY TIME YOU SEND_DATA! (IN OUR CASE
         SEND TWICE)!!!!!!!!!!!!!!!!!!!!!!!!!
278      roll_previous = roll_present;
279     }
280     button_present = (PINA & _BV(PA6)) ? 1:0;
281     //sprintf(ch,"\n\rbutton_present = %u", button_present);
282     //send_string(ch);
283     if ((button_present == 1) && (button_previous == 0))
284     {
285      send_string("\n\rCargo hook up!");
286      Send_data(OP_BUTTON, BTN_CARGO_HOOK_UP);
287      _delay_ms(2);
288      Send_data(OP_BUTTON, BTN_CARGO_HOOK_UP);
289      _delay_ms(2);
290      rfm12_tick();
291
292     }
293     else if ((button_present == 0) && (button_previous == 1))
294     {
```

```
      send_string("\n\rCargo hook down!");
      Send_data(OP_BUTTON, BTN_CARGO_HOOK_DOWN);
      _delay_ms(2);
      Send_data(OP_BUTTON, BTN_CARGO_HOOK_DOWN);
      _delay_ms(2);
      rfm12_tick();
     }
     button_previous = button_present;

     kill_present = (PINA & _BV(PA4)) ? 1:0;
     if ((kill_present == 1) && (kill_previous == 0))
     {
      send_string("\n\rKill switch ON!");
      Send_data(OP_BUTTON,BTN_POWER_OFF);
      _delay_ms(2);
      Send_data(OP_BUTTON,BTN_POWER_OFF);
      _delay_ms(2);
      rfm12_tick();
     }
     else if ((kill_present == 0) && (kill_previous == 1))
     {
      send_string("\n\rKill switch OFF!");
      Send_data(OP_BUTTON, BTN_POWER_ON);
      _delay_ms(2);
      Send_data(OP_BUTTON, BTN_POWER_ON);
      _delay_ms(2);
      rfm12_tick();
     }
     kill_previous = kill_present;


     // DO THE 2 MORE BUTTON FEATURES


    }
    else if (!flight_present)
    {
     if (tyh == 'r' || tyh == 'y' || tyh == 'h')
     {
      send_string("\n\r Enter a K type (p or i or d): ");
      pid_enter_check = 1;// always set this variable to 1 when asked to enter p or
          i or d
      tyh_copy = tyh;// we need a copy of tyh to display what we have to the user
      tyh = NULL;// definitely clear this as this condition will keep getting
          satisfied!

     }

     else if (pid == 'p' || pid == 'i' || pid == 'd')
      {
       send_string("\n\r Enter k value of this type: ");
       pid_copy = pid;
```

```
      pid = NULL;// definitely clear this as this condition will keep getting satisfied!
    }

   else if (counter == CHAR_MAX)// char bufer is ready to transmit
   {
    counter = 0;// back to zero
    f = atof(k);// convert it to float
    f_temp = f*100; //
    if (f_temp > 1023)// check if number not withing range
    {
     send_string("\n\r Error: k value is not in range!");
     send_string("\n\r Enter a K type (p or i or d): ");
     continue;
    }
    ten_bit = (uint16_t) (f_temp+0.5);
    sprintf(ui,"\n\r 10-bit dec value for k type: %c of %c: %d",pid_copy,tyh_copy,ten_bit);
    send_string(ui);
    // transmit the data for pid_copy,tyh_copy and ten_bi
    switch(tyh_copy)
    {
     case 'r':   if (pid_copy == 'p')
         {
          Send_data(OP_KP_ROLL,ten_bit);
          _delay_ms(2);
          // Send_data(OP_KP_ROLL,ten_bit);
          // _delay_ms(2);
          rfm12_tick();
         }
        else if (pid_copy == 'i')
        {
         Send_data(OP_KI_ROLL,ten_bit);
         _delay_ms(2);
         // Send_data(OP_KI_ROLL,ten_bit);
         // _delay_ms(2);
         rfm12_tick();
        }
        else if (pid_copy == 'd')
        {
         Send_data(OP_KD_ROLL,ten_bit);
         _delay_ms(2);
         // Send_data(OP_KD_ROLL,ten_bit);
         // _delay_ms(2);
         rfm12_tick();
        }
        break;
     case 'y':   if (pid_copy == 'p')
         {
          Send_data(OP_KP_YAW,ten_bit);
          _delay_ms(2);
          // Send_data(OP_KP_YAW,ten_bit);
          // _delay_ms(2);
```

```
            rfm12_tick();
           }
          else if (pid_copy == 'i')
          {
           Send_data(OP_KI_YAW, ten_bit);
           _delay_ms(2);
           // Send_data(OP_KI_YAW, ten_bit);
           // _delay_ms(2);
           rfm12_tick();
          }
          else if (pid_copy == 'd')
          {
           Send_data(OP_KD_YAW, ten_bit);
           _delay_ms(2);
           // Send_data(OP_KD_YAW, ten_bit);
           // _delay_ms(2);
           rfm12_tick();
          }
          break;
      case 'h':    if (pid_copy == 'p')
           {
            Send_data(OP_KP_PITCH, ten_bit);
            _delay_ms(2);
            // Send_data(OP_KP_PITCH, ten_bit);
            // _delay_ms(2);
            rfm12_tick();
           }
          else if (pid_copy == 'i')
          {
           send_string("\n\rFLAG!");
           Send_data(OP_KI_PITCH, ten_bit);
           _delay_ms(2);
           // Send_data(OP_KI_PITCH, ten_bit);
           // _delay_ms(2);
           rfm12_tick();
          }
          else if (pid_copy == 'd')
          {
           Send_data(OP_KD_PITCH, ten_bit);
           _delay_ms(2);
           // Send_data(OP_KD_PITCH, ten_bit);
           // _delay_ms(2);
           rfm12_tick();
          }
          break;
     }
     // clear pid_copy and tyh type sp that they dont transmitted again
     pid_copy = NULL;
     tyh_copy = NULL;
     send_string("\n\rEnter roll(r)/yaw(y)/pitch(h): ");
     tyh_enter_check = 1;
     counter = 0;// to stop receivng these datas event houghtb i havnt entered
         anything
```

```
    }
    else
     continue;
   }


  }



  // Allows program to stop

  while (1) {};

}
```

### D.6 Uplink/drone_comms.c

```
/*
 drone_comms.c
 Code to implement communications from ground to air on the drone.
 Author: Joel Trickey
*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>

#include "rfm12.h"
#include "drone_comms.h"

int main(void)
{
 // Initialise rfm12 and interrupts
 rfm12_init();
 sei();

 uint8_t receivedpacketype;
 uint16_t receiveddata;

 while (1)
 {
  rfm12_tick();

  // Wait for data to be fully received
  if (rfm12_rx_status() == STATUS_COMPLETE)
  {
   // Get the received packet type and data
   receivedpacketype = rfm12_rx_type();
   receiveddata = *rfm12_rx_buffer();

```

```
    // Decrypt (if enabled) and extract 10-bit data and packet type from the received packet
    Retrieve_data(&receivedpacketype, &receiveddata);

    #if UPLINK_TEST
      // Send data to UART
    #endif
   }
  }
}

void Retrieve_data(uint8_t* type, uint16_t* data)
{
  // Combine packet type and data into a single 16-bit int
  uint16_t totalpacket;
  totalpacket = *type;
  totalpacket = (totalpacket << DATA_BIT_SIZE) + *data;

  #if ENCRYPTION_ENABLED
   // Decrypt the received packet
   totalpacket = Decrypt_data(totalpacket);
  #endif // ENCRYPTION_ENABLED

  // Split the decrypted packet into the data and the packet type
  Decode_data(type, data, totalpacket);
}

void Decode_data(uint8_t* type, uint16_t* data, uint16_t totalpacket)
{
  // Get 10-bit data from the 16 bit packet
  *data = totalpacket & (uint16_t)1023;

  // Get packet type
  *type = (totalpacket >> DATA_BIT_SIZE);
}

uint16_t Decrypt_data(uint16_t packet)
{
  // Retrieve the encryption key
  uint8_t encryption_key;
  encryption_key = (packet >> (DATA_BIT_SIZE + COMMAND_BIT_SIZE));

  // Retrieve bits that are shifted out when the left shift is done
  uint8_t rotated_out_bits;
  rotated_out_bits = (packet >> (DATA_BIT_SIZE + COMMAND_BIT_SIZE - encryption_key));

  // Get completely rotated bits by adding the shifted out bits to the
  // original packet left-shifted by the required number of bits.
  // It is & with a sequence of 1s to remove the encryption key from the overall packet
  uint16_t decrypted_packet;
```

```
 decrypted_packet = ((packet << encryption_key) & ((uint16_t)pow(2, DATA_BIT_SIZE
     + COMMAND_BIT_SIZE) - 1)) + rotated_out_bits;

 return decrypted_packet;
}
```

### D.7 basestation_comms.c

```
/*
 basestation_comms.c
 Implements communications from ground to air on the basestation.
 Author: Joel Trickey
*/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>

#include "rfm12.h"
#include "basestation_comms.h"

volatile uint8_t encryption_key;

int main(void)
{
 // Initialise rfm12 and interrupts
 rfm12_init();
 sei();

 encryption_key = 5;

 // Send test data
 uint16_t testdata = 0;

 while (1)
 {
  rfm12_tick();

  #if UPLINK_TEST
   Send_data(OP_ROLL, testdata);
   testdata++;
   if (testdata == 1024) break;
   _delay_ms(1000);
  #endif

 }
 while (1) {};
}

//!
/*
 Process data and send it to the transceiver for transmission.
```

```
 If encryption is enabled in the basestation_comms.h then the data will be encrypted.
 The 10-bit data is encoded such that the 2 MSBs are stored in the packet type.
*/
void Send_data(uint8_t type, uint16_t data)
{
 // Combine packet type and data into a single 16-bit int
 uint16_t totalpacket;
 totalpacket = type;
 totalpacket = (totalpacket << DATA_BIT_SIZE) + data;

 // Encrypt data
 #if ENCRYPTION_ENABLED
  totalpacket = Encrypt_data(totalpacket);
 #endif

 // Split 16-bit packet into two 8-bit ints - packet type and data
 uint8_t datapacket;
 Encode_data(&type, &datapacket, totalpacket);

 // Send packet to the buffer for transmission
 rfm12_tx(sizeof(datapacket), type, &datapacket);
}

//!
/*
 Encode the total packet into the type and data
*/
void Encode_data(uint8_t* type, uint8_t* data, uint16_t totalpacket)
{
 // Data is equal to the 8 LSBs
 *data = totalpacket;

 // Type, encryption key and 2 bits of data are held in the 8 MSBs
 *type = (totalpacket >> 8);
}

//!
/*
 Encrypt the packet type and data using an encryption key.
 This encryption key changes every time the data is encrypted.
*/
uint16_t Encrypt_data(uint16_t packet)
{
 // Retrieve bits that are shifted out when the right shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet & ((uint8_t) pow(2, encryption_key) - 1));

 // Get completely rotated bits by adding the shifted out bits to the
 // original packet right-shifted by the required number of bits.
 uint16_t encrypted_packet;
 encrypted_packet = (packet >> encryption_key) + (rotated_out_bits << (COMMAND_BIT_SIZE + DATA_BIT_SIZE - encryption_key));
```

```

 // Add on the encryption key to the MSBs of the packet
 encrypted_packet = encrypted_packet + (encryption_key << (COMMAND_BIT_SIZE + DATA_BIT_SIZE));

 // Adjust encryption key for next transmission
 encryption_key = (encryption_key < 3) ? encryption_key + 5 : encryption_key - 3;
 if (encryption_key == 0) encryption_key = 5;

 return encrypted_packet;
}
```

### D.8 encode_test.c

```
/*
 encode_test.c
 Tests the encoding and encryption of
 Author: Joel Trickey and Mohammed Ibrahim
*/

#include <avr/io.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "../comms.h"

// Mohammed's UART code
#define BAUD 57600                                  // define baud
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)            // set baud rate value for UBRR

void init_uart1()// initialize UART
{
 //1. set the baud rate, lets configure to 9600;
 // set the baud rate registers Ref: [1],[2]
 UBRR0H = BAUDRATE >> 8;// UBRRnH is 8 bits left
 UBRR0L = BAUDRATE;

 //2. setting up data packet: 8 bits ,no parity 1 stop bit
 // setting 8 bits got to UCSCR register Ref:[3], pg 185 of data sheet

 UCSR0C = _BV(UCSZ00) | _BV(UCSZ01); // 8 bits, USBS1 = 0 for 1 stop bit

          // note: havnt set up the stop bit in Ref [2] slides
          // 3. from Ref[2] we now enable Transmission and receive n UCSRnB register
 UCSR0B = _BV(TXEN0) | _BV(RXEN0);
}

void uart_transmit(char data)
{
```

```
 while (!(UCSR0A & _BV(UDRE0))); //  data register enable bit is 1 if tx buffer is empy
          // if its 1 we load data onto UDR- Uart Data Register(buffer)
 UDR0 = data;
}

void send_string(char *str)
{
 int i;
 for (i = 0; str[i]; i++) uart_transmit(str[i]);
}
//

// Joel's encoding and encryption test
uint8_t test_encode(uint16_t totalpacket)
{
 // Encode
 //send_string("Encoding data\n");

 uint8_t encodeddata, encodedpackettype;
 encodeddata = totalpacket;
 encodedpackettype = (totalpacket >> 8);

 uint16_t totalencodedpacket;
 totalencodedpacket = encodeddata + (encodedpackettype << 8);

 // Decode
 //send_string("Decoding data\n");

 uint16_t decodeddata;
 uint8_t decodedpackettype;

 decodeddata = totalencodedpacket & (uint16_t)1023;
 decodedpackettype = (totalencodedpacket >> DATA_BIT_SIZE);

 uint16_t totaldecodedpacket;
 totaldecodedpacket = encodeddata + (decodedpackettype << 8);

 //char sendData[30];
 //sprintf(sendData, "Decoded data: %d\n", decodeddata);
 //send_string(sendData);

 return (encodeddata == decodeddata && encodedpackettype == decodedpackettype) ? 1 : 0;
}

// Original encryption/decryption code
uint16_t test_decrypt(uint16_t packet)
{
 // Retrieve the encryption key
 uint8_t encryption_key;
 encryption_key = (packet >> (DATA_BIT_SIZE + COMMAND_BIT_SIZE));

```

```
 char sendData0[30];
 sprintf(sendData0, "Encryption key: %u\r\n", encryption_key);
 send_string(sendData0);

 // Remove the encryption key from the packet
 packet = (packet & ((uint16_t)pow(2, DATA_BIT_SIZE + COMMAND_BIT_SIZE) - 1));

 // Retrieve bits that are shifted out when the left shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> (DATA_BIT_SIZE + COMMAND_BIT_SIZE - encryption_key
     ));

