D4 Individual Report

by

Joel R. Trickey

Team Ganges

jrt1g15

27570541

16 March 2017

Contents

1. Contribution

As part of Team Ganges I was responsible for designing and implementing a communications system to enable controls and other data to be sent to the drone. This was achieved using RFM12B RF transceiver modules and two Il Matto micro-controllers. The majority of my work consisted of C code development for the drone Il Matto to allow interfacing with the transceiver and processing the data to be passed on to the on-board Arduino. This system was successfully developed and utilised as an integral part of the project.

In terms of admin I was tasked with documenting and taking minutes for team meetings.

2. Specification

The system I was attempting to design sent potentiometer and switch values to the drone to allow control of it. To allow the controller to be responsive for the pilot, it was required that the Il Matto in the air was able to receive all four potentiometer values at a rate of 100Hz (i.e. rate of 4*100=400Hz). As the data read from the ADC would be a 10-bit value, the communications would have to be capable of sending data packets of this length.

As well as sending data to the UAV, telemetry data would be sent back to the base at 1Hz. This data would consist of the battery level, gyroscope angles and the height of the drone from an IR sensor.

The range was required to be sufficient to allow control from around 10 meters.

Once complete, this system could be extended further with some sort of security feature (e.g. encryption of transmitted data) that does not severely affect the transmission rate.

3. Design & Simulation

The subsystem that I was responsible for designing was the communications from the base to the drone (the 'uplink'). This system would send data collected from the HID controller with the user to the drone, allowing it to be controlled from the ground. This includes code to interface with the RFM12B-S2 transceivers that we planned to use and process data that was received and ensure that is dealt with correctly.

As I had experience using the Il Matto micro-controller previously, this made it a natural choice for dealing with communications on the drone. From here it was then a matter of researching the RFM12B and gaining an understanding of how it worked. This led me to a free-to-use library [1] that allowed straightforward use of an RFM12B with an Atmel AVR micro-controller.

This only required setting up the source files correctly and calling a few functions to send and receive data.

3.1 Initial Uplink Code

After reading through the documentation and gaining a good understanding of how the library was used, I drew up some initial code for the Il Matto to implement basic transmission and receiving of data

(basestation_comms.c and Uplink/drone_comms.c).

First, the library was set up as suggested in the documentation (downloaded from [1]) with an "rfm12_config.h" file that allowed the settings of the library to be tweaked and an "src" folder with the core library files. In my code, the wireless modules were initialised and then set up to send or receive using the functions provided with the library. They were set up so that the basestation would send values from 0 to 1023 (all possible 10-bit values) to the drone. They were then compiled to ensure correct configuration.

To make the code more readable and more easily extended later, I created a header file (comms.h) that defined a set of op codes to be sent as the packet type to allow both micro-controllers to identify the data that is sent and received. Initially, this included codes for the four potentiometer values (thrust, roll, pitch and yaw), button presses and sending three new K values to the controller. It also contained macros that allowed various options to be enabled/disabled (such as encryption and the uplink test) without adding lots of extra code. As the code grew new codes and macros were added.

The library sends an 8-bit packet type to identify what is being sent, followed by the data in a discrete number of bytes. As 10-bit values were being sent this would cause 6 bits of the data to be wasted, leading to less efficient data transfer.

To optimise this, I employed an encoding in which the 'packet type' would send the extra two bits of data, leading to only one data byte being sent for one ten-bit value instead of two. The two most significant bits of the 10-bit ADC reading were placed in the two least significant bits of the packet type with the required code being left shifted by two bits (Figure 1). The "Encode_data" function (basestation_comms.c lines 72-79) implements this by simply converting the total 16-bit packet, with ADC reading and op code concatenated, to two 8-bit variables. After being received, this is then decoded back into the op code and ADC reading using "Decode_data" (uplink/drone_comms.c lines 62-69).

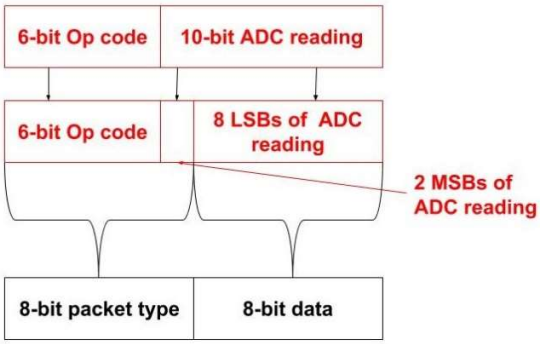

Figure 1: Method of encoding data to maximise efficiency

Now that the main code for the transceivers was in place, I started designing an encryption method that would enable packets to be sent securely, to and from the drone. As there were three bits free in the packet type, the initial design used these bits for an encryption key. A simple algorithm was implemented in which the 13 bits (excluding the key) are rotated right by the number in the three encryption key bits (3 most significant bits in the packet type). The receiver then decrypted the packet by rotating the 13 bits back again. This encryption code is implemented in the "Encrypt_data" and "Decrypt_data" functions (basestation_comms.c lines 86-105 and Uplink/drone_comms.c lines 71-88 respectively).

