UAS Technology

UNIVERSITY OF SOUTHERN DENMARK

Notes and findings on eleventh laboratory exercises

Frederik Mazur Andersen
fande14@student.sdu.dk

Kristian Møller Hansen
khans13@student.sdu.dk

Olliver Ordell
olord13@student.sdu.dk

Hand-in date: May 7, 2018

1 Introduction

This journal will shortly describe findings and what has been chosen to get the SDU UAS TX running with full stick response and remote control. The code will be put in separate files.

2 Audio stick - Flight Simulator

The audio-wire for the simulator had to be rewired so it went from stereo plug to stereo plug instead of mono to stereo. We did have it working without the rewiring as long as a computer with dedicated microphone port was used. But to get it working on computers with dual headphones and microphone plug it required to be rewired.

3 Full mapping of the sticks

To get the controller fully functional it was necessary to make some changes in the firmware to get full throttle response and to map the switches. This section will cover said changes.

3.1 Full response from joysticks

The hardware potentiometers doesn't give full response from 1150 ppm to 1850 ppm, so it was required to correctly scale the analog input to be in the that range. This was done by measuring the actual range of the potentiometers and correcting for that so the input range maps to the desired output range. The potentiometer was also used to change the output range of the pitch and roll axes, so the response could be weakened.

3.2 Switches and their mapped functions

Per standard the left 3-way switch is mapped to armed, non armed. The right 3-way switch is mapped to three different flight modes, being `angle, horizon` and `rate` modes. The right 2-way switch was changed to change between remote control from serial to manual control. This was a quality-of-life change to quickly change over to test features safely.

4 Remote control with computer though rosnode

4.1 Transmitter to rosnode interface

The interface with the rosnode is done over serial connection though a FTDI cable. The chosen format is a Json string as seen in code 1. The values is given in ppm.

This is chosen for easy implementation and handling. It also ensures that the values only get read if the deserialization of the Json string succeeds. In that way values that get scrambled up while transmitting won't get sent to the drone. The transmitter uses the Arduino library, ArduinoJson.[1] This supports reading and deserialize Json strings directly from a stream. It just takes the serial stream as an object and then continuously read from it. This meant it was possible to achieve speeds over 80 hz.

When the same was attempted with first reading line from the serial and then parse into the Json deserializer it made many errors and often cut off the strings. Parsing directly from the stream resolved that issue.

The code in the transmitter to handle this part can be seen in the file, or in the github repo[1].

```
{
    "t": throttle,
    "r": roll,
    "p": pitch,
    "y": yaw
}
```

Code 1: The Json message format used to send data over serial connection.

4.2 Rosnode

The rosnode subscribes to a topic called `"/remote_control/set_controller"` with the message type seen in code 2. The floats are percentages. The rosnode then checks the message and clamp signals that is outside the range, so it isn't possible to send values like 200 %. The message then gets converted to ppm based on the input percentage. Throttle goes from 0 % to 100 %, while the other channels goes from -100 % to 100 %, where the midpoint is 0 %. The midpoint matches 1500 ppm. The message with the ppm values gets serialized into a Json string which gets sent to the transmitter over serial connection. When this node is running, the arduino only functions as a relay to the transmitter, which means that more advanced features like changing the sensitivity or other behaviors of the controller, can be baked into the node to make it more easily configurable from the outside.

```
# This contains the percentage of stick movements for the channels
float64 thrust
float64 roll
float64 pitch
float64 yaw
```

Code 2: The message format used to send data to the rosnode that bridges the data into the transmitter.

4.3 Virtual Joystick

A quick rosnode that listens to keyboard events was made for controlling the drone though the keyboard. It listens to keyboard arrow keys and the `w,a,s,d` keys. Then it increases the percentage of the channels depending on what keys you press and constantly transmits it to the `"/remote_control/set_controller"` topic. The controls are described in table 1.

[1] `https://github.com/Crowdedlight/SDU-UAS-TX/blob/remotecontrol/python_controller/python_controller.ino#L188-L220`

Key	Action
w	Increase throttle
s	Decrease throttle
a	Decrease yaw
d	Increase yaw
up-arrow	Increase pitch
down-arrow	Decrease pitch
left-arrow	Decrease roll
right-arrow	Increase roll

Table 1: keyboard controls for the `remote_control_keyboard` rosnode that enables flying the drone using the keyboard.

5 Spektrum Remote Receiver

To read to serial data from the receiver, we had to modify the Arduino script[2]. Originally the RX input would interfere with the serial communication over USB, meaning that nothing would be read. The signal was sent from the receiver, but something internal to the RX pin made the voltage drop to a level that was unreadable to the arduino as soon as the connection was made to the receiver; pull-up or pull-down resistors did not get rid of this problem. To fix this, we selected two other pins (10 and 11) as serial pins via [2]. After doing so, we were able to read data from the serial port, using this python script[3]. On figure 1 a screenshot of the received data. It was very difficult to make sense of the contents of the message since everything kept being overwritten and the addresses seemed to be all jumbled together, but we were able to locate some of the ppm signals sent from the arduino on addresses 0 through 6.

[2] `https://github.com/SDU-UAS-Center/SPM9645-Arduino-Bind/blob/master/spektrum_spm9645_bind/spektrum_spm9645_bind.ino`

[3] `http://frobomind.org/web/doku.php?id=uas:c2:spektrum_dsmx_remote_receiver_parser`

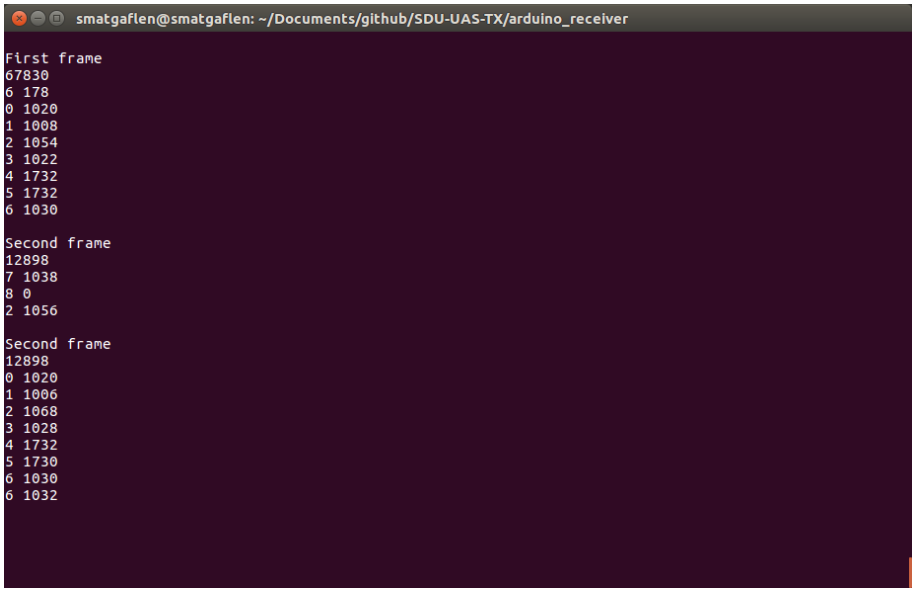

Figure 1: This is the output from the python script. The addresses 0 to 6 seem to be the values transmitted for the joysticks and switches on the controller, but the overall structure is hard to read.

6 References

[1] ArduinoJson. *ArduinoJson*. 2018. URL: `https://arduinojson.org/`.

[2] Arduino. *SoftwareSerial Library*. 2018. URL: `https://www.arduino.cc/en/Reference/SoftwareSerial`.