**Technical design ROC Ter Aa**

**Project: FlightSim**

**Client: ROC Ter Aa**

**Project number: 00004**

**Author: Giel Willemsen**

**Date:** <Geef de datum op wanneer het document opgesteld is.>

**Version: 0.1**

The undersigned declare their agreement with the content of this functional design.

**Client Project manager**

***Initial seeen: Initial seen:***

Date: <Geef de datum op.> Date: <Geef de datum op.>

Place: <Geef de plaats op.> Place: <Geef de plaats op.>

Content

[Introduction 3](#_Toc24616087)

[Technical requirements 3](#_Toc24616088)

[FlightGear data IO 3](#_Toc24616089)

[The Forwarder 3](#_Toc24616090)

[FlightGear XML 4](#_Toc24616091)

[Speed meter 5](#_Toc24616092)

[Artificial horizon 5](#_Toc24616093)

[Heading 6](#_Toc24616094)

[Altitude meter 6](#_Toc24616095)

[Climbing meter 6](#_Toc24616096)

[ADF 6](#_Toc24616097)

[RPM meter 6](#_Toc24616098)

[Throttle 7](#_Toc24616099)

[Brake 7](#_Toc24616100)

[Flaps 7](#_Toc24616101)

[Steer 7](#_Toc24616102)

[Pedals 7](#_Toc24616103)

[Carb heat & mixture lever 7](#_Toc24616104)

[Reset button 8](#_Toc24616105)

[Switches (not the levers) 8](#_Toc24616106)

[About this document 8](#_Toc24616107)

[Abbreviations 8](#_Toc24616108)

[References 8](#_Toc24616109)

[Definition 8](#_Toc24616110)

[Used materials 8](#_Toc24616111)

[Appendix 9](#_Toc24616112)

[APPENDIX B Agreement 10](#_Toc24616113)

# Introduction

This is the technical document for FlightSim. This is a project that is going to require hardware as well as a lot of software knowledge. This is because all the wiring has to be redone and thus all meters and inputs need to be reconnected. This document will be describing what the new hardware layout is and why something is connected the way it is. But also, how the software is going to be controlling it. This does require that the reader (you) need to know the basics about electronics hardware but also how the software works to control things like PWM. Basic knowledge in C++ is not required but does make things a lot easier to visualize.

# Technical requirements

The application is split up in 2 parts. One part is going to be running on the desktop and the other on the Arduino. The part that is going to run on the desktop is going to be written as a Console application in C#. It is going to be responsible for getting data from serial to UDP (FlightGear) and back. It has to be reasonably error resilient because an error in this program not going to be obvious immediately to the end user. Performance is also nice to have but has not the up most priority, it should not take a lot of CPU just for copying things over.

The Arduino part is going to be written in C++ as this allows for each meter to be split up in classes while having the main sketch reasonably abstract of them. This is going to be rather performance sensitive as there will be lots things happening at the same time. The goal is to keep everything responsive even if there are other long running things also running.

C# is going to be written in Visual Studio using .NET Core 3.0 (this allows to also be deployed to a Linux machine if required in the future). The Arduino part is going to be written in Atmel Studio.

# FlightGear data IO

To get the FlighGear simulator to work with retrieving and sending data from/to the Arduino a middleware program has to be written. As the simulator does not support sending and receiving data from serial directly. It also does not support the direct communication even if it’s one way as it will hang when in the state “finalizing position”. The solution to this is to make a program (in C# for example) that opens the serial port and forwards everything to the simulator using UDP on port x. And because UDP is not bi bidirectional you need to use another port (port y) to receive on. So, the program that is in the middle just performs a binary copy as both the Arduino serial and the Generic Protocol of FlightGear as ASCII based. The result is that all the data that is send from FlightGear is binary copies to the serial, and all the data is read from the serial is binary copies to another connection to FlightGear.

Using this method, the FlightGear simulator never has to use the serial connection, but still allows a Arduino to be used.