3.2 Double transceivers

Due to the high data rates demanded by the specification and the ability to adjust the radio frequency used by the RFM12B modules, it was decided that two transceivers could be used on each end to allow two different channels for uplink and downlink that could transmit in parallel. In order to achieve this, I adapted the library to communicate with two slave SPI devices – one that dealt with all of the

transmission and another that dealt with the receiving.

This code successfully compiled and was ready for testing with the transceivers but did not make it into the final design due to the system showing sufficient performance without adding a second channel.

3.3 Processing uplink data

Following its completion, I added the uplink program to a new folder and extended it with code to collect control data at the base and process received data in the drone. Mohammed was responsible for the former and myself for the latter. A series of test files (test1_drone_comms.c – test3_drone_comms.c) allowed us to slowly add more features into the system and test them before implementing them into the main system.

The first element to be added was to send potentiometer data and assign it to a variable in the receiver according to what type of data had been sent. This simply required a switch statement for the packet type received on the drone.

Next, K values to tune the controller were processed. This required converting the received data back into the original decimal number – as the range was 0-10.23 this was as simple as dividing the number by 100.

To send telemetry back to the base once a second, as per the specification, this was added (test3_drone_comms.c lines 184-193) with a simple counter system to send data back after 150 packets had been received from the base. Later, this was swapped with a timer to allow more consistent telemetry even when fewer packets were being received. With the library dealing with most of it this was as simple as calling the "Send_data" function and 'ticking'.

Finally, the switches were implemented. There were three switches that controlled the cargo hook servo, killed the motors and switched between two modes for flying or adjusting the K values. To reduce the number of packet type op codes that were required, the switches used the "OP_BUTTON" op code with different data for each switch state (comms.h lines 32-37). The switch for flight mode and PID mode determined whether telemetry was disabled and whether K values would be processed.

After all the above features were added, the number of bits dedicated to the encryption key had to be reduced to allow more op codes to be introduced. Therefore, the encryption method had to change to avoid the use of bits dedicated to the encryption key. Instead, I changed the algorithm to use the first four bits of the data as an encryption key, these bits would remain in position while the rest of the total packet was rotated right. This has the added advantage of being more random than a repeating sequence of numbers.

3.4 Combining tested code

These features were tested thoroughly before being fully integrated into a new file – drone_serial_comms.c. Here, test code was removed and functionality from other members of the team was inserted. Primarily, Ben's code to send data to the Arduino was integrated with the communications code from the previously mentioned programs (test1_drone_comms.c etc.). This enabled data to be sent from the Il Matto to the Arduino over UART when requested with an interrupt.

For the telemetry, Mohammed's ADC code was added to read off IR sensor and battery voltage and these

values were sent back to the base. As part of this, a small bit of code was written to flash an LED twice a second when the battery level drops below a certain level. This did not make it into the final system due to more time-critical design tasks.

The cargo hook control was implemented by calling Ben's "pwm_duty" function (drone_serial_comms.c lines 386-392) with either SERVO_PWM_DUTY_MIN or SERVO_PWM_DUTY_MAX depending on whether it needed to go up or down respectively.

4. Testing & Results

4.1 Encoding and Encryption

I tested both the encoding of the 10-bit values and the encryption in the same way – by encoding/encrypting every possible value then decoding/decrypting it and breaking from the test if this was incorrect (encode_test.c). This allowed the test to be automated, enhancing its speed and coverage. After a few corrections, this passed successfully.

4.2 Uplink test

To test reliable transmission from one Il Matto to another, I added some test code to the initial uplink program (Upink/drone_comms.c and basestation_comms.c) that sent all possible 10-bit values (0-1023) in sequence before stopping, with the results printed to a PC via UART. The transmission of all 1024 values was timed with a stopwatch to gain a rough maximum rate at which the potentiometer values could be acquired. The outcome of this test gave us (me and Mohammed) an upper limit of ~200Hz. There were minimal delays introduced (none from reading ADC values etc.) at this stage; it was therefore clear from this test that a rate of 400Hz may not be feasible if potentiometer values are read and sent at a constant rate.

This test also showed when packets of data were being lost and allowed us to tune the system to maximise data rate and minimise packet loss. When packets were sent at the fastest possible rate with no delays a lot of packet loss was experienced. To remove this, we introduced delays between sending different packets to ensure that the previous data was given sufficient time to be transmitted. The length of this delay was tweaked to achieve the fastest possible transmission rate with no packet loss.

Even with this tuning an issue remained where every other packet would not be received, regardless of the delay time, and despite further investigation of the library, we were unable to locate the source of the problem. As a stop-gap solution every packet was sent twice which gave us reliable transmission without any data being lost.

