**Name:** Carmen Fang

**NetID:** jf442Carmen Fang (jf442)

**Major:** Mechanical Engineering

**Year:** Class of 2017

**Start and End Dates:** Jun 17th to Aug 13th, 2016

**Objective:** Developing preliminary thermal control subsystem on the spacecraft-design MATLAB tool

**Hours Worked:** ~120 hours in total

Weekly meeting and individual meetings: ~18 hours

Reading materials and researching component properties: ~40 hours

Coding, debugging and testing on existing CubeSat: ~60 hours

Writing reports: ~5 hours

**Overview:**

Throughout the course of 8 weeks I ironed out the rough thermal balance on the satellite as a whole as well as on individual component level (i.e. to run temperature calculation on payload, antenna, batteries and other parts loaded). Running through my code package will return the overall radiator configuration and the power for heater, cooler or heat pipe if such active thermal components are needed to maintain the desirable temperature range on the satellite as a whole or on individual component basis. During the first stage of my research I also studied fair amount of thermal configuration design. Due to the limitation on thermal design details disclosed on existing satellites and the realization that thermal design and configuration is largely individualized and repeatable configurations are less than common, I decided to put this part of the coding off and hence didn’t make much influence on structure subsystem in terms of its execution logics.

In my opinion, whoever continues the thermal subsystem after me should proceed to the design on each individual component and to masterplan the thermal configuration on the satellite. Develop temperature gradient on the entire satellite should be priority on larger scale satellites but may not be as critical for CubeSat. General design principles should be easily found and understood. Materials I found helpful include: SMAD, Spacecraft Thermal Control Handbook by Gilmore, Spacecraft Thermal Control by Meseguer. Search related topics on google including the search word “NASA” usually returns fruitful results as well.

Brief explanations and math equations for each MATLAB file in my folder are included in the following pages.

**CreatComps.m**

This code was created on the sole purpose of testing and debugging other thermal codes I wrote. The format and field name follows the same rules as the list of components that is returned from running structure subsystem code and hence any further modification on this code should keep the format and only change the properties for smoother future merge with the main code.

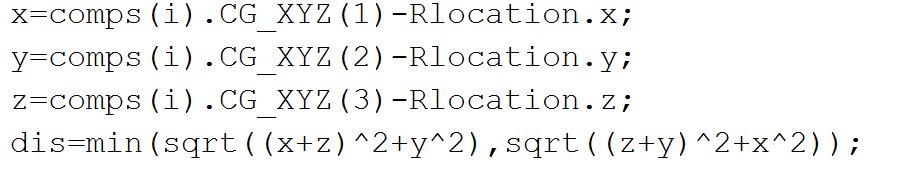
The properties written in this code are in no ways truthful to the reality beside the *component(i).Thermal* field , which is the temperature range I found from the reading materials I listed in the overview section.

**HeatPipeComponent.m**

This code runs after *InsideComponent.m* and *OutsideComponent.m.* After those two scripts decide whether each component need heat pipes to take extra heat in order to operate within the operational temperature range, this code will calculate the length needed assuming each component only gets one single heat pipe, and the diameter of the heat pipe.

The diameter of the heat pipe is determined based on the power needed on specific heat pipe and criteria were set with the aid from the information available on: <http://www.thermacore.com/>.

The length of each individual heat pipe is calculated as the shortest length between the gravitation center of the component(component.CG\_XYZ) and the location of the radiator on the satellite(Rlocation) with the track on the walls of the satellite. The method used to find the distance makes sure the track goes through either the floor or the ceiling then to one or more side walls.



Component’s location

Radiator’s location

After the math is done, this script will return the total volume of the heat pipe modified as a thick plate (for the purpose of easier display on satellite structure) as well as the total mass and cost of the heat pipes.