## The Forwarder

The program takes a couple of arguments. The ones that are necessary are,

* udp-in-port This defines the UDP port to listen to for messages from FlightGear
* udp-out-ip This defines the IP address to send the message to for FlightGear
* udp-out-port This defines the port that FlightGear is listening on for incoming data
* com This gives the name of the com port to connect with on the machine
* baud This defines the baud rate to use on the serial port
* debug This tells the program to be more verbose about what it is doing
* copypaste Use the basic forwarder (defines below)

The forwarder has 2 modes it can operate in. It can operate in the mode that is just copy pastes everything that is send on the connections between the Arduino and FlightGear. But it also has the mode that I can detect changes in the values and only send those changes. In that mode it parses the incoming message and splits it based on the \n character. Then it splits the line on comma’s and keeps those values in a dictionary. If one of those values has changed, it will set a boolean that it needs to forward the new data. Both the incoming and outgoing connections have their own dictionary for this. Using this method, the data pressure on the Arduino for receiving data is much lower does not have to keep sending and retrieving the values even if they have not changed. The second mode is the standard that it will use.

## FlightGear XML

The input/output to FlightGear can be customized using what FlightGear calls the “Generic protocol”. When you start the simulator, you need to tell it to use this file. This is done by using the –generic parameter and then other parameters are given like a string assignment using a, to separate values.

--generic=socket,in,100,127.0.0.1,49999,udp,FlightGearToArduino

* Socket tells FG to use sockets as connection (as opposed to “file” for example
* In tell FG that that socket is used for the input (other options are “out” and “bi”)
* 100 is the maximum refresh rate (in Hz) for that connection (the max is the framerate)
* 127.0.0.1 is the IP for the socket to use
* 49999 is the port for the socket to use
* Udp tells what kind of socket to use
* FlightGearToArduino is the name of the input/output definition file that is located in FG\_root/Protocol/(filename) and by default it check for files with the extension .xml

The file itself is just plain XML. The root element is name “PropertyList” and the only child element is called “generic” (probably as in Generic protocol, I have not found another xml node name that can also be put in “PropertyList”).

This Generic node then contains 2 other nodes, “input” and “output”. They define exactly the thing you’d expect they do. The “input” node defines the data the is received by FG and how it should be parsed. The “output” node defines the data that is send by FG and how it should be printed.

They both have a few parts in common, they both have the nodes

* “line\_separator”, this defines the last string appended to one frame of data
* “var\_separator”, this defines the string that is placed between 2 data parts
* “binary\_mode”, use the protocol in byte mode not ASCII (the data is not string encoded but given in raw binary data
* “preamble”, only used on output, a header to put on each frame of data
* “postamble”, only used on output, a footer to put on each frame of data (much like “line\_separator”)

We are only going to use the string output, the following will assume that you are using ASCII mode, but could also be compatible with binary mode.

“var\_separator” and “line\_separator” can contain any string you want, but also have a few special keywords. The \n (0x0A, ASCII) is forexample aliased by the “newline” word.

|  |  |  |
| --- | --- | --- |
| Character | keyword | Special character |
| Newline feed | newline | \n |
| Tab | tab | \t |
| Form feed | formfeed | \f |
| Carriage return (CR) | Carriagereturn | \r |
| Vertical tab | Verticaltab | \v |

After those nodes, each input and output block as “chunk” nodes, they will be read/written in the order that you written them down. Each chunk can have a “name” node, this is just purely for the one that is reading, it will not be read from/written to the connection. Each chunk then has a xml node called “node”. This is the property tree that is used for that block (ie “/controls/gear/parking-break”).   
Then it needs a node called “type” this is not mandatory, but when trying to output float’s when the type is not set to float, it behaves very strangely (it does allow it self to be formatted as a integer, but not as a floating point for example). So, this makes it mandatory for anything other than integers as that is its default. There then is the node with the name “format”, this is the format parameter and it behaves the printf from regular C/C++ (I.E. %2.3F prints something like this 12.345).  
There are some other nodes (offset and factor) but they are rarely used (only on converting from like Celsius to Fahrenheit.

## Speed meter

FlightGear location: /instrumentation/airspeed-indicator/indicated-speed-kt (double)

The speed meter is controlled using a PWM output from the Arduino. This means that by using a simple map from 240 knots (on the meter) to 0, and have this map to a value from 0 to 128 or 174 at 3.3V instead of 5V (NOT 255) for the PWM output to run.

## Artificial horizon

FlightGear location: /instrumentation/attitude-indicator/  
 indicated-roll-deg (double, -360:360)  
 indicated-pitch-deg (double, -360:360)

This meter has 2 motors as it has 2 dimensions. They are both 24V DC motor and they have a pot meter connected to them. The value of the pot meter is almost 0 – 1024 for the max range of the rotation and pitch value (I.E. when the rotation is rolled most to the left, the meter is more around 995, but fully to right is 0). The roll meter also does not max at the (-)60 degrees like a normal artificial horizon usually does, it is more at (-)50-55 degrees. And the pitch can actually go a bit outside the scale shown on the paper.