 // Get completely rotated bits by adding the shifted out bits to the
 // original packet left-shifted by the required number of bits.
 // It is & with a sequence of 1s to remove the encryption key from the overall
     packet
 uint16_t decrypted_packet;
 decrypted_packet = ((packet << encryption_key) & ((uint16_t)pow(2, DATA_BIT_SIZE
     + COMMAND_BIT_SIZE) - 1)) + rotated_out_bits;

 char sendData2[30];
 sprintf(sendData2, "Decrypted: %u\r\n", decrypted_packet);
 send_string(sendData2);

 return decrypted_packet;
}

uint16_t test_encrypt(uint16_t packet)
{
 //uint16_t packet;
 //packet = 343 + (6 << DATA_BIT_SIZE);

 char sendData1[50];
 sprintf(sendData1, "\r\nEncrypting %u\r\n", packet);
 send_string(sendData1);

 uint8_t encrypt_key;
 encrypt_key = 4;

 //send_string("Encrypting 868 with a packet of 2 by 2 bits\r\n");

 // Retrieve bits that are shifted out when the right shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet & ((uint8_t)pow(2, encrypt_key) - 1));

 // Get completely rotated bits by adding the shifted out bits to the original
     packet right-shifted by the required number of bits.
 uint16_t encrypted_packet = (packet >> encrypt_key) + (rotated_out_bits << (
     COMMAND_BIT_SIZE + DATA_BIT_SIZE - encrypt_key));

 // Add on the encryption key to the MSBs of the packet
 encrypted_packet = encrypted_packet + (encrypt_key << (COMMAND_BIT_SIZE +
     DATA_BIT_SIZE));
```

```
 char sendData2[30];
 sprintf(sendData2, "Encrypted: %u\r\n", encrypted_packet);
 send_string(sendData2);

 return encrypted_packet;
}

// Updated encryption/decryption code
uint16_t get_1s(uint8_t num)
{
 // Generate a variable that is num 1's
 uint16_t out;
 out = 1;

 uint8_t i;
 for (i = 1; i < num; i++)
 {
  out = (out << 1);
  out++;
 }

 return out;
}

uint16_t Encrypt_data(uint16_t packet)
{
 // Get the encryption key from the first 3 bits of the packet
 uint8_t encryption_key;
 encryption_key = (packet & 7);

 // Retrieve bits that are shifted out when the right shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> 3) & get_1s(encryption_key);

 // Right shift packet and add on rotated-out bits in their new position
 uint16_t encrypted_packet;
 encrypted_packet = (packet >> encryption_key) + (rotated_out_bits << (13 + 3 -
     encryption_key));

 // Remove any values from the first 3 bits and add on the encryption key
 encrypted_packet = (encrypted_packet & 65528) + encryption_key;

 return encrypted_packet;
}

uint16_t Decrypt_data(uint16_t packet)
{
 // Get the encryption key from the first 3 bits of the packet
 uint8_t encryption_key;
 encryption_key = (packet & 7);

 // Remove the encryption key from the packet
```

```
 packet = (packet & 65528);

 // Retrieve bits that are shifted out when the left shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> (16 - encryption_key)) & get_1s(encryption_key);

 // Left shift packet and add on rotated-out bits in their previous position
 uint16_t decrypted_packet;
 decrypted_packet = (packet << encryption_key) + (rotated_out_bits << 3);

 // Remove any values from the first 3 bits and add on the encryption key
 decrypted_packet = (decrypted_packet & 65528) + encryption_key;

 return decrypted_packet;
}

int main(void)
{
 init_uart1();

 char ch[100];
 // Test all possibilities of data for the chosen encryption key and break if an
    encryption fails
 uint16_t testpacket, result;
 for (testpacket = 0; testpacket < 65536; testpacket++)
 {
  result = Encrypt_data(testpacket);
  if (Decrypt_data(result) != testpacket)
  {
   sprintf(ch, "\n\rError! - Encrypted %u as %u but decrypted as %u", testpacket,
       result, Decrypt_data(result));
   send_string(ch);
   break;
  }
  else
  {
   sprintf(ch, "\n\rSuccessfully encrypted and decrypted %u", testpacket);
   send_string(ch);
  }
 }

 while (1) {};

 // Test encoding
 /*
 for (testpacket = 0; testpacket < 65536; testpacket++)
 {
  if (test_encode(testpacket) == 0)
  {
   sprintf(ch, "Error encoding %u", testpacket);
   send_string(ch);
   break;
  }
```

```
  else
  {
   sprintf(ch, "\n\rSuccessfully encoded and decoded %u", testpacket);
   send_string(ch);
  }
 }

 while (1) {};*/
}
```

### D.9 comms.h

```
// Define codes for the different commands
#define OP_THRUST 0
#define OP_ROLL 1
#define OP_YAW 2
#define OP_PITCH 3
#define OP_BUTTON 4

// K values sent to the controller
#define OP_KP_ROLL 5
#define OP_KI_ROLL 6
#define OP_KD_ROLL 7
#define OP_KP_YAW 8
#define OP_KI_YAW 9
#define OP_KD_YAW 10
#define OP_KP_PITCH 11
#define OP_KI_PITCH 12
#define OP_KD_PITCH 13

// Telemetery sent back from the drone
#define OP_BATTERY_LEVEL 14
#define OP_ANGLE_X 15
#define OP_ANGLE_Y 16
#define OP_ANGLE_Z 17
#define OP_TELEMETRY_ACKNOWLEDGE 18
#define OP_IR_SENSOR 19
#define OP_NULL 63

// K value size
#define CHAR_MAX 6

// Define data code for the button command
#define BTN_CARGO_HOOK_UP 0
#define BTN_CARGO_HOOK_DOWN 1
#define BTN_PID_MODE 2 // Allow setting of K values and disable telemetry
#define BTN_FLIGHT_MODE 3 // Receive telemetry
#define BTN_POWER_ON 4
#define BTN_POWER_OFF 5

// ADC pin definitions
//  Base-side
```

```
#define PIN_THRUST 0
#define PIN_ROLL 1
#define PIN_YAW 2
#define PIN_PITCH 3
//  Drone-side
#define PIN_BATTERY 0
#define PIN_IR_SENSOR 1

// Define bit sizes for the packets
#define DATA_BIT_SIZE 10
#define COMMAND_BIT_SIZE 5
#define ENCRYPT_KEY_BIT_SIZE 1

// Enable telemetry back to the base
#define ENABLE_TELEMETRY 1

// Enable the cargo hook
#define ENABLE_CARGO_HOOK 1

// Enable encryption
#define ENABLE_ENCRYPTION 1

// Enable/configure UI and setting K values
#define ENABLE_UI 1
#define K_RANGE_MIN 0
#define K_RANGE_MAX 10.23

// Enable controls
#define ENABLE_CONTROLS 1

// Enable isolated uplink/downlink tests
#define UPLINK_TEST 0
#define DOWNLINK_TEST 0
```

### D.10 Combined_interface_comms/drone_comms.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>

#include "rfm12.h"
#include "drone_comms.h"

void Retrieve_data(uint8_t* type, uint16_t* data)
{
 // Combine packet type and data into a single 16-bit int
 uint16_t totalpacket;
 totalpacket = *type;
 totalpacket = (totalpacket << 8) + *data;
 #if ENABLE_ENCRYPTION
  // Decrypt the received packet
```

```
  totalpacket = Decrypt_data(totalpacket);
 #endif

 // Split the decrypted packet into the data and the packet type
 Decode_data(type, data, totalpacket);
}

void Decode_data(uint8_t* type, uint16_t* data, uint16_t totalpacket)
{
 // Get 10-bit data from the 16 bit packet
 *data = totalpacket & (uint16_t)1023;

 // Get packet type
 *type = (totalpacket >> DATA_BIT_SIZE);
}

void Send_data(uint8_t type, uint16_t data)
{
#if ENABLE_UPLINK
 // Combine packet type and data into a single 16-bit int
 uint16_t totalpacket;
 totalpacket = type;
 totalpacket = (totalpacket << DATA_BIT_SIZE) + data;

 // Encrypt data
#if ENABLE_ENCRYPTION
 totalpacket = Encrypt_data(totalpacket);
#endif
 // Split 16-bit packet into two 8-bit ints - packet type and data
 uint8_t datapacket;
 Encode_data(&type, &datapacket, totalpacket);

 //char ch[100];
 //sprintf(ch, "\n\rSending: %u %u", type, datapacket);
 //send_string(ch);
 // Send packet to the buffer for transmission
 rfm12_tx(sizeof(datapacket), type, &datapacket);
#endif

}

void Encode_data(uint8_t* type, uint8_t* data, uint16_t totalpacket)
{
 // Data is equal to the 8 LSBs
 *data = totalpacket;

 // Type, encryption key and 2 bits of data are held in the 8 MSBs
 *type = (totalpacket >> 8);
}

#if ENABLE_ENCRYPTION
uint16_t Encrypt_data(uint16_t packet)
{
```

```
 // Get the encryption key from the first 3 bits of the packet
 uint8_t encryption_key;
 encryption_key = (packet & 7);

 // Retrieve bits that are shifted out when the right shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> 3) & get_1s(encryption_key);

 // Right shift packet and add on rotated-out bits in their new position
 uint16_t encrypted_packet;
 encrypted_packet = (packet >> encryption_key) + (rotated_out_bits << (13 + 3 -
     encryption_key));

 // Remove any values from the first 3 bits and add on the encryption key
 encrypted_packet = (encrypted_packet & 65528) + encryption_key;

 return encrypted_packet;
}

uint16_t Decrypt_data(uint16_t packet)
{
 // Get the encryption key from the first 3 bits of the packet
 uint8_t encryption_key;
 encryption_key = (packet & 7);

 // Remove the encryption key from the packet
 packet = (packet & 65528);

 // Retrieve bits that are shifted out when the left shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> (16 - encryption_key)) & get_1s(encryption_key);

 // Left shift packet and add on rotated-out bits in their previous position
 uint16_t decrypted_packet;
 decrypted_packet = (packet << encryption_key) + (rotated_out_bits << 3);

 // Remove any values from the first 3 bits and add on the encryption key
 decrypted_packet = (decrypted_packet & 65528) + encryption_key;

 return decrypted_packet;
}

uint16_t get_1s(uint8_t num)
{
 uint16_t out;
 out = 1;

 uint8_t i;
 for (i = 1; i < num; i++)
 {
  out = (out << 1);
  out++;
 }
```

```

 return out;
}
#endif

void init_uart1()// initialize UART
{
  //1. set the baud rate, lets configure to 9600;
 // set the baud rate registers Ref: [1],[2]
 UBRR0H = BAUDRATE >> 8;// UBRRnH is 8 bits left
 UBRR0L = BAUDRATE;

  //2. setting up data packet: 8 bits ,no parity 1 stop bit
  // setting 8 bits got to UCSCR register Ref:[3], pg 185 of data sheet

 UCSR0C = _BV(UCSZ00) | _BV(UCSZ01); // 8 bits, USBS1 = 0 for 1 stop bit

  // note: havnt set up the stop bit in Ref [2] slides
 // 3. from Ref[2] we now enable Transmission and receive n UCSRnB register
 UCSR0B = _BV(TXEN0) | _BV(RXEN0);

}
// transmit data function
void uart_transmit( char data)
{
 while(!( UCSR0A & _BV(UDRE0) ) ); //  data register enable bit is 1 if tx
     buffer is empy
 // if its 1 we load data onto UDR- Uart Data Register(buffer)
 UDR0 = data;
}

void send_string(char *str)
{
 int i;
 for( i = 0; str[i]; i++) uart_transmit(str[i]);
}

void init_timer()
{
 TCCR1B |= _BV(CS12) | _BV(CS10);
 TCNT1 = 0;
}
```

### D.11 test1_drone_comms.c

```
// This code involves taking a copy from drone_comms.c and simply printing off
    the received data off putty
// Arthur: Joel And Mohammed
// Date : 3/1/17
// this code is also for test 1 im mentioned in my pink book
#include <avr/io.h>
#include <avr/interrupt.h>
```

```
#include <util/delay.h>
#include <stdio.h>// for sprintf()
#include <math.h>

#include "rfm12.h"
//#include "rfm12_config.h"
#include "drone_comms.h"

int main(void)
{
 // Initialise rfm12 and interrupts
 init_uart1();
 rfm12_init();
 sei();
 send_string("Initialised");

 uint8_t receivedpacketype;
 uint16_t receiveddata, i;

 uint16_t thrust, roll, pitch, yaw;

 char ch[30];
 while (1)
 {
  rfm12_tick();

  // Wait for data to be fully received
  if (rfm12_rx_status() == STATUS_COMPLETE)
  {
   //i++;
   // Get the received packet type and data
   receivedpacketype = rfm12_rx_type();
   receiveddata = *rfm12_rx_buffer();

   rfm12_rx_clear();
   // Decrypt (if enabled) and extract 10-bit data and packet type from the received packet
   Retrieve_data(&receivedpacketype, &receiveddata);

   //sprintf(ch, "\n\rReceived: %u %u", receivedpacketype, receiveddata);
   //send_string(ch);

#if UPLINK_TEST
   //send_string("\n\rReceived transmission:");
   // Send received packet type to UART
   //sprintf(ch,"\n\r Packet type: %u",receivedpacketype);
   //send_string(ch);
   // Send received data to UART
   //sprintf(ch, "\n\r Data = %u", receiveddata);
   //send_string(ch);

   //if (receiveddata > 1000) break;
#endif
```

```
#if ENABLE_CONTROLS
    switch (receivedpacketype)
    {
    case OP_THRUST:
     thrust = receiveddata;

     sprintf(ch, "\n\r Thrust = %u", thrust);
     send_string(ch);
     break;
    case OP_ROLL:
     roll = receiveddata;

     sprintf(ch, " Roll = %u", roll);
     send_string(ch);
     break;
    case OP_YAW:
     yaw = receiveddata;

     sprintf(ch, " Yaw = %u", yaw);
     send_string(ch);
     break;
    case OP_PITCH:
     pitch = receiveddata;

     sprintf(ch, " Pitch = %u", pitch);
     send_string(ch);
     break;
    }
#endif

   }
  }

  while (1)
  {

  }
}
```