**HeatPipeProperties.m**

This code contains the properties of the heat pipe that’s used to get the mass and cost of heat pipes in *HeatPipeComponent.m.* The unit price of the heat pipe is completely made up since I was able to get in touch with sales on any of the commercial heat pipe companies online. The unit mass (or density) is the number I calculated with Aluminum tubing, Ammonia working fluid and 5mm diameter in mind. As this is the most common combination of a heat pipe and works great for most of the CubeSat. Further elaboration on heat pipe types and configuration designs is definitely needed in the future.

**incoating.m**

This code adds one layer of black paint on every single object that will stay inside the satellite during operation. The assumption is not accurate as not all surfaces of the inside components can be covered (i.e. computer screen) but the result should not vary too much from the reality since the paint is light and the components are not huge. It returns the mass of the paint and the cost of the black paint assuming Chemglaze Z306 is used on the components. This paint is hardcoded in the current thermal calculation as it’s economical and offers reliable performance. Further material selection might be needed in future coding of thermal subsystem.

**inpaintProperties.m & outpaintProperties.m**

I combined these two scripts because the reason behind them are the same. I handpicked Chemglaze Z306 as the inside black paint and Z93 as the outside white paint. For the purpose of calculating mass, cost of the coating and the thermal balance on individual components, I cross referenced the properties of each paint from websites that sell those paints as well as Spacecraft Thermal Control. The links of the websites can be found in each script. For easy reference: (white)http://www.aztechnology.com/materials-coatings-az-93.html (black)<http://www.spacematdb.com/spacemat/manudatasheets/Aeroglaze%20z306.pdf>

**InsideComponent.m**

This code does individual thermal analysis on component level with all components inside the satellite. It has a hard coded 0.3 factor on radiation hoping to account for the influence of view factors and backload from other components inside. This can be modified in the future as seen fit.

I divided the components with their operational temperature range in three groups

278K 308K Temperature

Group II

Satellite target temperature range

Group I

Group III

Group I contains the components that have its own temperature low limit lower than the satellite level low limit and its own temperature high limit higher than the satellite level high limit (i.e. the component can operate fine in the range of [-40,100] while the satellite will always be maintained in range [5,35]). In this case, the component will not be wrapped with MLI but instead just have a layer of black paint to reduce radiation disturbance inside the capsule.

For all components fell into this criterion, the code will run a radiation balance between the component at its high limit and the satellite at its high limit to determine if the component need help from heat pipes to guide some heat it produces out in order to prevent the component temperature from going beyond its high limit.

Specifically, if Qw is the heat power the component produces, Tsh is the high limit of the satellite and Th is the high limit of the satellite itself. T is the unknown value the code will solve with the following equation to determine the necessity of heat pipes.

If the T solved is higher than Th, which means when the radiation balance is established the component will have a higher temperature than what its allowed (or to say with no extra help, component at its own fails to expel all the heat it produces). In this case, heat pipes will be needed and the power the heat pipe requires is the difference between Qw and

Group II contains components with temperature high limit lower than the satellite’s high limit (i.e. if the component can operate at [-40,20] while the capsule will be maintained at the range of [5,35]). In this case the component will be coupled with MLI and this code will run math to determine if the component need active coolers to cool its temperature down to operational range when the satellite environment has temperature too high.

MLI with unknown outside boundary temperature T

Satellite at its high limit Tsh

Component at its own high limit Th

Qw

Conduction

Radiation

The following conduction and radiation balance is established to find the boundary temperature T

is the thickness of the multilayer insulation. And after T is found, the amount of heat passing can be calculated by plug T back in either the conduction equation or the radiation equation. Comparison between this heat (called Qout in the script) and the heat produced by the component Qw. If Qout is larger, which means the component will likely not reach its high limit even when the satellite is at its worst case hot temperature, nothing needs to be done. On the other hand, if Qout is smaller than Qw, cooling component will be added and the max power the cooler needs to offer is the difference between Qout and Qw.

