George Johnson – 18024970

Systems, Avionics and Control

**Individual Report**

**Abstract**

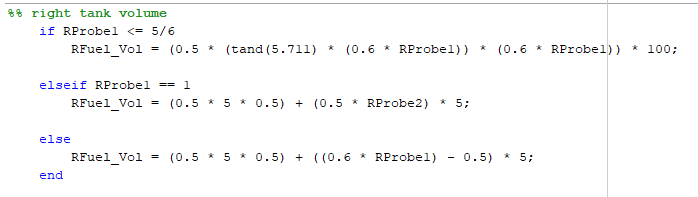
The project aim was to take an existing MATLAB simulation of a duel engine fuel system and implement working avionics including failure control and control automation so a pilot doesn’t have to control every system by hand.

**Introduction**

Our first group meeting to discuss the initial problems we had to overcome to start the project. I made a GitHub repository to keep the group project files up to date so no one would be working on old saves. It also allowed the creation of issues meaning we could help each other and keep track of previous issues we have had. After creating the repository, the first problems highlighted in the meeting were designated to each group member. The problem assigned to me was understanding the fuel tank probes and volumes.

**Individual Contribution**

To understand how the fuel tanks worked I first had to figure out how the fuel would interact with the geometry of the fuel tanks as the were triangles with a rectangle on top. I wrote the following equation to account

The first equation is to calculate the volume of fuel as the triangular section of the tank fills and the following equations calculate the volume of the rectangular section of the fuel tank. These volumes were then multiplied by the density of the fuel (797kg/m^3) to calculate the mass of the fuel. This output variable made the rest of the code more streamlined and easier to understand for the other group members. The second task I was assigned was to create failure checks for each sensor. The fuel pump failure checks were fairly straight forward as once I had found a solution for one I could copy and paste that with different variable names. I had one major problem with the system storing that fact that the pump had failed and that it shouldn’t be turned on again. I cover the solution to this problem in the next heading “Issues”. After solving the pump failure checks I then moved on to the probe failure check. This was not as straight forward as the probes did not have a feedback like the pumps did to check that status. The first solution I came up with was to compare the probe read out to an assumed probe readout based on the engine fuel consumption and the transfer input/output. However, this solution proved bulky and a much cleaner solution was to compare the probe read outs in on tank to their equivalent in the other tank. This assumed that both respective probes don’t fail at the same time.

**Issues**

The fuel volume calculation came with many issues as it was the first problem I had to solve so I was unfamiliar with the fuel system layout. The first issue was thinking my equation was wrong as I did not realise MATLAB defaulted to radians for trigonometric calculation. This resulted in a lot of deleted functions and would have been solved by simply reading the documentation for MATLAB in greater detail. The use of persistent variables was extremely helpful in keeping error states of failed pumps. One major problem I had was the function “forgetting” that a pump had failed and it would be turned back on again causing the pump to be turned on and off every time the function ran through. This was solved with persistent variables. This fixed the problem however causes another smaller problem were the failed pump could not be turned on again even if the pump failure had resolved itself. I could not find a fix to this using persistent variables so I would look into this further or find another solution.

**Test Procedures**

Once the GUI had been complete test procedures became easier as I wouldn’t be using constant blocks as inputs per run and could change the variables in a single run. I ended up using a sub system to induce a failure in the read out of the fuel pumps. The subsystem included a switch block toggled by a switch on the GUI. Each section of code that checked for a failure would also output a line of text to the console. This would inform me if that section of code was running when it should or shouldn’t. This helped refine the conditions under which the code would run to ensure the fuel management system kept fuel pressure to the engines unless there was a major fault in which this couldn’t happen.

**Integration of Work with Group**

As stated in the introduction, a GitHub repository was one of the first things set up to prioritise work flow between group member and limit time spent on integrating other members code into a main document. MATLABS incorporation of git made it easy to track modified files and update content for other users. When I initially started a problem, I would create a separate function to test out the solution to the problem to limit errors caused by the main “FuelContol.m” function, as it has many variables and inputs which could possible conflict. This ensured that the code I wrote worked and that any errors caused by the integration were caused by another systems variables or outputs.

**Improvements and Recommendations**

Instead of running all the variables through a single mux for a single MATLAB interpreter it would be easier to make smaller functions for each system as this would reduce the amount of bloated comments and code sections in one MATLAB file. As all the failure checks for each pump and probe really extended the length of the file making it hard to find what I was looking for with out using the find function. It would have been better to use MATLABS design app for the GUI instead of having it in the Simulink file as when the simulation was running moving the screen around the GUI was very slow.