### D.12 test2_drone_comms.c

```
// This code involves taking a copy from drone_comms.c and simply printing off
    the received data off putty
// Arthur: Joel And Mohammed
// Date : 3/1/17
// this code is also for test 2 im mentioned in my pink book (receiving the PID
    values)
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>// for sprintf()
#include "rfm12.h"
```

```c
#include "drone_comms.h"

int main(void)
{
 // Initialise rfm12 and interrupts
 rfm12_init();
 init_uart1();
 sei();
 send_string("\n\rInitialised\n\r");

 uint8_t receivedpacketype, PIDmode;
 uint16_t receiveddata;
 PIDmode = 1;
 char ch[30];
 float k_value;
 while (1)
 {
  rfm12_tick();

  // Wait for data to be fully received
  if (rfm12_rx_status() == STATUS_COMPLETE)
  {
   // Get the received packet type and data
   receivedpacketype = rfm12_rx_type();
   receiveddata = rfm12_rx_buffer();
   rfm12_rx_clear();

   sprintf(ch, "\n\r Raw data: %u %u", receivedpacketype, receiveddata);
   send_string(ch);
   // Decrypt (if enabled) and extract 10-bit data and packet type from the received packet
   Retrieve_data(&receivedpacketype, &receiveddata);
   sprintf(ch, "\n\r Received: %u %u", receivedpacketype, receiveddata);
   send_string(ch);
    #if UPLINK_TEST
     send_string("\n\rReceived transmission:");
     // Send received packet type to UART
     sprintf(ch,"\n\r Packet type: %d",receivedpacketype);
     send_string(ch);
     // Send received data to UART
     sprintf(ch, "\n\r Data = %d", receiveddata);
     send_string(ch);
    #endif

#if ENABLE_CONTROLS
   switch (receivedpacketype)
   {
    case OP_THRUST :
       sprintf(ch, "\n\r Thrust = %u", receiveddata);
       send_string(ch);
       break;
    case OP_ROLL :
       sprintf(ch, "\n\r Roll = %u", receiveddata);
```

```
        send_string(ch);
        break;
     case OP_YAW :
        sprintf(ch, "\n\r Yaw = %u", receiveddata);
        send_string(ch);
        break;
     case OP_PITCH :
        sprintf(ch, "\n\r Pitch = %u", receiveddata);
        send_string(ch);
        break;
     default : send_string("Didnt receive K values");
    }
#endif
#if ENABLE_UI
    // If in PID mode also check for K values
    if (PIDmode)
    {
     switch (receiveddata)
     {
      case OP_KP_ROLL:
       k_value = receiveddata / 100;
       sprintf(ch, "Roll Kp value = %f", k_value);
       send_string(ch);
       break;
      case OP_KI_ROLL:
       k_value = receiveddata / 100;
       sprintf(ch, "Roll Ki value = %f", k_value);
       send_string(ch);
       break;
      case OP_KD_ROLL:
       k_value = receiveddata / 100;
       sprintf(ch, "Roll Kd value = %f", k_value);
       send_string(ch);
       break;
      case OP_KP_YAW:
       k_value = receiveddata / 100;
       sprintf(ch, "Yaw Kp value = %f", k_value);
       send_string(ch);
       break;
      case OP_KI_YAW:
       k_value = receiveddata / 100;
       sprintf(ch, "Yaw Ki value = %f", k_value);
       send_string(ch);
       break;
      case OP_KD_YAW:
       k_value = receiveddata / 100;
       sprintf(ch, "Yaw Kd value = %f", k_value);
       send_string(ch);
       break;
      case OP_KP_PITCH:
       k_value = receiveddata / 100;
       sprintf(ch, "Pitch Kp value = %f", k_value);
       send_string(ch);
```

```
      break;
     case OP_KI_PITCH:
      k_value = receiveddata / 100;
      sprintf(ch, "Pitch Ki value = %f", k_value);
      send_string(ch);
      break;
     case OP_KD_PITCH:
      k_value = receiveddata / 100;
      sprintf(ch, "Pitch Kd value = %f", k_value);
      send_string(ch);
      break;
    }
   }
#endif // ENABLE_UI
  }
 }
}
```

### D.13 test3_drone_comms.c

```
// This code involves taking a copy from drone_comms.c and simply printing off
    the received data off putty
// Arthur: Joel And Mohammed
// Date : 3/1/17
// this code is also for test 1 im mentioned in my pink book
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>// for sprintf()
#include <stdlib.h>
#include <math.h>

#include "rfm12.h"
//#include "rfm12_config.h"
#include "drone_comms.h"

int main(void)
{
 // Initialise rfm12 and interrupts
 init_uart1();
 rfm12_init();
 init_timer();
 sei();
 send_string("Initialised");

#if ENABLE_UI
 uint8_t PIDmode;
 PIDmode = 1;
#endif

 uint8_t receivedpacketype, cargohookup;
 cargohookup = 1;
```

```
32  uint16_t receiveddata, i;
33  i = 0;
34
35  uint16_t thrust, roll, pitch, yaw;
36
37  char ch[30];
38  float k_value;
39  while (1)
40  {
41   rfm12_tick();
42   //rfm12_poll();
43   //receivedpacketype = rfm12_read_fifo_inline();
44   //sprintf(ch, "\n\rFIFO: %u", receivedpacketype);
45   //if (receivedpacketype != 0) send_string(ch);
46   // Wait for data to be fully received
47   if (rfm12_rx_status() == STATUS_COMPLETE)
48   {
49    i++;
50    //sprintf(ch, "i = %u", i);
51    //send_string(ch);
52
53    // Get the received packet type and data
54    receivedpacketype = rfm12_rx_type();
55    receiveddata = *rfm12_rx_buffer();
56    //sprintf(ch, "\n\rReceived: %u %u", receivedpacketype, receiveddata);
57    //send_string(ch);
58    rfm12_rx_clear();
59
60    // Decrypt (if enabled) and extract 10-bit data and packet type from the
        received packet
61    Retrieve_data(&receivedpacketype, &receiveddata);
62
63    //sprintf(ch, "\n\rDecrypted %u %u", receivedpacketype, receiveddata);
64    //send_string(ch);
65
66 #if UPLINK_TEST
67    sprintf(ch, "\n\rReceived: %u %u", receivedpacketype, receiveddata);
68    send_string(ch);
69    //if (receiveddata > 1000) break;
70 #endif
71 #if ENABLE_CONTROLS
72    switch (receivedpacketype)
73    {
74    case OP_THRUST:
75     thrust = receiveddata;
76     sprintf(ch, "\n\r Thrust = %u", receiveddata);
77     //send_string(ch);
78     break;
79    case OP_ROLL:
80     roll = receiveddata;
81     sprintf(ch, " Roll = %u", receiveddata);
82     //send_string(ch);
83     break;
```

```
    case OP_YAW:
     yaw = receiveddata;
     sprintf(ch, " Yaw = %u", receiveddata);
     //send_string(ch);
     break;
    case OP_PITCH:
     pitch = receiveddata;
     sprintf(ch, " Pitch = %u", receiveddata);
     //send_string(ch);
     break;
    case OP_BUTTON:
     switch (receiveddata)
     {
     case BTN_CARGO_HOOK_UP:
      send_string("\n\rCargo hook up\n\r");
      break;
     case BTN_CARGO_HOOK_DOWN:
      send_string("\n\rCargo hook down\n\r");
      break;
#if ENABLE_UI
     case BTN_FLIGHT_MODE:
      send_string("\n\rEntering flight mode\n\r");
      PIDmode = 0;
      break;
#endif // ENABLE_UI

#if ENABLE_UI
     case BTN_PID_MODE:
      send_string("\n\rEntering PID mode\n\r");
      PIDmode = 1;
      break;
#endif // ENABLE_UI
     case BTN_POWER_OFF:
      send_string("\n\r Power off\n\r");
      break;
     case BTN_POWER_ON:
      send_string("\n\r Power on\n\r");
      break;
     }
     break;
    }
#endif // ENABLE_CONTROLS

#if ENABLE_UI
    if (PIDmode == 1)
    {
     switch (receivedpacketype)
     {
     case OP_KP_ROLL:
      k_value = (float) receiveddata / 100;
      sprintf(ch, "\n\rRoll Kp value = %.3f", k_value);
      send_string(ch);
      break;
```

```
    case OP_KI_ROLL:
     k_value = (float) receiveddata / 100;
     sprintf(ch, "\n\rRoll Ki value = %.3f", k_value);
     send_string(ch);
     break;
    case OP_KD_ROLL:
     k_value = (float) receiveddata / 100;
     sprintf(ch, "\n\rRoll Kd value = %.3f", k_value);
     send_string(ch);
     break;
    case OP_KP_YAW:
     k_value = (float) receiveddata / 100;
     sprintf(ch, "\n\rYaw Kp value = %.3f", k_value);
     send_string(ch);
     break;
    case OP_KI_YAW:
     k_value = (float) receiveddata / 100;
     sprintf(ch, "\n\rYaw Ki value = %.3f", k_value);
     send_string(ch);
     break;
    case OP_KD_YAW:
     k_value = (float) receiveddata / 100;
     sprintf(ch, "\n\rYaw Kd value = %.3f", k_value);
     send_string(ch);
     break;
    case OP_KP_PITCH:
     k_value = (float) receiveddata / 100;
     sprintf(ch, "\n\rPitch Kp value = %.3f", k_value);
     send_string(ch);
     break;
    case OP_KI_PITCH:
     k_value = (float) receiveddata / 100;
     sprintf(ch, "\n\rPitch Ki value = %.3f", k_value);
     send_string(ch);
     break;
    case OP_KD_PITCH:
     k_value = (float) receiveddata / 100;
     sprintf(ch, "\n\rPitch Kd value = %.3f", k_value);
     send_string(ch);
     break;
    }
   }
#endif // ENABLE_UI

  }


  if (TCNT1 >= 5500 && TCNT1 <= 6000 && !PIDmode)
  {
   //send_string("Sending telemetry");
   Send_data(OP_THRUST, receiveddata);
   _delay_ms(2);
   Send_data(OP_THRUST, receiveddata);
```

```
    _delay_ms(2);
    rfm12_tick();
    TCNT1 = 0;
   }

  }
  //sprintf(ch, "\n\rI = %u", i);
  //send_string(ch);
  while (1)
  {

  }
}
```

### D.14 drone_serial_comms.c

```
//
// Arthur: Joel And Mohammed
// Date : 3/1/17

// Pin Outs
/*
 II Matto Pins | Output/function
 ----------------------------------------
 PA0     | Battery level
 PA1     | IR Sensor
      |
 PB4     | RFM12 (SEL)
 PB5     | RFM12 (SDI)
 PB6     | RFM12 (SDO)
 PB7     | RFM12 (SCK)
      |
 PC0     | Battery level LED
      |
 PD1     | Leonardo (UART RX)
 PD2     | Leonardo (Interrupt)
 PD3     | RFM12 (IRQ)
 PD7     | Cargo hook servo
      |
 VCC     | RFM12 (3V3)
      | Cargo hook servo
 GND     | RFM12 (GND)
*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>// for sprintf()
#include <math.h>

#include "rfm12.h"
#include "drone_serial_comms.h"
```

```
#include "debug.h"

#ifdef BAUD
#include <util/setbaud.h>
#endif

volatile uint8_t serial_interrupt_flag = 0;

ISR(INT0_vect)
{
 serial_interrupt_flag = 1;    // Interrupt sets flag could be replaced
}          // with longer ISR if immediately needed

void send_string(char *str)
{
 int i;
 for (i = 0; str[i]; i++) printf(str);
}

int main(void)
{
 // Initialise rfm12 and interrupts
 init_timer();
 init_adc();
 rfm12_init();
 init_debug_uart0();
 init_pwm();
 init_interrupt();
 init_LED();

 struct euler_angles desired_values;

 uint8_t receivedpacketype;
 uint16_t receiveddata;
 uint16_t batterylevel, irheight;

 uint8_t PIDmode;
 PIDmode = 0;

 // Counter for telemetry
 //uint16_t counter;
 //counter = 0;

 char ch[100];
 while (1)
 {
  rfm12_tick();

  // Wait for data to be fully received
  if (rfm12_rx_status() == STATUS_COMPLETE)
  {
   //counter++;
   // Get the received packet type and data
```

```
    receivedpacketype = rfm12_rx_type();
    receiveddata = *rfm12_rx_buffer();
    rfm12_rx_clear();
    //send_string("received");

    // Decrypt (if enabled) and extract 10-bit data and packet type from the received packet
    Retrieve_data(&receivedpacketype, &receiveddata);
#if UPLINK_TEST
    sprintf(ch, "\n\rReceived: %u %u", receivedpacketype, receiveddata);
    send_string(ch);
    if (receiveddata > 1000) break;
#endif
    // Assign received data to the desired_values struct
#if ENABLE_CONTROLS
    switch (receivedpacketype)
    {
    case OP_THRUST:
     desired_values.throttle = receiveddata;
     //sprintf(ch, "\n\r Thrust = %u", desired_values.throttle);
     //send_string(ch);
     break;
    case OP_ROLL:
     desired_values.roll = receiveddata;
     //sprintf(ch, " Roll = %u", desired_values.roll);
     //send_string(ch);
     break;
    case OP_YAW:
     desired_values.yaw = receiveddata;
     //sprintf(ch, " Yaw = %u", desired_values.yaw);
     //send_string(ch);
     break;
    case OP_PITCH:
     desired_values.pitch = receiveddata;
     //sprintf(ch, " Pitch = %u", desired_values.pitch);
     //send_string(ch);
     break;
    case OP_BUTTON:
     switch (receiveddata)
     {
#if ENABLE_CARGO_HOOK
     case BTN_CARGO_HOOK_UP:
      // Set cargo hook up
      pwm_duty(SERVO_PWM_DUTY_MIN);
      //send_string("\n\rHook up");
      break;
     case BTN_CARGO_HOOK_DOWN:
      // Set cargo hook down
      pwm_duty(SERVO_PWM_DUTY_MAX);
      //send_string("\n\rHook down");
      break;
#endif // ENABLE_CARGO_HOOK
     case BTN_FLIGHT_MODE:
```

```
      //send_string("\n\rEntering flight mode");
      PIDmode = 0;
      break;
     case BTN_PID_MODE:
      //send_string("\n\rEntering PID mode");
      PIDmode = 1;
      break;
     }
     break;
    }
#endif

    receivedpacketype = OP_NULL;
    receiveddata = OP_NULL;
   }

   batterylevel = adc_read(PIN_BATTERY);

   // Send telemetry back to the base
   if (TCNT1 >= 5500 && TCNT1 <= 6000 && !PIDmode)
   {
    TCNT1 = 0;
    //send_string("Sending telemetry");

    // Flash LED if battery level is low
    /*if (batterylevel < 886)
    {
     PINC |= _BV(PC0);
    }*/

#if ENABLE_TELEMETRY
    Send_data(OP_BATTERY_LEVEL, batterylevel);
    _delay_ms(2);
    rfm12_tick();
    Send_data(OP_BATTERY_LEVEL, batterylevel);
    _delay_ms(2);
    rfm12_tick();

    irheight = adc_read(PIN_IR_SENSOR);

    Send_data(OP_IR_SENSOR, irheight);
    _delay_ms(2);
    rfm12_tick();
    Send_data(OP_IR_SENSOR, irheight);
    _delay_ms(2);
    rfm12_tick();
    //sprintf(ch, "\n\rBattery %u Ir sensor %u", batterylevel, irheight);
    //send_string(ch);
#endif // ENABLE_TELEMETRY
   }
   //sprintf(ch, "\n\rCounter %u", TCNT1);
   //send_string(ch);