The motor controller is actually pretty simple, it is not an H-Bridge. You need to set either of the pin HIGH and the other to low. This way you set the motor forward and backward. The motor speed it actually like 3 RPM as stated on the motor itself. I was not able to test this fully as I ran the motors mostly at 12V or 20V as the max voltage of the supply was 20V 0.8 A. I tested this at 12V, this way the altitude meter can also feed on the 12V rail and save more cabling mess up (and the change of happening) and it worked perfectly fine. Even in real situations 12V is enough for the motors to run at a decent speed for their purpose.

## Heading

FlightGear location: /instrumentation/heading-indicator/indicated-heading-deg (double, 0:359.99)

This meter was later decided not to implement as there was not enough time and also not important enough for basic flight.

## Altitude meter

FlightGear location: /instrumentation/altimeter/indicated-altitude-ft (double)

The altitude meter has a stepper motor attached with it that is controlled by the EasyDriver from <http://www.schmalzhaus.com/EasyDriver/index.html>. It has a very basic control but does require some set up to have it make single steps correctly. By default, the controller makes 1 full step why you go from LOW to HIGH on the STEP pin. However, this does not seem to work always and sometimes misses steps. The solution that I found was to put the controller is ¼ step. This means that instead of making 1 full step which was 1.68 degrees (double check this) on the motor axis, the motor makes only a fourth of that step when you pull from LOW to HIGH on the STEP pin. This does also mean that the accuracy also increases of the meter. You can set the mode by setting the MS1 pin to LOW and set the MS2 to HIGH.

Something else that is rather useful to know is that you should try and make it that the full duty cycle of the step is double the time that the STEP was high. So, the full cycle from LOW to HIGH to LOW is twice the length of going from LOW to HIGH (or HIGH to LOW for that matter). Also, you should have all the ground of the control voltage connected to each other (the ground of the chip should be connected to the ground of the controller that uses the STEP and DIR pin). If you do not have that, the step motor will go erratic and start moving back and forward.

## 

## Climbing meter

FlightGear location: /instrumentation/vertical-speed-indicator/indicated-speed-(fpm|kts|mps) (double)

This meter has 2 cable connecting to PWM ports on the Arduino. The thing almost acts as a H-Bridge, just some minor differences. The 2 pins each represent one of the sides, one is the + scale, the other is the – scale. When you put 0 on both, the needle will stay in the middle. When you put 65 on either of the cable it will then max out to the scale it belongs to (that be either the + or the -1). Watch out though, it never gets to 2, it maxes out just after 1.5. For the top one when you look at it from the back also represents the top scale on the front (bottom back = bottom front). For the top one to be precisely on 1.5 you need to set the PWM regulation on 62 out of a width of 255(on 3.3V it is 102). To put the bottom one at 1.5 you need to put 63 (97 on 3.3V) on that same scale (the default scale on Arduino is 0 – 255). If you would put 2 different voltages on the 2 lines, they will cancel each other out (it essentially takes the difference between the 3 voltages).

## ADF

FlightGear location: /instrumentation/adf/indicated-bearing-deg (double0:359.99)

This meter was later decided not to implement as there was not enough time and also not important enough for basic flight.

## RPM meter

FlightGear location: /engines/engine/rpm (double 0:----)

This meter has only 1 cable connecting to an Arduino, this is also a PWM meter. To control the meter using this pin you need to map the scale of 0 to 3000 RPM to the 0 to 124 (195 at 3.3V) range of the PWM scale on the Arduino (NOT 255 again). This value can then be written to the PWM output and this is then shown on the meter.

## Throttle

FlightGear location: /controls/engine/current-engine/throttle (double)

This input is rather super simple, it just has a pot meter that you read out like you would normally. Just divide the analogRead result by 1024 and you have the value of the throttle normalized between 0 and 1, just like FlightGear wants.

## Brake

FlightGear location: /controls/gear/brake-parking (boolean, 0:1)

/brake-left & brake-right (double, 0:1)

The break lever has a resistance of 0 ohm when pushed in fully, and 15K ohm when pulled out.

This ‘button’ is also connected using a pullup to the control voltage. When the then become true, the lever is pushed in (the state ‘on’). And then when pulled out it is the opposite as expected, LOW.

