D4 Individual Report

Joseph Hindmarsh

## Contributions

During the initial design meetings of the first few days of the project, design responsibilities of subsystems were split up amongst the team. I was tasked with the interface of the gyroscope/accelerometer unit with the micro-controller, the payload system and integrating the two microcontrollers together. By the end of the project I had set up the gyroscope to send values to the Arduino Leonardo; designed and produced the motorised cargo hook and begun investigations into the use of an SD card to log flight data.

## Specification

For the gyroscope and its interface, I was trying to design a system that could produce angles at a rate of 100 readings every second, having converted them from angular velocities into offsets from the vertical.

For the payload system, the hook had to keep the cargo secure in flight with pitches of up to 10 degrees. With the hook at its lowest position and the UAV was in level flight, the ring mounted package also had to be able to slide off. This meant a range of movement of at least 45o.

The specification for the SD card interface was that it should be capable of logging at a rate of 1 packet/second, where the packets would consist of the battery level and the gyroscope data. As the data logging was relatively low priority compared to the critical path of data from the receiver to the PID controller, a fast protocol was required to keep the overheads as low as possible, taking at most 10 milliseconds to log a packet.

## Design

### Gyroscope

The Gyroscope was the first task on the list to be designed, as it was a key part of the core flight systems of the UAV. The gyroscope module chosen was the MPU-6050, as it is a common choice of IMU (Inertial Measurement Unit) for low power applications, and therefore is well supported with software libraries especially among the Arduino community.

To make the budget stretch further additional cost savings were considered with the option of just buying the IMU and constructing a breakout board from stripboard, but the IMU was only available from the suppliers in QFN (Quad Flat No-Lead) packages. This would have required a separate pinout board and learning the new soldering techniques to solder them, and so the idea was rejected as too risky for only a small cost saving of £2.

The I2C protocol is a common interface for sensors, as often in electronic systems where they are deployed there will be a variety of sensors for the microcontroller to communicate with. I2C uses a 2-wire bus (one clock line, one data line) regardless of the number of devices in the system, making it an excellent choice for systems with many inter-connected devices [7]. It was not ideal for the UAV project as there were only two devices on the bus – and so sending a slave address for every read is unnecessary overhead for the communications. This was the only communications protocol available on the selected sensor, although this was alleviated by the register ordering. The gyroscope registers are all sequential [6], which allowed all the required data to be read in a single ‘Burst Read’ that only requires the acknowledge/not acknowledge handshaking between each sequential byte read rather than needing to resend the slave address byte every time [5].

The initial design for the interface was based around communicating with an Il Matto microcontroller board, using an I2C library [1] to attempt to open communications with the gyroscope. The test program “i2c lib test\main.c” produced only zeros upon reading the gyroscope registers. Looking further into the library, it was apparent from the function mpu6050\_init() on line 459 of AVR-MPU6050-lib\src\mpu6050\mpu6050.c that the IMU had to be initialised on startup to set up the clock and the sensor ranges. It was at this point that the microcontroller connected to the IMU changed to an Arduino Leonardo for access to more 16-bit timers required for the ESC control. As the library being investigated was a port of the Arduino I2C sensor library [2], this was not a large change to make for this subsystem. Lines 133-323 of the [8] included the configuration for the on-board DMP, allowing the processing of the raw angular velocity values to be converted to absolute offsets from the origin without any processing overhead on the microcontroller which would have increased the control loop time considerably. The DMP was set up by writing a large, sparsely documented block of memory to the MPU6050s register banks, making it a task out of the scope of this project to analyse and understand. The other core difference in the code from the microcontrollers perspective was that instead of polling the data registers from the sensors it switched to an interrupt based system with the IMUs internal FIFO (first in, first out) buffer, with an interrupt firing every time new values were available. This improvement avoided slowdown of the control loop at the cost of negligible delays caused by the ISR (Interrupt Service Routine) setting a bit flag 100 times/second.

### Cargo Hook

Two initial concept ideas were proposed for this design, a servo -actuated hook and a rack-and-pinion style bolt. The bolt based design would have required both a large cavity in the chassis and a more difficult cargo pickup procedure, and so the idea was dismissed as impractical.