```

```
195    if (serial_interrupt_flag)
196     send_packet(&desired_values);
197   }
198 }
199
200 // Ben's code
201 void send_packet(struct euler_angles *input)
202 {
203   if (is_valid_packet(input))
204    printf("t%dy%dp%dr%d\n", input->throttle, input->yaw, input->pitch, input->roll
       );
205   else
206    printf("t%dy%dp%dr%d\n", 0, 0, 0, 0);
207
208   serial_interrupt_flag = 0;
209 }
210
211 uint8_t is_valid_packet(struct euler_angles *input)
212 {
213   return ((input->throttle <= 1023) && (input->yaw <= 1023) &&
214    (input->pitch <= 1023) && (input->roll <= 1023));
215 }
216 //
217
218
219 void Retrieve_data(uint8_t* type, uint16_t* data)
220 {
221   // Combine packet type and data into a single 16-bit int
222   uint16_t totalpacket;
223   totalpacket = *type;
224   totalpacket = (totalpacket << 8) + *data;
225 #if ENABLE_ENCRYPTION
226   // Decrypt the received packet
227   totalpacket = Decrypt_data(totalpacket);
228 #endif
229
230   // Split the decrypted packet into the data and the packet type
231   Decode_data(type, data, totalpacket);
232 }
233
234 void Decode_data(uint8_t* type, uint16_t* data, uint16_t totalpacket)
235 {
236   // Get 10-bit data from the 16 bit packet
237   *data = totalpacket & (uint16_t)1023;
238
239   // Get packet type
240   *type = (totalpacket >> DATA_BIT_SIZE);
241 }
242
243 #if ENABLE_ENCRYPTION
244 uint16_t Decrypt_data(uint16_t packet)
245 {
246   // Get the encryption key from the first 3 bits of the packet
```

```
 uint8_t encryption_key;
 encryption_key = (packet & 7);

 // Remove the encryption key from the packet
 packet = (packet & 65528);

 // Retrieve bits that are shifted out when the left shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> (16 - encryption_key)) & get_1s(encryption_key);

 // Left shift packet and add on rotated-out bits in their previous position
 uint16_t decrypted_packet;
 decrypted_packet = (packet << encryption_key) + (rotated_out_bits << 3);

 // Remove any values from the first 3 bits and add on the encryption key
 decrypted_packet = (decrypted_packet & 65528) + encryption_key;

 return decrypted_packet;
}

uint16_t get_1s(uint8_t num)
{
 // Generate a variable that is num 1's
 uint16_t out;
 out = 1;

 uint8_t i;
 for (i = 1; i < num; i++)
 {
  out = (out << 1);
  out++;
 }

 return out;
}
#endif

void Send_data(uint8_t type, uint16_t data)
{
 // Combine packet type and data into a single 16-bit int
 uint16_t totalpacket;
 totalpacket = type;
 totalpacket = (totalpacket << DATA_BIT_SIZE) + data;

 // Encrypt data
#if ENABLE_ENCRYPTION
 totalpacket = Encrypt_data(totalpacket);
#endif
 // Split 16-bit packet into two 8-bit ints - packet type and data
 uint8_t datapacket;
 Encode_data(&type, &datapacket, totalpacket);

 // Send packet to the buffer for transmission
```

```
 rfm12_tx(sizeof(datapacket), type, &datapacket);
}



void Encode_data(uint8_t* type, uint8_t* data, uint16_t totalpacket)
{
 // Data is equal to the 8 LSBs
 *data = totalpacket;

 // Type, encryption key and 2 bits of data are held in the 8 MSBs
 *type = (totalpacket >> 8);
}



#if ENABLE_ENCRYPTION
uint16_t Encrypt_data(uint16_t packet)
{
 // Get the encryption key from the first 3 bits of the packet
 uint8_t encryption_key;
 encryption_key = (packet & 7);

 // Retrieve bits that are shifted out when the right shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> 3) & get_1s(encryption_key);

 // Right shift packet and add on rotated-out bits in their new position
 uint16_t encrypted_packet;
 encrypted_packet = (packet >> encryption_key) + (rotated_out_bits << (13 + 3 -
     encryption_key));

 // Remove any values from the first 3 bits and add on the encryption key
 encrypted_packet = (encrypted_packet & 65528) + encryption_key;

 return encrypted_packet;
}

#endif

void init_adc()//[1]
{
 // In ADCSRA Enable ADC (set ADEN) and prescaler of 64
 ADCSRA |= _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);
 // Set Vcc as the reference voltage
 ADMUX |= _BV(REFS0);
}



uint16_t adc_read(int n)//[1]
{
 ADMUX = n;// represents PA2
```

```
352        // start conversion
353
354  ADCSRA |= _BV(ADSC);
355
356  // wait for conversion to complete
357  //while(!(ADCSRA & _BV(ADIF))){};
358  while (ADCSRA & _BV(ADSC));
359  ADC = (ADCH << 8) | ADCL;// [1]
360
361  return ADC;
362
363 }
364
365 void init_timer()
366 {
367  TCCR1B |= _BV(CS12) | _BV(CS10);
368  TCNT1 = 0;
369 }
370
371
372 #if ENABLE_CARGO_HOOK
373 void init_pwm(void)
374 {
375  /* TIMER 2 */
376  DDRD |= _BV(PD6); /* PWM out */
377  DDRD |= _BV(PD7); /* inv. PWM out */
378
379
380  TCCR2A = _BV(WGM20) | /* fast PWM/MAX */
381   _BV(COM2A1); /* A output */
382  TCCR2B = _BV(CS21) |
383   _BV(CS22);    /* 1/256 prescaling */
384 }
385
386 void pwm_duty(uint8_t duty)
387 {
388  duty = duty > SERVO_PWM_DUTY_MAX ? SERVO_PWM_DUTY_MAX : duty;
389  duty = duty < SERVO_PWM_DUTY_MIN ? SERVO_PWM_DUTY_MIN : duty;
390  //printf("\nPWM=%3u  ==>  ", duty);
391  OCR2A = duty;
392 }
393 #endif // ENABLE_CARGO_HOOK
394
395 void init_interrupt()
396 {
397  DDRD |= _BV(PD2);    // Set PD2 to output
398  PORTD &= ~_BV(PD2);
399  EICRA |= _BV(ISC01) | _BV(ISC00);  // Set to trigger on rising edge
400  EIMSK |= _BV(INT0);     // Enable interrupt
401  sei();
402 }
403
404 // Set up LED for the battery level
```

```
void init_LED()
{
 DDRC |= _BV(PC0);
 PORTC &= ~_BV(PC0);
}
```

## D.15 ESCPID/AccelGyro/AccelGyro.ino

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//      2012-06-20 - improved FIFO overflow handling and simplified read process
//      2012-06-19 - completely rearranged DMP initialization code and simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//      2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
//      2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
```

```
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"


//Lawrence
#include "PWM.h"
#include "PID.h"
#include "utils.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
PID yawPID(0,0,0);
PID pitchPID(0.4,1e-5,18);
//PID pitchPID(0.4,1e-5,18);
//PID pitchPID(0.1,0.002,10);
//PID rollPID(0.4,1e-5,18);
PID rollPID(0.4,2e-5,18);
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* ============================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 * ============================================================================ */

/* ============================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

```

```
    http://arduino.cc/forum/index.php/topic,109987.0.html
    http://code.google.com/p/arduino/issues/detail?id=958
 * ========================================================================== */


// ============================================================================
// ===                      CONFIG AND SETUP                              ===
// ============================================================================

//uncomment this for debugging and to test data collection
//#define SERIAL_ENABLED


// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

unsigned long delta_time=0;
unsigned long previous_time=0;
unsigned long time_elapsed=0;

bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success
    , !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor
    measurements
```

```
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector

float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

//controller values
float throttle;
float targetYaw;
float targetPitch;
float targetRoll;
//gyro values
float gyroYaw = 0;   //initalising to zero for rate of change
float gyroYawPrev = 0;
float gyroPitch;
float gyroRoll;
//PID output values
float pidYaw;
float pidPitch;
float pidRoll;

// ==================================================================
// ===          VALUES FOR SERIAL COMMS WITH IL-MATTO          ===
// ==================================================================


String throttleString = "";         // a string to hold incoming data
String yawString      = "";
String pitchString      = "";
String rollString      = "";
int throttleInput = 0;
int yawInput = 0;
int pitchInput = 0;
int rollInput = 0;

boolean stringComplete = false;  // whether the string is complete
char packetFlag;

void pulse(void);
void serialEvent(void);
void printInt(void);

// ==================================================================
// ===                INTERRUPT DETECTION ROUTINE                ===
// ==================================================================


volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}
```

```
// ==========================================================================
// ===                        INITIAL SETUP                             ===
// ==========================================================================

void setup() {
    pinMode(LED_PIN, OUTPUT);    // configure LED for output
    digitalWrite(LED_PIN, HIGH);

    delay(1000);  //delay to allow ESCs to boot up

    digitalWrite(LED_PIN, LOW);
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    #ifdef SERIAL_ENABLED
    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
    // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    #endif

    mpu.initialize();



    pinMode(GYRO_INTERRUPT, INPUT);   //interrupt for gyro setup

    // verify connection
    #ifdef SERIAL_ENABLED
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

```

```
    // wait for ready
    //Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    /*while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again*/

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    #endif

    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(37);
    mpu.setYGyroOffset(21);
    //mpu.setYGyroOffset(20);
    mpu.setZGyroOffset(-4);
    mpu.setZAccelOffset(1545); // 1688 factory default for my test chip
    mpu.setYAccelOffset(2627);
    mpu.setXAccelOffset(1632);

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        //Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        #ifdef SERIAL_ENABLED
        Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
        #endif

        // set up ISR cause ISR_name mode
        attachInterrupt(digitalPinToInterrupt(GYRO_INTERRUPT), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        #ifdef SERIAL_ENABLED
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        #endif
        dmpReady = true;


        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
```

```
        // (if it's going to break, usually the code will be 1)
        #ifdef SERIAL_ENABLED
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
        #endif
    }

    delay(10000);
    /*-----------------------------------------------*/
      // reserve 4 bytes for the input strings:
    throttleString.reserve(4);
    yawString.reserve(4);
    pitchString.reserve(4);
    rollString.reserve(4);

    pinMode(SERIAL_INTERRUPT,OUTPUT);     //Interrupt for serial comms
    digitalWrite(SERIAL_INTERRUPT,LOW);

    Serial1.begin(57600);
    //Serial.println("Setup Complete");
    //Serial.println("Waiting for non-zero throttle");
    while(throttleInput == 0)
    {
      pulse();
      serialEvent();
      delay(100);
    }
    //Serial.println("Non-zero throttle acquired, entering loop");
    init_pwm();      //initialise pwm to 400Hz 5 to 10% duty

    digitalWrite(LED_PIN, HIGH);
}



// ==================================================================
// ===                    MAIN PROGRAM LOOP                       ===
// ==================================================================

void loop() {

    // if programming failed, don't try to do anything
    if (!dmpReady) return;



    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {


        previous_time = time_elapsed;
        time_elapsed = millis();
```

```
        delta_time = time_elapsed - previous_time;

        //get gyro data
        gyroYawPrev = gyroYaw;
        gyroYaw = ypr[0] * 180/PI;
        gyroYaw = (gyroYaw - gyroYawPrev)/delta_time;
        gyroPitch = (ypr[1] * 180/PI);
        gyroRoll = (ypr[2] * 180/PI);
        #ifdef SERIAL_ENABLED
            Serial.print("ypr\t");
            Serial.print(gyroYaw);
            Serial.print("\t");
            Serial.print(gyroPitch);
            Serial.print("\t");
            Serial.println(gyroRoll);
        #endif

        serialEvent();
        // complete the string when a newline arrives:
        if (stringComplete) {
          printInt();
          stringComplete = false;
          pulse();
        }

       throttle = rawToThrottle(throttleInput);
       targetYaw = rawToAngle(yawInput);
       targetPitch = rawToAngle(pitchInput);
       targetRoll = rawToAngle(rollInput);

        #ifdef SERIAL_ENABLED
            Serial.print("target typr\t");
            Serial.print(throttle);
            Serial.print("\t");
            Serial.print(targetYaw);
            Serial.print("\t");
            Serial.print(targetPitch);
            Serial.print("\t");
            Serial.println(targetRoll);
            Serial.print("delta_time: ");
            Serial.println(delta_time);
        #endif

       //apply PID
       pidYaw = yawPID.updatePID(targetYaw, constrain(gyroYaw,-50,50), delta_time);
       pidPitch = pitchPID.updatePID(targetPitch, constrain(gyroPitch,-50,50), delta_time);
       pidRoll = rollPID.updatePID(targetRoll, constrain(gyroRoll,-50,50), delta_time);

        #ifdef SERIAL_ENABLED
            Serial.print("PID ypr\t");
```

```
            Serial.print(pidYaw);
            Serial.print("\t");
            Serial.print(pidPitch);
            Serial.print("\t");
            Serial.println(pidRoll);
        #endif

      //update motors
       if(throttle==0 || gyroRoll>80 || gyroRoll < -80 || gyroPitch > 80 ||
           gyroPitch < -80)
       {
          setMotors(0,0,0,0);
          yawPID.resetError();
          pitchPID.resetError();
          rollPID.resetError();
       }
       else
       {
        setMotors(throttle,pidYaw, pidPitch, pidRoll);
       }
    }



    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();


    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too
        inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        #ifdef SERIAL_ENABLED
        Serial.println("FIFO overflow!");
        #endif

    // otherwise, check for DMP data ready interrupt (this should happen
        frequently)
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY short wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;
```

```
        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            // #ifdef SERIAL_ENABLED
            // Serial.print("ypr\t");
            // Serial.print(ypr[0] * 180/M_PI);
            // Serial.print("\t");
            // Serial.print(ypr[1] * 180/M_PI);
            // Serial.print("\t");
            // Serial.println(ypr[2] * 180/M_PI);
            // #endif
        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            #ifdef SERIAL_ENABLED
            Serial1.print("areal\t");
            Serial1.print(aaReal.x);
            Serial1.print("\t");
            Serial1.print(aaReal.y);
            Serial1.print("\t");
            Serial1.println(aaReal.z);
            #endif
        #endif

        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove
                gravity
            // and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            #ifdef SERIAL_ENABLED
            Serial1.print("aworld\t");
            Serial1.print(aaWorld.x);
            Serial1.print("\t");
            Serial1.print(aaWorld.y);
            Serial1.print("\t");
            Serial1.println(aaWorld.z);
            #endif
        #endif
```