Overall, this test showed that the system worked but not quite to the required specification and with some unknown behaviour. With the temporary solution, the communications did however transmit reliably and at a fast rate.

4.3 Test 1 – Uplink with potentiometer ADC (test1_drone_comms.c)

The next feature to be tested was reading, sending and receiving potentiometer values. In the drone this involved simply printing out the received data to UART based on which control was received. We then

speed tested this in a similar way to the previous test – the number of packets received in ten seconds was measured and then used to calculate an estimate of the transmission speed. This test passed with little/no effect on the data rate. As found previously, the packets for roll, yaw and pitch were lost without sending twice with a 2ms delay between each. This solution is not ideal but made the system reliable without losing speed.

4.4 Test 2 – Tuneable K values (test2_drone_comms.c)

Following the successful test of the potentiometer values, a user interface was added to the ground Il Matto by Mohammed. In turn, I coded the drone to convert the K values it receives back into the original floating point number and print it off, along with the parameter that was being changed (lines 77-127).

At first this test only printed out question marks instead of float values, but after researching using lab notes from year 1 [2] I found that this was due to some missing compile command options. Apart from this, the system correctly received lots of different values for various parameters. As we had decided that this would not be done when the drone was in the air it was not necessary for the potentiometer values to be received at a fast rate (or at all) while the K values were being changed and therefore we did not need to test the speed with this new feature.

4.5 Test 3 – Switches and telemetry (test3_drone_comms.c)

Finally, controller switches and telemetry being sent back to the base was tested. For the switches, I added a few print statements to allow us to check if the commands were being correctly sent, received and interpreted (lines 94-123).

Using a counter, the drone sent telemetry that consisted of some random data after every 150 packets that it received. The intention of this test was to ensure that data could be sent back at least once per second without slowing down the uplink. At first, very few packets were successfully sent back to the base but after some debugging I found that the drone program was being bottlenecked by printing out all the potentiometer values it received over UART. After these print statements were removed, the telemetry was sent back more successfully but still at an intermittent rate. As the received values were no longer being printed it was also hard to tell if the potentiometers were still being transmitted reliably.

The slow telemetry was fixed by sending it at a faster rate of ~2Hz. This was achieved by using the on-board timer to signal when roughly half a second had passed and therefore send telemetry back to the base. Using the timer also increased reliability especially when a more efficient transmission method (only sending potentiometer data when the values change) was employed.

5. Team Working

At the beginning of the project Ben volunteered himself and was chosen as the team leader. The team was very well managed: Ben kept track of everyone's progress and gave us well-defined jobs.

In order to manage expertise and distribute the workload, we split up into two sub teams – control and communications. Me and Mohammed were part of the team that designed and developed the communications system and the base controller. Within our sub-team we then divided up the tasks further, I was specifically given the task of developing the uplink (ground to air communications).

When it came to documentation and other admin duties we were assigned tasks by Ben. I was charged with documenting the agendas and minutes for every team meeting that we had over the course of the project.

As a team, we often kept in close contact which allowed us to consult with one another on overlapping parts of our design. Throughout development and testing, I worked closely with Mohammed to more quickly discuss ideas, debug issues and evaluate the results of our tests.

Thorough research into the components we were going to use not only reduced development time but it also enabled us to reject parts that would not give us the desired functionality/performance. This greatly reduced unexpected risks. This is what led us to use an Arduino rather than an Il Matto for control and to avoid the Spektrum controller. Despite our best efforts we still ran into a few unexpected issues such as a constant backwards pitch during flight and packet loss in the communications. For the more critical sub-systems we all worked together to find a possible solution more quickly.

6. Critical Evaluation & Reflection

The communications system that I was heavily involved in designing worked extremely well - it was reliable, especially for long days of tuning the PID control and despite it not meeting the specification for speed set at the beginning of the project, it was more than sufficient for requirements in the end. It was also extended beyond the criteria with security from encryption and telemetry for the battery level and IR sensor.

On the other hand, I was unable to find the source of some of the errors that I ran into with the communications and the solutions that were employed were not satisfactory. Also, being more systematic in my approach to development and keeping better track of what I had done throughout would have aided my debugging. My work with Mohammed could have been more clearly split up to allow us to do more work individually.

I also did not contribute to the chassis design despite being assigned that role with Lawrence, I could have been more involved in this area of the project.

Bibliography

[1] Das Labor. *RFM12 library/en,* wiki.das-labor.org. [Online]. Available: https://wiki.das-labor.org/w/RFM12_library/en . [Accessed: Feb. 26, 2017]

[2] Steve R. Gunn. *C9: Analogue Input.* secure.ecs.soton.ac.uk. [Online]. Available: https://secure.ecs.soton.ac.uk/notes/ellabs/1/c9/c9.pdf . [Accessed: Mar. 8, 2017]