The motorised hook was designed to avoid stressing the servo while in flight, as that will be one of the most fragile parts of the design. To achieve this, the hook was not designed to screw onto the servo, but to instead pivot around a hole in one of the main chassis support struts. This means that the weight of the payload is on the centre of the chassis, rather than hanging off-centre from the servo gearing.

The initial design consisted of the hook linking in directly to a servo horn, but this was soon discarded when a quick mock-up of the mechanism from cardboard showed that this would give the hook no freedom of movement, with the servo horn being of a fixed radius. This meant that a slot to allow one of the hook axles to move within it was required. There were two options for the placement of this slot, either in the chassis strut or attached to the servo horn. Using a horizontal chassis mounted slot would not provide the servo with as much leverage and it would have been easier to jam or move in an unpredictable direction with the hook being pulled down into the side of the slot. This meant that the design shown in figure 1 was a better configuration of the slot. With the servo mounted slot position decided, it was just down to the design of the hook itself. The materials chosen for the hook was 2mm coat hanger wire as the wire was thick enough to be rigid under load. The slotted servo horn was made from 3mm thick acrylic as it wasn’t a structural part but merely a guide for the hook to follow. Laser cutting made the machining of this complex shape straightforward.

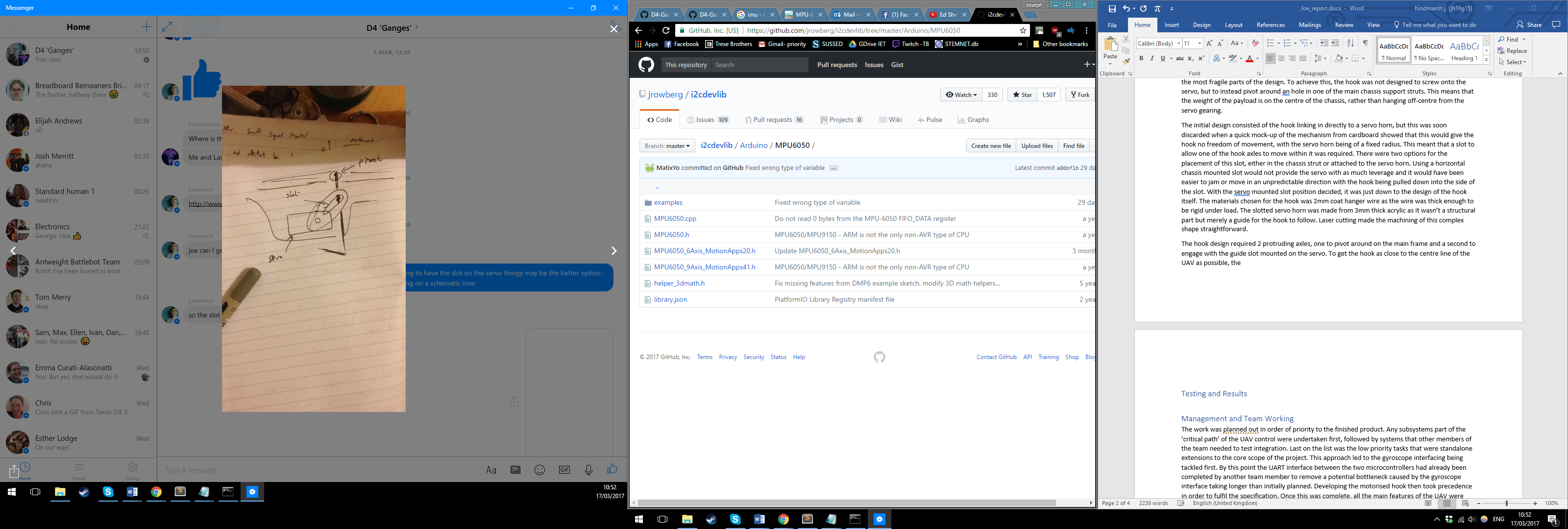


Figure 1. A sketch of the hook mechanism

The hook design required 2 protruding axles, one to pivot around on the main frame and a second to engage with the guide slot mounted on the servo. To get the hook as close to the centre line of the UAV as possible, the hook was sandwiched between the servo horn and the chassis. The initial plan was for the guide slot to rotate between angles of 0o-90o from the vertical, but testing with a cardboard mock-up of the chassis revealed that it would interfere with the chassis base and jam up when trying to raise the hook. After this testing the design was modified to have the servo rotate between 90o and 180o from the vertical to minimise this issue at the cost of making it harder to pick up cargo as the guide slot was then partially obstructing the hook while it was in the lowered position. This however was determined to be a worthwhile compromise to reduce the chance of the hook jamming.