```
        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}

void printInt(void)
{
    Serial1.print("Packet Received:");
    Serial1.print("t");
    Serial1.print(throttleInput);
    Serial1.print("y");
    Serial1.print(yawInput);
    Serial1.print("p");
    Serial1.print(pitchInput);
    Serial1.print("r");
    Serial1.print(rollInput);
    Serial1.print("\tt");
    Serial1.print((int)rawToThrottle(throttleInput));
    Serial1.print("y");
    Serial1.print((int)rawToThrottle(yawInput));
    Serial1.print("p");
    Serial1.print((int)rawToThrottle(pitchInput));
    Serial1.print("r");
    Serial1.println((int)rawToThrottle(rollInput));


    return;
}


void serialEvent(void) {
  while (Serial1.available()) {
    // get the new byte:
    char inChar = (char)Serial1.read();
    // add it to the input string selected by setValue:
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n')
    {
      stringComplete = true;
      throttleInput = throttleString.toInt();
      yawInput = yawString.toInt();
      pitchInput = pitchString.toInt();
      rollInput = rollString.toInt();
      throttleString = "";
      yawString = "";
      pitchString = "";
      rollString = "";
    }
    else
    {
      if(inChar == 't' ||inChar == 'y'||inChar == 'p'||inChar == 'r')
```

```
      {
        packetFlag= inChar;
      }
      else{
        switch (packetFlag) {
          case THROTTLE_ID://throttle
            throttleString += inChar;
            break;
          case YAW_ID: //yaw
            yawString += inChar;
            break;
          case PITCH_ID: //pitch
            pitchString += inChar;
            break;
          case ROLL_ID: //roll
            rollString += inChar;
            break;
        }
      }
    }
  }
  return;
}

void pulse(void)
{
  digitalWrite(SERIAL_INTERRUPT,HIGH);
  delayMicroseconds(1);
  digitalWrite(SERIAL_INTERRUPT,LOW);
  return;
}
```

## D.16 ESCPID/AccelGyro/Definitions.h

```
// Speeds are in microseconds
#define MIN_THROTTLE_IN 0.0  //in ms but not for servo, its kina arbitary
#define MAX_THROTTLE_IN 256.0  //in ms

#define LEFT_FRONT_MOTOR 0
#define RIGHT_FRONT_MOTOR  1
#define LEFT_REAR_MOTOR 2
#define RIGHT_REAR_MOTOR 3

#define TIMER_TOP 40000
//PWM VALUES FOR ESC
#define PWM_DUTY_MIN 2000.0
#define PWM_DUTY_MAX 4000.0
//PWM for mohameds servo
//#define PWM_DUTY_MIN 500
//#define PWM_DUTY_MAX 5400

```

```
#define SPEED_TO_PWM_GAIN ((PWM_DUTY_MAX-PWM_DUTY_MIN)/(MAX_THROTTLE_IN-MIN_THROTTLE_IN))

#define N_ANGLE 12.8//scalar to convert to correct range for input angles
//#define N_THROTTLE 0.511

#define DELTA_TIME 0.01

#define GYRO_INTERRUPT 7  // use pin 2 on Arduino Uno & most boards. atmega32u4 boards should use pin 7 to avoid conflict with I2C and UART
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
#define SERIAL_INTERRUPT 6


#define THROTTLE_ID 't'
#define YAW_ID 'y'
#define PITCH_ID 'p'
#define ROLL_ID 'r'

#define YAW_P_ID  'a'
#define YAW_I_ID  'b'
#define YAW_D_ID  'c'

#define PITCH_P_ID  'd'
#define PITCH_I_ID  'e'
#define PITCH_D_ID  'f'

#define ROLL_P_ID  'g'
#define ROLL_I_ID  'h'
#define ROLL_D_ID  'i'

#define PI 3.1415926
```

## D.17 ESCPID/AccelGyro/PWM.cpp

```
#include "Definitions.h"
#include "PWM.h"
#include "Arduino.h"
void init_pwm(void)
{
    /* TIMER 1 */
    DDRB |= _BV(PB5); /* PWM 1A out (pin 9 on pro micro)*/
    DDRB |= _BV(PB6); /* PWM 1B out (pin 10 on pro micro)*/
    DDRB |= _BV(PB7); /* PWM 1C out (non existant on pro micro )*/
    TCCR1A =   _BV(WGM11) | /* fast PWM/MAX */
               _BV(COM1A1)| /* A output enabled*/
               _BV(COM1B1)| /* B output enabled*/
               _BV(COM1C1); /* C output enabled*/
    TCCR1B =  _BV(CS10)  | /* no prescaling */
              _BV(WGM12) | /* fast PWM/MAX */
              _BV(WGM13) ; /* fast PWM/MAX */
    ICR1 = TIMER_TOP;
```

```
    /* TIMER 3 */
    DDRC |= _BV(PC6); /* PWM 3A out (pin 5 on pro micro)*/
    TCCR3A =  _BV(WGM31) | /* fast PWM/MAX */
              _BV(COM3A1); /* A output enabled*/
    TCCR3B =  _BV(CS30)  | /* no prescaling*/
              _BV(WGM32) | /* fast PWM/MAX */
              _BV(WGM33) ;
    ICR3 = TIMER_TOP;
    pwm_duty(LEFT_FRONT_MOTOR, MIN_THROTTLE_IN);
    pwm_duty(RIGHT_FRONT_MOTOR, MIN_THROTTLE_IN);
    pwm_duty(LEFT_REAR_MOTOR, MIN_THROTTLE_IN);
    pwm_duty(RIGHT_REAR_MOTOR, MIN_THROTTLE_IN);
}

void pwm_duty(uint8_t motor,float duty)
{   //duty is currently in ms, we need to convert it to a value in the correct range.
    //range ms: 1000-2000 , range registers: 2000-4000 therefore multiply by 2
    uint16_t intDuty;

    duty = duty*SPEED_TO_PWM_GAIN - ((SPEED_TO_PWM_GAIN*MIN_THROTTLE_IN)-PWM_DUTY_MIN);
    //Serial.println("   //check this is within the correct limits");
    if(duty>PWM_DUTY_MAX) duty = PWM_DUTY_MAX;
    else if(duty<PWM_DUTY_MIN) duty = PWM_DUTY_MIN;
    intDuty = (uint16_t)duty;
    #ifdef SERIAL_ENABLED
    Serial.print("duty: ");
    Serial.println(intDuty);
    #endif

    switch(motor)
    {
      case LEFT_FRONT_MOTOR:
        OCR1A = intDuty;
        break;
      case RIGHT_FRONT_MOTOR:
        OCR1B = intDuty;
        break;
      case LEFT_REAR_MOTOR:
        OCR1C = intDuty;
        break;
      case RIGHT_REAR_MOTOR:
        OCR3A = intDuty;
        break;
    }
}

void setMotors (float throttle, float yaw, float pitch, float roll)
{
  //reasons for these particular equations are given below
  pwm_duty(LEFT_FRONT_MOTOR,  (throttle - roll - pitch + yaw));
  pwm_duty(RIGHT_FRONT_MOTOR, (throttle + roll - pitch - yaw));
```

```
  pwm_duty(LEFT_REAR_MOTOR,    (throttle - roll + pitch - yaw));
  pwm_duty(RIGHT_REAR_MOTOR,   (throttle + roll + pitch + yaw));
}
/*
CW motors     A,C
CCW motors    D,B

            Front
            +1 pitch
          C   D
-1 roll     \-/      +1 roll    right
            /-\
          B   A
            -1 pitch

c = throttle - roll + pitch + yaw
d = throttle + roll + pitch - yaw
b = throttle - roll - pitch - yaw
a = throttle + roll - pitch + yaw
[1] http://robotics.stackexchange.com/questions/5116/
how-to-find-a-solution-for-quadcopter-pid-control
//seems to be inverted pitch so we changed it
  pwm_duty(LEFT_FRONT_MOTOR, (uint16_t)(throttle - roll + pitch + yaw));
  pwm_duty(RIGHT_FRONT_MOTOR, (uint16_t)(throttle + roll + pitch - yaw));
  pwm_duty(LEFT_REAR_MOTOR, (uint16_t)(throttle - roll - pitch - yaw));
  pwm_duty(RIGHT_REAR_MOTOR, (uint16_t)(throttle + roll - pitch + yaw));
*/
```

## D.18 ESCPID/AccelGyro_TEST/utils.cpp

```
#include "utils.h"
#include "Definitions.h"
float rawToAngle(int controlIn)
{
 float output = controlIn - 512;
 output = output/N_ANGLE;
 return output;
}

float rawToThrottle(int controlIn)
{
 float output = controlIn - 512;
  controlIn = (controlIn > MAX_THROTTLE_IN )? MAX_THROTTLE_IN : controlIn;
 if(output > 4)
 {
  return output/1.5;
 }
 return 0;
}
```

## D.19 ESCPID_TEST/AccelGyro/AccelGyro.ino

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//      2012-06-20 - improved FIFO overflow handling and simplified read process
//      2012-06-19 - completely rearranged DMP initialization code and simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//      2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
//      2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

```

```
// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"


//Lawrence
#include "PWM.h"
#include "PID.h"
#include "utils.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
PID yawPID(0,0,0);
PID pitchPID(1.5,0,0);
PID rollPID(1.5,0,0);
//PID rollPID(6,4.8,0.7);
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* ============================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 * ============================================================================ */

/* ============================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 * ============================================================================ */


// ===============================================================
// ===                    CONFIG AND SETUP                     ===
// ===============================================================

```

```
//uncomment this for debugging and to test data collection
//#define SERIAL_ENABLED


// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL



bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector

float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

//controller values
float throttle;
float targetYaw;
float targetPitch;
```

```
float targetRoll;
//gyro values
float gyroYaw = 0;   //initalising to zero for rate of change
float gyroYawPrev = 0;
float gyroPitch;
float gyroRoll;
//PID output values
float pidYaw;
float pidPitch;
float pidRoll;

// ==================================================================
// ===          VALUES FOR CERIAL COMMS WITH IL-MATTO           ===
// ==================================================================


String throttleString = "";         // a string to hold incoming data
String yawString      = "";
String pitchString      = "";
String rollString      = "";
int throttleInput = 0;
int yawInput = 0;
int pitchInput = 0;
int rollInput = 0;

boolean stringComplete = false;  // whether the string is complete
char packetFlag;

void printInt(void);

// ==================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ==================================================================


volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin
    has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}



// ==================================================================
// ===                       INITIAL SETUP                      ===
// ==================================================================

void setup() {
    pinMode(LED_PIN, OUTPUT);    // configure LED for output
    digitalWrite(LED_PIN, HIGH);

    delay(1000);  //delay to allow ESCs to boot up

```

```
    digitalWrite(LED_PIN, LOW);
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    #ifdef SERIAL_ENABLED
    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
    // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    #endif

    mpu.initialize();


    /*-----------------------------------------------*/
      // reserve 4 bytes for the input strings:
    throttleString.reserve(4);
    yawString.reserve(4);
    pitchString.reserve(4);
    rollString.reserve(4);

    pinMode(SERIAL_INTERRUPT,OUTPUT);     //Interrupt for serial comms
    digitalWrite(SERIAL_INTERRUPT,LOW);

    Serial1.begin(57600);
    Serial.println("Setup Complete");
    Serial.println("Waiting for non-zero throttle");
    // while(throttleInput == 0)
    // {
    //    pulse();
    //    serialEvent();
    //    delay(100);
    // }
    Serial.println("Non-zero throttle acquired, entering loop");
    init_pwm();      //initialise pwm to 400Hz 5 to 10% duty

```

```
    pinMode(GYRO_INTERRUPT, INPUT);    //interrupt for gyro setup

    // verify connection
    #ifdef SERIAL_ENABLED
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F(
        "MPU6050 connection failed"));

    // wait for ready
    //Serial.println(F("\nSend any character to begin DMP programming and demo:
        "));
    while (Serial.available() && Serial.read()); // empty buffer
    /*while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again*/

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    #endif

    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(38);
    mpu.setYGyroOffset(22);
    mpu.setZGyroOffset(-4);
    mpu.setZAccelOffset(1547); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        //Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        //Serial.println(F("Enabling interrupt detection (Arduino external
            interrupt 0)..."));


        // set up ISR cause ISR_name mode
        attachInterrupt(digitalPinToInterrupt(GYRO_INTERRUPT), dmpDataReady,
            RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to
            use it
        //Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;


        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
```

```
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        #ifdef SERIAL_ENABLED
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
        #endif
    }



    digitalWrite(LED_PIN, HIGH);
}



// ==========================================================================
// ===                      MAIN PROGRAM LOOP                           ===
// ==========================================================================

void loop() {

    // if programming failed, don't try to do anything
    if (!dmpReady) return;



    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        //get gyro data
        gyroYawPrev = gyroYaw;
        gyroYaw = ypr[0] * 180/M_PI;
        gyroYaw = (gyroYaw - gyroYawPrev)/DELTA_TIME;
        gyroPitch = (ypr[1] * 180/M_PI);
        gyroRoll = (ypr[2] * 180/M_PI);


        // serialEvent();
        // // complete the string when a newline arrives:
        // if (stringComplete) {
        //   printInt();
        //   stringComplete = false;
        //   pulse();
        // }
      Serial.print("ypr\t");
      Serial.print(ypr[0] * 180/M_PI);
      Serial.print("\t");
      Serial.print(ypr[1] * 180/M_PI);
      Serial.print("\t");
      Serial.println(ypr[2] * 180/M_PI);
       throttle = 0;
       targetYaw = 0;
```

```
        targetPitch = 0;
        targetRoll = 0;

        //apply PID
        pidYaw = yawPID.updatePID(targetYaw, constrain(gyroYaw,-50,50), DELTA_TIME);
        pidPitch = pitchPID.updatePID(targetPitch, constrain(gyroPitch,-50,50), DELTA_TIME);
        pidRoll = rollPID.updatePID(targetRoll, constrain(gyroRoll,-50,50), DELTA_TIME);

      Serial.println(pidYaw);
      Serial.println(pidPitch);
      Serial.println(pidRoll);

      setMotors(throttle,pidYaw, pidPitch, pidRoll);
    }



    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();


    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        #ifdef SERIAL_ENABLED
        Serial.println("FIFO overflow!");
        #endif

    // otherwise, check for DMP data ready interrupt (this should happen frequently)
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY short wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
```

```
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            #ifdef SERIAL_ENABLED
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
            #endif
        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            #ifdef SERIAL_ENABLED
            Serial1.print("areal\t");
            Serial1.print(aaReal.x);
            Serial1.print("\t");
            Serial1.print(aaReal.y);
            Serial1.print("\t");
            Serial1.println(aaReal.z);
            #endif
        #endif

        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove
                gravity
            // and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            #ifdef SERIAL_ENABLED
            Serial1.print("aworld\t");
            Serial1.print(aaWorld.x);
            Serial1.print("\t");
            Serial1.print(aaWorld.y);
            Serial1.print("\t");
            Serial1.println(aaWorld.z);
            #endif
        #endif



        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
```

```
}

void printInt(void)
{
    Serial1.print("Packet Received:");
    Serial1.print("t");
    Serial1.print(throttleInput);
    Serial1.print("y");
    Serial1.print(yawInput);
    Serial1.print("p");
    Serial1.print(pitchInput);
    Serial1.print("r");
    Serial1.print(rollInput);
    Serial1.print("\tt");
    Serial1.print((int)rawToThrottle(throttleInput));
    Serial1.print("y");
    Serial1.print((int)rawToThrottle(yawInput));
    Serial1.print("p");
    Serial1.print((int)rawToThrottle(pitchInput));
    Serial1.print("r");
    Serial1.println((int)rawToThrottle(rollInput));


    return;
}
```