## Flaps

FlightGear location: /controls/flight/flaps (double, 0:1)

This is more of a combination of 2 types of switches than just one. The way that it is currently used is the following. The pot meter is just like the throttle, do a read, divide by 1024 and you are done with that. The other thing that I did was the usage of the flaps lever, when you set the switch to up, the value becomes 0 (all the way up, not visible), the value of the pot meter is only read if the switch is set to allow the flaps to come down. This gives the option to very quickly retract and open the flaps without having to remember where to put the flaps again each time.

## Steer

This part of the program is already taken over by an Arduino that had software running to read the steer and give this input to Windows as a HID device called “Arduino Joystick”. The location however would be under /controls/flight/aileron (double, -1:1) and /controls/flight/elevator(double, -1:1).

The middle point for the elevator pot meter is 1.94 volt, then the joystick value is 0.00 (give or take another few decimals). The roll pot meter is connected to pin A2, the pitch is connected to pin A0.

## Pedals

FlightGear location: /controls/flight/rudder (double, -1:1)

The pedals are just potmeters, they do have a bit of a weirder scale, their analogRead value goes from around 370 to 605 give to take a few. This does make for some more strange mapping. The way that it is currently solved is by using the map function formula and make my own version of it. I took it from Arduino themselfs (it is on their website) and passed the x as float. This way the Arduino will use floating point math for this, and you can just directly use it to map those values to -1 and 1.

## Carb heat & mixture lever

These levers have a resistance of 0 ohm when pushed in, and 15K ohm when pulled out.

These levers act the same as the brakes lever. The mixture lever is working just normal, only the carborator’s result value must be inverted as the switch is flipped 180 degrees.

The location for the carb heat setting in FG is: /controls/engines/current-engine/carb-heat and the location for the mixture setting (which is actually a float but normalized between 0 and 1, so sending a boolean as 0 & 1 just works fine): /controls/engines/current-engine/mixture.

## Reset button

This push button has a resistance of 0 ohm when pushed in, and 15K ohm when not pressed.

## Switches (not the levers)

The simple 2 state switches that are not with a lever all need to have a pull up as they do not have one build in. The one that is currently separately wired is the landing lights. It switches the landing light on the wing on and off.

The 3-state switch has the following layout when looking at it from the cockpit. If the switch to lever all the way the pin on the bottom and the pin in the middle are connected. If you switch the lever one step down so, it is in the middle state, then no pin is connected to each other. If you switch the lever all the way down, the middle and up pin are connected. This switch has currently not connections.

# About this document

## Abbreviations

|  |  |
| --- | --- |
| Abbreviation | Description |
|  | Plaats hier in dit document gebruikte afkortingen en de betekenis |
| FG | Short for FlightGear |
|  |  |
|  |  |

## References

|  |  |
| --- | --- |
| References | Description |
|  | Voorbeelden: ‘bovenliggende’ documenten (ProjectPlan, Functioneel Ontwerp, etc.) |
| ProjectPlan | Contains general description + planning |

## Definition

|  |  |
| --- | --- |
| Definition | Omschrijving |
| Showstopper | Technical part of the application that can jeopardize the overall feasibility of the application. It is important that the showstoppers are first identified and tested. |
|  |  |

## Used materials

|  |  |
| --- | --- |
| Component | Description |
|  | Voorbeelden: hardware (elektronica, arduino, PC-type), software (IDE, DLL’s) |
| EasyDriver | A hardware helper for controlling stepper motors |
| Arduino Due | An ARM based variant of the arduion Mega, just runs a lot faster |
| Arduino Uno | Small/standard version of the Arduino |
| 12V PSU | For powering the altitude meter + artificial horizion |
| Desktop PC | This was running C# and FlightGear |
|  |  |
|  |  |

# Appendix

[APPENDIX B Agreement 6](#_Toc499208804)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Agreement point** | **Achieved** | | **Note** |
|  |  | **yes** | **no** |  |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |
|  | <Geef het acceptatiepunt op.> |  |  | <Geef opmerkingen.> |

**Other notes:**

|  |
| --- |
| <Geef eventueel opmerkingen.> |
|  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Seen on behalf of: |  |  | Paraph seen: | Date seen: |
| Project manager | : | <Geef de eigen bedrijfsnaam op.> |  | <Geef de datum op.> |
| Client | : | <Geef de bedrijfsnaam van de opdrachtgever op.> |  | <Geef de datum op.> |

APPENDIX B Agreement

If you agree with the content of this technical design, we request that you return this project document signed.