### SD Card

As an extension to the project that was only started in the last few days, not much design work could be done. The SD card library found for the Il Matto used a very confusing makefile system as part of a larger library, and as such working out how to successfully compile the library with the provided example code. After this had been achieved, the example program was tested for functionality. There was no support for text files included with the library, and so ‘apps.h’ was extended to add this functionality (lines 136 - 194). Upon testing the new code with a freshly formatted SD card it was not able to create any new files. If the file already existed if could open it, but returned an EOF (end of file, write failed) character when text was written using the ‘fputs’ function from the included AVR library <stdio.h> [3]. The ‘fgetc’ function worked as expected, returning the ASCII codes from text files placed on the SD card by a PC. Further investigation into file.cpp [4] showed a function for fgetchar but not fputchar, which may have been the cause of the functions for writing data not working correctly.

## Testing and Results

### Gyroscope

The gyroscope system was tested in several ways. The initial testing method was to use the serial monitor from the Arduino IDE to display the yaw, pitch and roll values as the sensor was moved. This was valuable for showing that the basic functionality of the system was correct and fulfilled the specification. There were small variations of about 0.1 degrees with the sensor flat on the table, but this margin for error was deemed to be manageable. During this testing phase an oscilloscope was used to probe the interrupt pin signalling when fresh data was ready. This was found to be a steady 100Hz normally-high digital signal with a low-time of 50us when the interrupt was triggered. The frequency of interrupts was not affected by any movement of the gyroscope, so the data rate was constant.

The second stage of the testing was integrating the gyroscope together with the PID controller on the Leonardo. A breadboard with the IMU was attached to a servo controlled by the PID controller. The servo was rolled around its axis and the position of the breadboard observed. After a little tuning, the IMU could be kept upright successfully regardless of the speed that the servo was moved with. This was repeated with the DMP set to a 200Hz frequency, however despite the interrupt signal being a constant 200Hz the angle data from the IMU was noisier and was detrimental to the stabilisation of the breadboard.

### Cargo Hook

The cargo hook was tested first in a cardboard mock-up of the chassis to check that clearances were sufficient before wiring up the servo to a function generator with PWM capabilities. This enabled the appropriate values for the duty cycle to be found through a quick sweep and establish the range of movement that could be expected from the hook once mounted. These produced values of a 12.8% duty cycle for the cargo hook to be in its lowered position and the minimum duty cycle for the hook when raised. Moving to the actual frame, the increased thickness of the frame as compared to the cardboard reduced the movement of the hook in its socket. This led to much less freedom of movement for the hook, from about 45o to only about 50-100, as the servo was not powerful enough to overcome the increased stiffness of the mechanism. The integration with the rest of the system was tested through testing if the switch on the controller could move the hook. Although the hook only moved about 10o between configurations, the integration of the system was successful.

### SD Card

This optional subsystem was not completed in time for testing to be performed.

## Management and Team Working

### Personal Management

The work was planned in order of priority to the finished product. Any subsystems part of the ‘critical path’ of the UAV control were undertaken first, followed by systems that other members of the team needed to test integration. Last on the list was the low priority tasks that were standalone extensions to the core scope of the project. This approach led to the gyroscope interfacing being tackled first. By this point the UART interface between the two microcontrollers had already been completed by another team member to remove a potential bottleneck caused by the gyroscope interface taking longer than initially planned. Developing the motorised hook then took precedence to fulfil the specification. Once this was complete, all the main features of the UAV were either completed or in development by other members of the team. This led to the start of investigation into the additional extension features such as the introduction of an SD Card to the il Matto for additional onboard logging.