## D.20 ServoPID/AccelGyro/AccelGyro.ino

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//      2012-06-20 - improved FIFO overflow handling and simplified read process
//      2012-06-19 - completely rearranged DMP initialization code and simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//      2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
//      2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
```

```
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"


//Lawrence
#include "PWM.h"
#include "PID.h"
#include "utils.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
PID yawPID(1,0,0);
PID pitchPID(1,0,0);
```

```
//PID rollPID(8,6,2);   //200hz
PID rollPID(6,4,1);     //100hz
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* ============================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 * ============================================================================ */

/* ============================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 * ============================================================================ */


// ===============================================================
// ===                    CONFIG AND SETUP                     ===
// ===============================================================

//uncomment this for debugging and to test data collection
#define SERIAL_ENABLED


// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL


```

```
#define INTERRUPT_PIN 7  // use pin 2 on Arduino Uno & most boards. atmega32u4 boards should use pin 7 to avoid conflict with I2C and UART
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector

float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

//controller values
float throttle;
float targetYaw;
float targetPitch;
float targetRoll;
//gyro values
float gyroYaw;
float gyroPitch;
float gyroRoll;
//PID output values
float pidYaw;
float pidPitch;
float pidRoll;


// ===============================================================
// ===                INTERRUPT DETECTION ROUTINE                ===
// ===============================================================


volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}


```

```
// ===========================================================================
// ===                          INITIAL SETUP                              ===
// ===========================================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    #ifdef SERIAL_ENABLED
    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
    // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    #endif

    mpu.initialize();
    init_pwm();

    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    #ifdef SERIAL_ENABLED
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // wait for ready
    //Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    /*while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again*/

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
```

```
    #endif
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(34);
    mpu.setYGyroOffset(21);
    mpu.setZGyroOffset(-5);
    mpu.setZAccelOffset(1519); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        //Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        //Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));


    // set up ISR        cause                ISR_name    mode
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        //Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;


        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        #ifdef SERIAL_ENABLED
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
        #endif
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}



// ==========================================================================
// ===                    MAIN PROGRAM LOOP                               ===
```

```
// ==================================================================

void loop() {

    // if programming failed, don't try to do anything
    if (!dmpReady) return;



    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        //get gyro data
        gyroYaw = ypr[0] * 180/M_PI;
        gyroPitch = ypr[1] * 180/M_PI;
        gyroRoll = ypr[2] * 180/M_PI;

        //get throttle data
        if(throttle<MAX_MOTOR_SPEED) throttle++;
          else throttle=MIN_MOTOR_SPEED;
       targetYaw = 0;
       targetPitch = 0;
       targetRoll = 0;

       //apply PID
       //pidYaw = yawPID.updatePID(targetYaw, gyroYaw, DELTA_TIME);
       //pidPitch = pitchPID.updatePID(targetPitch, gyroPitch, DELTA_TIME);
       pidRoll = rollPID.updatePID(targetRoll, constrain(gyroRoll,-50,50),
           DELTA_TIME);
       //update motors
       //setMotors (throttle, 0, 0, 0);
      pwm_duty(LEFT_FRONT_MOTOR, 1500-pidRoll);
    }

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();


    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too
        inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        #ifdef SERIAL_ENABLED
        Serial.println(F("FIFO overflow!"));
        #endif

    // otherwise, check for DMP data ready interrupt (this should happen
        frequently)
    } else if (mpuIntStatus & 0x02) {
```

```
        // wait for correct available data length, should be a VERY short wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            #ifdef SERIAL_ENABLED
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
            #endif
        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            #ifdef SERIAL_ENABLED
            Serial.print("areal\t");
            Serial.print(aaReal.x);
            Serial.print("\t");
            Serial.print(aaReal.y);
            Serial.print("\t");
            Serial.println(aaReal.z);
            #endif
        #endif

        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove
                gravity
            // and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            #ifdef SERIAL_ENABLED
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
```

```
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
            #endif
        #endif



        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

## D.21 leonardoPWM.ino

```
#define TIMER_TOP 40000
//PWM DUTY for ESCs
#define PWM_DUTY_MIN 2000 //does this need to be 2020 when we have a big delay?
#define PWM_DUTY_MAX 4000
void init_pwm(void);
void pwm_duty(uint16_t A,uint16_t B,uint16_t C,uint16_t D);
bool done = false;
int x1 =PWM_DUTY_MIN;
int x2 =PWM_DUTY_MIN;
void setup()
{
  delay(1000);
  init_pwm();
  pwm_duty(PWM_DUTY_MAX,PWM_DUTY_MAX,PWM_DUTY_MAX,PWM_DUTY_MAX);
  delay(10000);
  pwm_duty(PWM_DUTY_MIN,PWM_DUTY_MIN,PWM_DUTY_MIN,PWM_DUTY_MIN);
  delay(10000);
}

void loop() {
    //TXLED1;]
    while(!done)
    {
     delay(10);
     if(x1<3000) x1++;
     else
     {
      x1=PWM_DUTY_MIN;
      done = true;
     }
     pwm_duty(x1,x1,x1,x1);
    }

}

```

```
void init_pwm(void)
{
    /* TIMER 1 */
    DDRB |= _BV(PB5); /* PWM 1A out (pin 9 on pro micro)*/
    DDRB |= _BV(PB6); /* PWM 1B out (pin 10 on pro micro)*/
    DDRB |= _BV(PB7); /* PWM 1C out (non existant on pro micro )*/
    ICR1 = TIMER_TOP;
    TCCR1A =   _BV(WGM11) | /* fast PWM/MAX */
               _BV(COM1A1)| /* A output enabled*/
               _BV(COM1B1)| /* B output enabled*/
               _BV(COM1C1); /* C output enabled*/
    TCCR1B =  _BV(CS10)  | /*no prescalar*/
              //_BV(CS11)  | /* /8 prescaling */
              _BV(WGM12) | /* fast PWM/MAX */
              _BV(WGM13) ; /* fast PWM/MAX */
    /* TIMER 3 */
    DDRC |= _BV(PC6); /* PWM 3A out (pin 5 on pro micro)*/
    TCCR3A =  _BV(WGM31) | /* fast PWM/MAX */
              _BV(COM3A1); /* A output enabled*/
    TCCR3B =  _BV(CS30)  | /*no prescalar*/
              //_BV(CS31)  | /* /8 prescaling */
              _BV(WGM32) | /* fast PWM/MAX */
              _BV(WGM33) ;
    ICR3 = TIMER_TOP;
}
void pwm_duty(uint16_t A,uint16_t B,uint16_t C,uint16_t D)
{   //uncomment for limits
    if(A>PWM_DUTY_MAX) A = PWM_DUTY_MAX;
    else if(A<PWM_DUTY_MIN) A = PWM_DUTY_MIN;
    if(B>PWM_DUTY_MAX) B = PWM_DUTY_MAX;
    else if(B<PWM_DUTY_MIN) B = PWM_DUTY_MIN;
    if(C>PWM_DUTY_MAX) C = PWM_DUTY_MAX;
    else if(C<PWM_DUTY_MIN) C = PWM_DUTY_MIN;
    if(D>PWM_DUTY_MAX) D = PWM_DUTY_MAX;
    else if(D<PWM_DUTY_MIN) D = PWM_DUTY_MIN;
    OCR1A = A;
    OCR1B = B;
    OCR1C = C;
    OCR3A = D;
}
```

## D.22 leonardoPWM_input.ino

```
#define TIMER_TOP 40000
//PWM DUTY for ESCs
#define PWM_DUTY_MIN 2000 //does this need to be 2020 when we have a big delay?
#define PWM_DUTY_MAX 4000



#define THROTTLE_ID 't'
#define YAW_ID 'y'
```

```
#define PITCH_ID 'p'
#define ROLL_ID 'r'

#define YAW_P_ID  'a'
#define YAW_I_ID  'b'
#define YAW_D_ID  'c'

#define PITCH_P_ID  'd'
#define PITCH_I_ID  'e'
#define PITCH_D_ID  'f'

#define ROLL_P_ID  'g'
#define ROLL_I_ID  'h'
#define ROLL_D_ID  'i'


void init_pwm(void);
void pwm_duty(uint16_t A,uint16_t B,uint16_t C,uint16_t D);
int thrust  = PWM_DUTY_MIN;
int yaw     = PWM_DUTY_MIN;
int pitch   = PWM_DUTY_MIN;
int roll    = PWM_DUTY_MIN;

String throttleString = "";          // a string to hold incoming data
String yawString      = "";
String pitchString      = "";
String rollString      = "";
int throttleInput = 0;
int yawInput = 0;
int pitchInput = 0;
int rollInput = 0;

//for the k values
String yawKpString = "";
String yawKiString = "";
String yawKdString = "";

String pitchKpString = "";
String pitchKiString = "";
String pitchKdString = "";

String rollKpString = "";
String rollKiString = "";
String rollKdString = "";

boolean stringComplete = false;  // whether the string is complete
char packetFlag;

void pulse(void);
void Serial1Event(void);
void printInt(void);

void setup()
```

```
{
  delay(1000);
  //wait for controller input that isnt zero data


  Serial1.begin(57600);
  // reserve 4 bytes for the input strings:
  throttleString.reserve(4);
  yawString.reserve(4);
  pitchString.reserve(4);
  rollString.reserve(4);
  pinMode(6,OUTPUT);
  digitalWrite(6,LOW);
  Serial1.println("Setup Complete");
  Serial1.println("Waiting for non-zero throttle");
  while(throttleInput == 0)
  {
    pulse();
    Serial1Event();
    delay(100);
  }
  Serial1.println("Non-zero throttle acquired, entering loop");


  init_pwm();
  //pwm_duty(thrust,yaw,pitch,roll);//change this to controller input
}

void loop()
{
    Serial1Event();
  // complete the string when a newline arrives:
  if (stringComplete) {
   printInt();
    // clear the string:
    stringComplete = false;
    delay(10);
    pulse();
  }

    //TXLED1;
      //get data from Ilmatto and output same to all motors
     pwm_duty((uint16_t)rawToThrottle(throttleInput),
              (uint16_t)rawToThrottle(yawInput),
              (uint16_t)rawToThrottle(pitchInput),
              (uint16_t)rawToThrottle(rollInput));
     //delay(5);



}

void init_pwm(void)
```

```
{
    /* TIMER 1 */
    DDRB |= _BV(PB5); /* PWM 1A out (pin 9 on pro micro)*/
    DDRB |= _BV(PB6); /* PWM 1B out (pin 10 on pro micro)*/
    DDRB |= _BV(PB7); /* PWM 1C out (non existant on pro micro )*/
    ICR1 = TIMER_TOP;
    TCCR1A =   _BV(WGM11) | /* fast PWM/MAX */
               _BV(COM1A1)| /* A output enabled*/
               _BV(COM1B1)| /* B output enabled*/
               _BV(COM1C1); /* C output enabled*/
    TCCR1B =  _BV(CS10)  | /*no prescalar*/
              //_BV(CS11)  | /* /8 prescaling */
              _BV(WGM12) | /* fast PWM/MAX */
              _BV(WGM13) ; /* fast PWM/MAX */
    /* TIMER 3 */
    DDRC |= _BV(PC6); /* PWM 3A out (pin 5 on pro micro)*/
    TCCR3A =  _BV(WGM31) | /* fast PWM/MAX */
              _BV(COM3A1); /* A output enabled*/
    TCCR3B =  _BV(CS30)  | /*no prescalar*/
              //_BV(CS31)  | /* /8 prescaling */
              _BV(WGM32) | /* fast PWM/MAX */
              _BV(WGM33) ;
    ICR3 = TIMER_TOP;
}
void pwm_duty(uint16_t A,uint16_t B,uint16_t C,uint16_t D)
{   //uncomment for limits
    /*if(A>PWM_DUTY_MAX) A = PWM_DUTY_MAX;
    else if(A<PWM_DUTY_MIN) A = PWM_DUTY_MIN;
    if(B>PWM_DUTY_MAX) B = PWM_DUTY_MAX;
    else if(B<PWM_DUTY_MIN) B = PWM_DUTY_MIN;
    if(C>PWM_DUTY_MAX) C = PWM_DUTY_MAX;
    else if(C<PWM_DUTY_MIN) C = PWM_DUTY_MIN;
    if(D>PWM_DUTY_MAX) D = PWM_DUTY_MAX;
    else if(D<PWM_DUTY_MIN) D = PWM_DUTY_MIN;*/
    OCR1A = A;
    OCR1B = B;
    OCR1C = C;
    OCR3A = D;
}

float rawToThrottle(int controlIn)
{
  float output = controlIn - 512;
  if(output > 4)
  {
    output = output*7.8125+2000;
    //Serial1.print(output);
    return output;
  }
  return 2000;
}

void printInt(void)
```

```
{
    Serial1.print("Packet Received:");
    Serial1.print("t");
    Serial1.print(throttleInput);
    Serial1.print("y");
    Serial1.print(yawInput);
    Serial1.print("p");
    Serial1.print(pitchInput);
    Serial1.print("r");
    Serial1.println(rollInput);
//    Serial11.print("\tt");
//    Serial11.print((int)rawToThrottle(throttleInput));
//    Serial11.print("y");
//    Serial11.print((int)rawToThrottle(yawInput));
//    Serial11.print("p");
//    Serial11.print((int)rawToThrottle(pitchInput));
//    Serial11.print("r");
//    Serial11.println((int)rawToThrottle(rollInput));


    return;
}


void Serial1Event(void) {
    bool throttle_set = false;
    bool yaw_set = false;
    bool pitch_set = false;
    bool roll_set = false;
    bool yaw_p_set = false;
    bool yaw_i_set = false;
    bool yaw_d_set = false;
    bool pitch_p_set = false;
    bool pitch_i_set = false;
    bool pitch_d_set = false;
    bool roll_p_set = false;
    bool roll_i_set = false;
    bool roll_d_set = false;
    float yaw_kp;
    float yaw_ki;
    float yaw_kd;
    float pitch_kp;
    float pitch_ki;
    float pitch_kd;
    float roll_kp;
    float roll_ki;
    float roll_kd;
  while (Serial1.available()) {
    // get the new byte:
    char inChar = (char)Serial1.read();
    // add it to the input string selected by setValue:
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
```

```
    if (inChar == '\n')
    {
        stringComplete = true;
        if(throttle_set)
        {
            throttleInput = throttleString.toInt();
            throttleString = "";
        }
        if(yaw_set)
        {
            yawInput = yawString.toInt();
            yawString = "";
        }
        if(pitch_set)
        {
            pitchInput = pitchString.toInt();
            pitchString = "";
        }
        if(roll_set)
        {
            rollInput = rollString.toInt();
            rollString = "";
        }
        if(yaw_p_set)
        {
            yaw_kp = yawKpString.toInt()/100;
            Serial1.print("yaw_kp:");
            Serial1.println(yaw_kp);
            yawKpString = "";
        }
        if(yaw_i_set)
        {
            yaw_ki = yawKiString.toInt()/100;
            Serial1.print("yaw_ki:");
            Serial1.println(yaw_ki);
            yawKiString = "";
        }
        if(yaw_d_set)
        {
            yaw_kd = yawKdString.toInt()/100;
            Serial1.print("yaw_kd:");
            Serial1.println(yaw_kd);
            yawKdString = "";
        }
        if(pitch_p_set)
        {
            pitch_kp = pitchKpString.toInt()/100;
            Serial1.print("pitch_kp:");
            Serial1.println(pitch_kp);
            pitchKpString = "";
        }
        if(pitch_i_set)
        {
```

```
            pitch_ki = pitchKiString.toInt()/100;
            Serial1.print("pitch_ki:");
           Serial1.println(pitch_ki);
           pitchKiString = "";
        }
        if(pitch_d_set)
        {
            pitch_kd =pitchKdString.toInt()/100;
            Serial1.print("pitch_kd:");
            Serial1.println(pitch_kd);
            pitchKdString = "";
        }
        if(roll_p_set)
        {
            roll_kp = rollKpString.toInt()/100;
            Serial1.print("roll_kp:");
           Serial1.println(roll_kp);
           rollKpString = "";
        }
        if(roll_i_set)
        {
            roll_ki = (float)rollKiString.toInt()/100;
            Serial1.print("roll_ki:");
           Serial1.println(roll_ki);
           rollKiString = "";
        }
        if(roll_d_set)
        {
            roll_kd =rollKdString.toInt()/100;
            Serial1.print("pitch_kd:");
            Serial1.println(roll_kd);
            rollKdString = "";
        }
    }
    else
    {
      if(inChar == THROTTLE_ID || inChar == YAW_ID || inChar == PITCH_ID ||
          inChar == ROLL_ID
        || inChar == YAW_P_ID|| inChar == YAW_I_ID|| inChar == YAW_D_ID
        || inChar == PITCH_P_ID|| inChar == PITCH_I_ID|| inChar == PITCH_D_ID
        || inChar == ROLL_P_ID|| inChar == ROLL_I_ID|| inChar == ROLL_D_ID)
      {
        packetFlag= inChar;
      }
      else{
        switch (packetFlag) {
          case THROTTLE_ID://throttle
            throttleString += inChar;
            throttle_set = true;
            break;
          case YAW_ID: //yaw
            yawString += inChar;
            yaw_set = true;
```

```
327             break;
328           case PITCH_ID: //pitch
329             pitchString += inChar;
330             pitch_set = true;
331             break;
332           case ROLL_ID: //roll
333             rollString += inChar;
334             roll_set = true;
335             break;
336           case YAW_P_ID:
337             yawKpString += inChar;
338             yaw_p_set = true;
339             break;
340           case YAW_I_ID:
341             yawKiString += inChar;
342             yaw_i_set = true;
343             break;
344           case YAW_D_ID:
345             yawKdString += inChar;
346             yaw_d_set = true;
347             break;
348           case PITCH_P_ID:
349             pitchKdString += inChar;
350             pitch_p_set = true;
351             break;
352           case PITCH_I_ID:
353             pitchKiString += inChar;
354             pitch_i_set = true;
355             break;
356           case PITCH_D_ID:
357             pitchKdString += inChar;
358             pitch_d_set = true;
359             break;
360           case ROLL_P_ID:
361             rollKpString += inChar;
362             roll_p_set = true;
363             break;
364           case ROLL_I_ID:
365             rollKiString += inChar;
366             roll_i_set = true;
367
368             break;
369           case ROLL_D_ID:
370             rollKdString += inChar;
371             roll_d_set = true;
372             break;
373         }
374       }
375     }
376   }
377   return;
378 }
379 void pulse(void)
```

```
{
  digitalWrite(6,HIGH);
  delayMicroseconds(1);
  digitalWrite(6,LOW);
  return;
}
```

### D.23 i2c-lib-test/main.c

```
//I2C test

// must be compiled with twimastertimeout.c


#include "debug.h"
#include <util/delay.h>
#include <avr/io.h>
#include "i2cmaster.h"

#define SLA 110100
#define DEV 0
#define TWI_READ 1
#define TWI_WRITE 0


void start_wrapped(unsigned char address)
{
 if(i2c_start(address))
 {
  // returns 1 if unable to connect
  printf("I2C start failed - device busy\n");
 }
 else
 {
 printf("Device Accessible\n");
 }
}

void write_wrapped(unsigned char data)
{
 if(i2c_write(data))
 {
  // returns 1 if write failed
  printf("Write Failed\n");
 }
 else
 {
  printf("Write Success\n");
 }
}


```

```

int main(void) {

 init_debug_uart0();
 i2c_init();
 printf("UART Connected\n");

 // therefore device address is 0xD1 for a read
 //          0xD0 for a write

 unsigned char data;

 for(;;)
 {
  start_wrapped(0xD0); // write mode
  write_wrapped(0x42); // selects gyro_x_low
  start_wrapped(0xD1); // read mode, now register is selected
  data = i2c_readNak(); // read a byte
  i2c_stop(); // release the bus
  printf("Data: %d\n\n", (int)data);
  _delay_ms(1000);
 }

}
```

### D.24 onboard_serial_test_ard_3.ino

```
String throttleString = "";         // a string to hold incoming data
String yawString      = "";
String pitchString      = "";
String rollString      = "";
int throttleInput;
int yawInput;
int pitchInput;
int rollInput;

boolean stringComplete = false;  // whether the string is complete
char packetFlag;

void pulse(void);
void serialEvent(void);
void printInt(void);


void setup() {
  // initialize serial:
  //Serial.begin(57600);
  Serial1.begin(57600);
  // reserve 4 bytes for the input strings:
  throttleString.reserve(4);
  yawString.reserve(4);
  pitchString.reserve(4);
```

```
  rollString.reserve(4);
  pinMode(7,OUTPUT);
  digitalWrite(7,LOW);
  pulse();
  Serial1.println("Starting program");
}

void loop() {
  serialEvent();
  // complete the string when a newline arrives:
  if (stringComplete) {
   printInt();
    // clear the string:
    stringComplete = false;
    delay(10);
    pulse();
  }
}

void printInt(void)
{
   Serial1.print("Packet Received:");
    Serial1.print("t");
    Serial1.print(throttleInput);
    Serial1.print("y");
    Serial1.print(yawInput);
    Serial1.print("p");
    Serial1.print(pitchInput);
    Serial1.print("r");
    Serial1.println(rollInput);
    return;
}


void serialEvent(void) {
  while (Serial1.available()) {
    // get the new byte:
    char inChar = (char)Serial1.read();
    // add it to the input string selected by setValue:
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n')
    {
      stringComplete = true;
      throttleInput = throttleString.toInt();
      yawInput = yawString.toInt();
      pitchInput = pitchString.toInt();
      rollInput = rollString.toInt();
      throttleString = "";
      yawString = "";
      pitchString = "";
      rollString = "";
    }
```

```
    else
    {
      if(inChar == 't' ||inChar == 'y'||inChar == 'p'||inChar == 'r')
      {
        packetFlag= inChar;
      }
      else{
        switch (packetFlag) {
          case 't'://throttle
            throttleString += inChar;
            break;
          case 'y': //yaw
            yawString += inChar;
            break;
          case 'p': //pitch
            pitchString += inChar;
            break;
          case 'r': //roll
            rollString += inChar;
            break;
        }
      }
    }
  }
  return;
}

void pulse(void)
{
  digitalWrite(7,HIGH);
  delayMicroseconds(1);
  digitalWrite(7,LOW);
  return;
}
```

### D.25 onboard_serial_test_final.c

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

#include <avr/interrupt.h>

#include "debug.h"

/* #define BAUD 57600

#ifdef BAUD
#include <util/setbaud.h>
#endif */

volatile uint8_t serial_interrupt_flag = 0;

```

```
ISR(INT0_vect)
{
 serial_interrupt_flag = 1;    // Interrupt sets flag could be replaced
}              // with longer ISR if immediately needed

struct euler_angles {      //Structure containing TYPR
 uint16_t throttle;
 uint16_t yaw;
 uint16_t pitch;
 uint16_t roll;
};

uint8_t is_valid_packet(struct euler_angles *input);
void send_packet(struct euler_angles *input);

int main()
{

 init_debug_uart0();

 struct euler_angles desired_values;

 desired_values.throttle = 512;
 desired_values.yaw = 554;
 desired_values.pitch = 747;
 desired_values.roll = 350;


 DDRD |= _BV(PD2);    // Set PD2 to output
 PORTD &= ~_BV(PD2);
 EICRA |= _BV(ISC01) | _BV(ISC00);  // Set to trigger on rising edge
 EIMSK |= _BV(INT0);    // Enable interrupt
 sei();     // Enable global interrupt

 /*
 INSERT REST OF INIT HERE
 */

 while(1)
 {
  if(serial_interrupt_flag)
   send_packet(&desired_values);
  /*
  INSERT REST OF LOOP HERE
  */
 }
}

void send_packet(struct euler_angles *input)
{
 if(is_valid_packet(input))
 {
```

```
  printf("t%dy%dp%dr%d\n", input->throttle, input->yaw, input->pitch, input->roll
      ); //
 }

 else
  printf("t%dy%dp%dr%d\n", 0, 0, 0, 0);

  serial_interrupt_flag = 0;
}

uint8_t is_valid_packet(struct euler_angles *input)
{
 return ((input->throttle <= 1023) && (input->yaw <= 1023) &&
   (input->pitch <= 1023) && (input->roll <= 1023));
}
```

## D.26 SD_card_test_main.cpp

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <util/delay.h>
#include <string.h>
#include <math.h>
#include "tft.h"
#include "conv.h"
#include "sd.h"
#include "mbr.h"
#include "fat32.h"
#include "dirent.h"
#include "file.h"
#include "errno.h"
#include "apps.h"
#include "dac.h"

#define PI 3.1415927

class sdhw_t sd;

void init(void)
{
 DDRB |= 0x80;    // LED
 PORTB |= 0x80;
 init_dac();
 sei();
 tft::init();
 tft::setOrient(tft::Landscape | tft::Flipped);
 tft::foreground = COLOUR_565_888(0x00FF00);
 tft::background = 0;
 tft::clean();
 stdout = tft::devout();
 tft::setBGLight(true);
```

```
35 }
36
37 int main(void)
38 {
39  init();
40  goto start;
41
42 fin:
43  puts("Test finished, remove SDCard to run again...");
44  while (sd.detect());
45 start:
46  tft::clean();
47  tft::zoom = 1;
48  puts("*** SDCard library test ***");
49  puts("Please insert SDCard...");
50  while (!sd.detect());
51  printf("SDCard detected, %swrite protected, ", sd.writeProtected() ? "" : "not "
      );
52
53  uint8_t err;
54  if ((err = sd.init()) != 0x00) {
55   printf("Initialisation failed: %u, err: %u\n", err, sd.err());
56   goto fin;
57  }
58  printf("%u GB\n", (uint16_t)(sd.size() / 1024 / 1024));
59
60  class mbr_t mbr(&sd);
61  if (mbr.err()) {
62   printf("Read MBR failed: %u\n", mbr.err());
63   goto fin;
64  }
65  if (mbr.type(0) != mbr_t::FAT32) {
66   printf("Partition 1 (0x%02X) is not FAT32!\n", mbr.type(0));
67   goto fin;
68  }
69  puts("Partition 1 is FAT32, reading...");
70
71  fat32_t fs(&sd, mbr.address(0));
72  if (fs.err()) {
73   printf("Read FAT32 file system failed: %u\n", fs.err());
74   goto fin;
75  }
76  op::setfs(&fs);
77
78 #if 0
79  puts("\nReading '/Il Matto/Testing' directory...");
80  DIR *dir = op::opendir("/Il Matto/Testing");
81  if (dir == NULL) {
82   printf("opendir failed: %u\n", errno);
83   goto fin;
84  }
85  uint16_t bgc = tft::foreground();
86  struct dirent *ent;
```

```
 while ((ent = op::readdir(dir)) != NULL) {
  if (ent->d_type & IS_DIR)
   tft::setForeground(conv::c32to16(0x0000FF));
  else
   tft::setForeground(conv::c32to16(0xFFFFFF));
  printf("%-25s\t|\t\%#02x\t|\t%lukB\n", ent->d_name, ent->d_type, ent->d_size /
      1024);
 }
 op::closedir(dir);
 tft::setForeground(bgc);

 puts("\nReading '/II Matto/Testing/Fin.bmp' file...");
 FILE *fp = op::fopen("/II Matto/Testing/Fin.bmp", "r");
 if (fp == NULL) {
  printf("Open file failed: %u\n", errno);
  goto fin;
 }
 for (uint8_t i = 0; i < 10; i++)
  fgetc(fp);
 uint16_t offset = conv::uint32(fp);
 for (uint8_t i = 10 + 4; i < offset; i++)
  fgetc(fp);
 uint16_t x, y;
 tft::bmp(true);
 tft::all();
 tft::start();
 for (y = 240; y > 0; y--)
  for (x = 0; x < 320; x++)
   tft::write16(conv::c32to16(conv::uint24(fp)));
 tft::bmp(false);
 op::fclose(fp);
#endif
 pwm::init();
 timer1::init();
 //apps::wav("/II Matto/Testing/river.wav");
 apps::wtxt("Data/write_test.txt");
 apps::wtxt("Data/log.txt");
 apps::rtxt("Data/Read_Test.txt");

 goto fin;

 return 1;
}
```