### Team Management

Team Ganges was managed effectively and flexibly, enabling the team to react to new events and challenges whilst keeping the project on schedule. Each team member was assigned an additional management-focussed role alongside their main design responsibilities to spread the load. These additional roles were:

* Team Leader: Ben Rowlinson, managed the overall team progress, priorities and performance. Set the agenda for meetings.
* Secretary: Joel Trickey, took minutes in meetings
* Financial Manager: Lawrence Gray, worked out costings and managed the budget
* Stores Liasion: Joseph Hindmarsh, managed interactions with the lab staff
* Documentation: Mohammed Ibrahim, organised the Git Repository for document sharing and code collaboration

It was decided at the outset of the project that a daily meeting structure was the most effective way of ensuring that all team members knew how the project and its subsystems were progressing. This frequent interaction ensured that interfacing between subsystems would be as seamless as possible and enabled the team to flexibly reassign resources as needed to deal with unexpected issues an. An example of this would be the chassis fracturing during testing on the final weekend of the project. Improvements to the chassis were designed while the damaged chassis was strapped up to allow continued tuning of the PID controller.

Work was initially split evenly amongst the group based on a brief skills audit of the members, although this did change slightly over the course of the project to keep all team members with tasks to work on. The work was planned using the high-level block diagram to establish the critical path of the systems – mainly focussing on the communications links between the various microcontrollers. Assigning all team members to a section of this critical path ensured that the core functionality of the UAV would be operational as quickly as possible. This meant that work on integrating the work across the team could begin early on to allow time for solving any compatibility issues that arose between subsystems.

## Critical Evaluation and Reflection

Although the final submitted prototype UAV cannot achieve stable flight, this was due to the PID tuning not being completed in time for final testing before the submission deadline rather than any electronics or programming not functioning or being incomplete. We knew that the PID tuning would take time, but didn’t count on wasting 8 hours of the final weekend due to uncalibrated ESCs. This level of delay could not have been anticipated by the project manager and occurring too late on in the project for our coping strategies to have an effect.

Overall the team worked well together.

With regard to my own contributions, I would say that from a programming perspective they were limited. My initial approach to the gyroscope interface was flawed as I wanted to experiment with a communications protocol to see if I could interface with the IMU without a dedicated library. Although the I2C knowledge I gained was interesting, due to the very time intensive nature of the project my time would have been better spent looking at the MPU-6050 specific library to get something workable as quickly as possible. This was made apparent by the code I had written being unable to read data correctly or access the DMP as there were internal registers that needed to be configured before any non-zero data could be read.

Nearly all the rest of my time working on the software was based around attempting to understand device libraries, which would often not compile easily. Because of this my limitations from a lack of understanding of makefiles led to further delays as I tried to work out the correct command to use the provided makefiles. The libraries that I selected to use for the project were large compilations of device libraries for either the Arduino platform or generic AVR microcontrollers, so most of the packaged library was unnecessary. I found the use of generic makefiles with definitions to be very confusing I wasted a lot of time before resorting to manually compiling the required code file by file.

My design of the motorised hook was flawed by lack of true consideration of the materials involved. Making the hook from a thick coat hanger was fine for maintaining its shape under load, but this same property led to it being difficult to work with when bending out the axles for the hook to rotate around. These were not perpendicular to the rest of the hook and caused the hook to jam frequently after only a short distance, not fulfilling its specification of a full range of movement - when mounted on the chassis, the hook could neither retract enough to drop off a load while stable nor properly secure the payload for the full range of angles.

## References

1. <http://davidegironi.blogspot.co.uk/2013/02/avr-atmega-mpu6050-gyroscope-and.html>
2. <https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050>
3. <http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__avr__stdio_1ga19c2bbe9ce4af9f0a7e3448387004fd3.html>
4. <https://github.com/zhiyb/Il-Matto/tree/master/SDCard>
5. <http://www.daedalus.ei.tum.de/attachments/article/57/PS-MPU-6000A.pdf>
6. <https://www.olimex.com/Products/Modules/Sensors/MOD-MPU6050/resources/RM-MPU-60xxA_rev_4.pdf>
7. <https://secure.ecs.soton.ac.uk/notes_so/elec1201/Lectures/SerialCommunication.pdf>
8. <https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/MPU6050_6Axis_MotionApps20.h>