## D.27 apps.h

```
#ifndef ANIMATION_H
#define ANIMATION_H

#include <stdio.h>
#include <inttypes.h>
#include "timer1.h"
```

```
7 #include "dirent.h"
8 #include "file.h"
9 #include "tty.h"
10 #include "pwm.h"
11 #include "dac.h"
12
13
14 namespace apps
15 {
16  //static uint8_t image(const char *path);
17  //static uint8_t image(struct dirent *ent);
18  //static uint8_t animation(const char *path, const uint_t clr);
19  static uint8_t wav(const char *path);
20  static uint8_t wtxt(const char *path);
21  static uint8_t rtxt(const char *path);
22 }
23
24 #if 0
25 static uint8_t apps::image(const char *path)
26 {
27  using namespace display;
28
29  fprintf(stderr, TTY_YELLOW "apps::image(): \'%s\'\n", path);
30  FILE *fp = op::fopen(path, "r");
31  if (fp == NULL) {
32   fprintf(stderr, TTY_RED "apps::image(): open file failed: %u\n", errno);
33   return 1;
34  }
35  for (uint_t r = 0; r < BUFF_H; r++)
36   for (uint_t c = 0; c < BUFF_W / 8; c++) {
37    buff[r][c][BuffRed] = fgetc(fp);
38    buff[r][c][BuffGreen] = fgetc(fp);
39   }
40  op::fclose(fp);
41  fputs(TTY_YELLOW "apps::image() finished\n", stderr);
42  return 0;
43  */
44 }
45
46 static uint8_t apps::image(struct dirent *ent)
47 {
48  /*
49  using namespace display;
50
51  fprintf(stderr, TTY_YELLOW "apps::image(): \'%s\'\n", ent->d_name);
52  FILE *fp = op::fopen(ent, "r");
53  if (fp == NULL) {
54   fprintf(stderr, TTY_RED "apps::image(): open file failed: %u\n", errno);
55   return 1;
56  }
57  for (uint_t r = 0; r < BUFF_H; r++)
58   for (uint_t c = 0; c < BUFF_W / 8; c++) {
59    buff[r][c][BuffRed] = fgetc(fp);
```

```
60      buff[r][c][BuffGreen] = fgetc(fp);
61    }
62  op::fclose(fp);
63  fputs(TTY_YELLOW "apps::image() finished\n", stderr);
64  return 0;
65  */
66 }
67
68 static uint8_t apps::animation(const char *path, const uint_t clr)
69 {
70  /*
71  using namespace display;
72
73  fprintf(stderr, TTY_YELLOW "apps::animation(): \'%s\'\n", path);
74  FILE *fp = op::fopen(path, "r");
75  if (fp == NULL) {
76   fprintf(stderr, TTY_RED "apps::animation(): open file failed: %u\n", errno);
77   return 1;
78  }
79  uint8_t cnt = 0;
80  int ch;
81  timer1::freq5();
82  while ((ch = fgetc(fp)) != -1) {
83   if (cnt++ % (30 / 5) == 0) {
84    while (!timer1::over());
85    timer1::clear();
86    cnt = 1;
87   }
88   for (uint_t r = 0; r < BUFF_H; r++)
89    for (uint_t c = 0; c < BUFF_W / 8; c++) {
90     if (r != 0 || c != 0)
91      ch = fgetc(fp);
92     buff[r][c][BuffRed] = 0;
93     if (clr & (Red << Foreground))
94      buff[r][c][BuffRed] |= ch;
95     if (clr & (Red << Background))
96      buff[r][c][BuffRed] |= ~ch;
97     buff[r][c][BuffGreen] = 0;
98     if (clr & (Green << Foreground))
99      buff[r][c][BuffGreen] |= ch;
100    if (clr & (Green << Background))
101     buff[r][c][BuffGreen] |= ~ch;
102   }
103   _delay_us(2000);
104  }
105  timer1::stop();
106  op::fclose(fp);
107  fputs(TTY_YELLOW "apps::animation() finished\n", stderr);
108  return 0;
109  */
110 }
111 #endif
112
```

```
static uint8_t apps::wav(const char *path)
{
 printf("apps::wav(): \'%s\'\n", path);
 FILE *fp = op::fopen(path, "r");
 if (fp == NULL) {
  printf("apps::wav(): open file failed: %u\n", errno);
  return 1;
 }
 int ch;
 pwm::enable(true);
 timer1::freq8000();
 while ((ch = fgetc(fp)) != -1) {
  while (!timer1::over());
  timer1::clear();
  pwm::set(ch);
  set_dac(ch);
 }
 timer1::stop();
 pwm::enable(false);
 op::fclose(fp);
 fputs("apps::wav() finished\n", stderr);
 return 0;
}
// Joes code

// file creation always fails
// existing files can be opened
// writing data always fails (returns EOF)

static uint8_t apps::wtxt(const char *path)
{
 printf("apps::wtxt(): \'%s'\n", path );
 FILE *fp = op::fopen(path, "w"); // create a new file in write mode at the
     specified path
 if (fp == NULL){
  printf("file creation failed\n");
  return 1;
 }
 else{
  printf("file open success\n");

  uint8_t temp = fputc('1', fp);
  if (temp ==0){
   printf("File Write Success\n");
  }
  else{
   printf("File write failed\n");
   printf("%d\n",errno );
  }
 //fputs("testing, testing\n", fp);
```

```
 op::fclose(fp);  // close the file
 }

 /* // iostream and fstream don't appear to exist for avr
 std::ofstream myFile;
 myFile.open("example.txt");
 myFile << "your text here\n";
 myFile.close();
 */
 return 0;
}

static uint8_t apps::rtxt(const char *path)
{
 printf("apps::rtxt(): \'%s'\n", path );
 FILE *fp = op::fopen(path, "r"); // openss file in read mode at the specified
     path
 if (fp == NULL){
  printf("file open failed\n");
  return 1;
 }

 else{
 printf("file open success\n");
 char ch = fgetc(fp); // produces ASCII code
 printf("data: %d\n",ch );

 }


}
#endif
```

# E Project Meeting Agendas and Minutes

25/02/17 15:00-15:35 All present (Joe, Ben, Mohammed, Lawrence, Joel)
Agenda

---

Discuss and assign subsystem roles.

Assign admin roles.

Discuss: - For the human controller will we use the Spektrum or our own?
- Chassis - Acrylic, aluminium or MDF?
- Extra features?
Plan for Monday:
- 150 word abstract
- High level block diagram

Minutes

---

Roles:
- Communications: Ground to air - Joel
Air to ground - Mohammed
Controller - Mohammed
- Control: Gyro/accelerometer interface - Joe
PID Controller - Lawrence
IR sensors - Ben
- Chassis Overall design - Joel & Lawrence
Cargo acquisition mechanism - Joe
- Power Management - Ben
- Integration - Ben
- File logging - Mohammed
- Soldering - Mohammed

Admin roles:
- Leader/Administrator - Ben
- Treasurer - Lawrence
- Communications officer - Joe
- Secretary - Joel
- Documenter - Mohammed

Decided to:
- Use our own controller.
- Laser-cut acrylic chassis.
- Extra features are an IR sensor for measuring height, downlink communications, remotely

tunable PID and security on the communications.
- Use 3 Il Mattos

26/02/17 18:15-18:45 All present (Joe, Ben, Mohammed, Lawrence, Joel)
Agenda

---

What needs discussing with Tim Forcer?
What do we need to bring?
Does the abstract need editing?
Is everyone agreed on the schematic diagram?
Use Github?
How are people getting on?

Minutes

---

Decided the specification part of the abstract could be skimmed down.
Agreed that the schematic diagram is complete.
Everyone happy to use Github.
Decided to use 2 transceivers on each end (i.e. two slaves connected to both Il Mattos). Extra slave of the SD card on the drone.

27/02/17 10:10-10:25 All present (Joe, Ben, Mohammed, Lawrence, Joel)
Agenda

---

What needs discussing at the clinic?
Does the abstract need more editing?

Minutes

---

To dicuss at the clinic:
- Simulation - what are they expecting?
- Control
- Thoughts on components
- Ask about marking - is more hardware design required?
- Is the frequency band of the wireless transceivers wide enough?
Request different frequency bands for the transceivers on the requisition form.
Abstract size needs reducing further

3/03/17 11:38-11:58 All present (Joe, Ben, Mohammed, Lawrence, Joel) with Rob Maunder
Agenda

---

Run through the project proposal form
Any other advice from Rob?

Minutes

Explained the block diagram:
- Focus on integration of blocks
- Explained our roles in terms of integration
- Power for the groun Il Matto needs to be considered
- Look at power distribution on the drone

Ensure we have distinct roles.
Risks:
- Have we considered if the Leonardo is powerful enough?
- Do we have enough knowledge of PID? We have found code that handles it.

Design completion milestones:
- e.g. for comms: 1. SPI interface 2. SPI transmitting, 3. SPI receiving 4. two way
- Rough guideline of one milestone per interface
- Milestones for different paths through the block diagram

9/03/17 13:02-13:19 All present (Joe, Ben, Mohammed, Lawrence, Joel)
Agenda

Plan for the rest of the construction time

Minutes

Work on critical path at the moment - add on extra features such as telemetry later on.
For the communications add more efficient comms scheme first then add the code to change k values remotely.
- K values vary from 0-10.23.

Chassis is glued - can work on any other stuff that needs doing (e.g. servo mechanism).
Finalise extra features by Friday.
Checked that we will all have enough to write on the report.
Get the ESCs calibrated and get the critical path done today.

10/03/17 17:49-18:01 All present (Joe, Ben, Mohammed, Lawrence, Joel)
Agenda

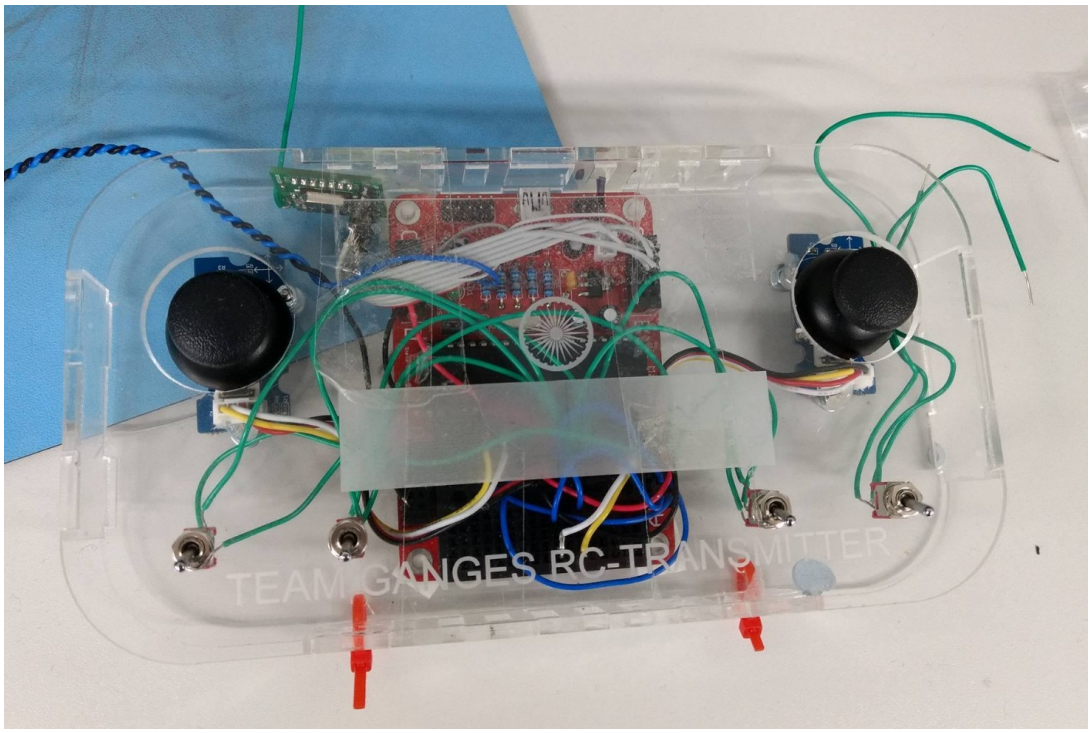
Plan for the last weekend of construction

Minutes

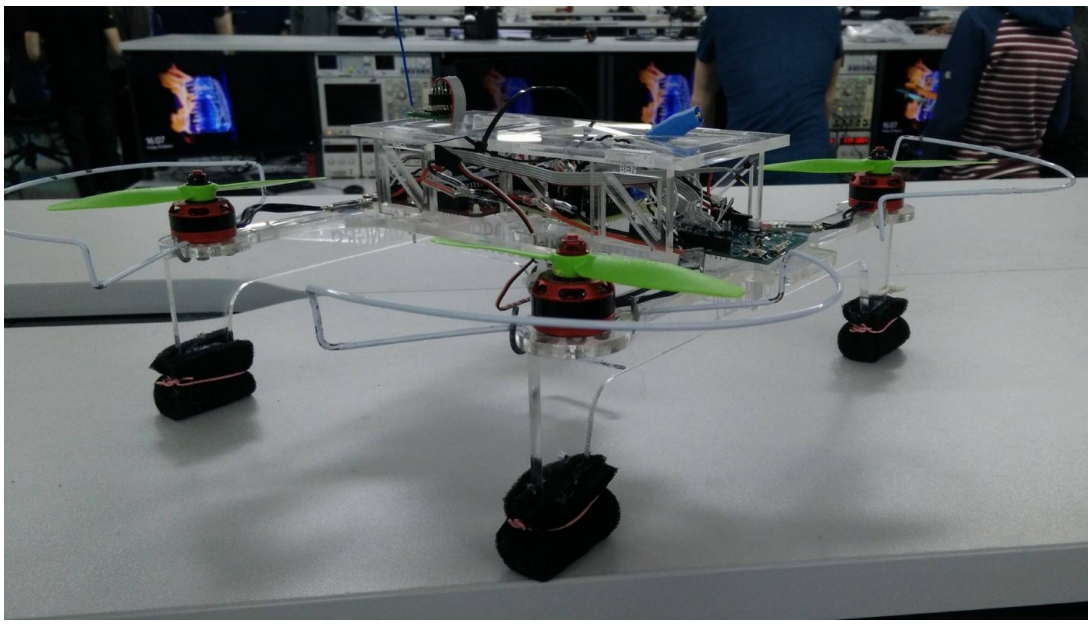
Mohammed and Joe are away - give each other our numbers to allow quick contact if required.
Joel, Lawrence and Ben will do the PID tuning over the weekend.
Bring a 12v power supply with a max current of 4/5A and a positive centre if possible.

# F Images

Figure 1: Final product images




(a) Final build of the Pilot-Drone interface. Controller electronics inside an acrylic casing.



(b) Final build of the Quad-copter UAV consisting of our communications and control systems, placed inside our chassis.