Group III contains components with temperature low limit higher than the satellite’s low limit (i.e. if the component’s operation range is [20,50] while as the satellite will be kept at [5, 35]). In this groups it can potentially happen even with MLI on the component that the component won’t be able to maintain its own low limit when the satellite is experiencing the worst case cold temperature.

Radiation

MLI with unknown outside boundary temperature T

Component at its own low limit Tl

Qw

Conduction

Satellite at its low limit Tsl

Similar thermal balance is established to find the boundary temperature, only in the reverse direction

With T found, if the Qout is smaller than Qw, which means at worst case cold satellite condition, this component won’t quite reach its own low temperature limit. Whereas if Qout is greater, heaters offering power which is the difference between Qout and Qw is needed to maintain the component’s low temperature.

After the script runs through the list of component and adds corresponding heater, cool or heat pipe power to the components, it will add up the total power consumption on the active components (heaters and coolers) and return the values as outputs for thermal’s power estimation.

**MLIproperties.m**

The reason behind this script is similar to *inpaintProperties.m*.I essentially hand calculated the density and thickness of the MLI by adding properties listed I found on Multilayer Insulation Material Guidelines published by NASA in year 1998 assuming the MLI is 25 layers of inner insulation with optimum performance. Cost of the MLI is a complete made up number I set to make sure the entire codes run smoothly.

**outcoating.m**

This code is essentially the same as *incoating.m* besides the fact that while adding white paint to all outside components, it also includes both surfaces of the main radiators of the satellite.

**OutsideComponent.m**

This code operates on similar logic as *InsideComponent.m* only that all the components outside has MLI by default. The performance of this code compared with the reality is in question because solar input should also be included in the thermal balance whereas for the current ability of the entire program, detailed solar information is not available on individual component level. The space environment is assumed at absolute zero temperature, so there’s no consideration on active cooling components.

**preMLI.m**

This code adds Multilayer insulation to all the components operating outside the satellite as well as the components inside that have operational temperature range that suggest they might need extra thermal components to make sure they operate in okay temperature environment. Specifically, if the component’s lower operation limit is higher than the satellite lower limit, that means there might be occasions when the satellite environment is too cold for the component to operate. In this case I choose to add MLI to said component. Similarly, if the component’s high limit is lower than the satellite’s high limit, meaning satellite environment can sometimes be too hot for the component, I add MLI to it as well.

After MLI is done wrapping, I update the dimension of components with MLI attached, adding the thickness of the MLI on top of the dimension information of the component. The code also returns the mass and cost of MLI utilizing the properties in *MLIproperties.m.*

**ThermalEstimate.m**

This code runs thermal estimation on the satellite level with hard coded desired temperature range as [5,35] in Celsius\*, relevant orbit information as input and radiator dimension as output. It establishes thermal balance with solar radiation, earth reflected solar radiation, earth infrared energy and the satellite’s power consumption. 2 situations are considered: worst case hot and worst case cold and louver angle, active component necessity are also estimated as when the case applies.

The operating sequence is as followed:

Find the worst case hot temperature when all solar related inputs are at their maximum and satellite power consumption at the lowest. 🡪 Compare worst case hot temperature with the hardcoded temperature upper limit to determine radiator mounting method. 🡪Replace the worst case hot temperature with hardcoded upper limit to find the area required of the radiator. 🡪 With modified solar related input at the lowest, satellite power consumption at the highest and the radiator area found in the last step, find the worst case cold temperature. 🡪 Compare worst case cold temperature with the hardcoded temperature lower limit to determine if louver is needed and determine the louver angle at this situation. (there’s a possible case where the worst case cold temperature is higher than the default value, in that case the lower limit should be upgraded to the worst case cold temperature but I chose not to do it because of the level of precision current estimation has.)

Several points needing further elaboration are as followed:

1. When the worst case hot is considered, due to the limitation of the orbit information available when I was writing the code, the case is simplified as the image illustrated: both the solar input and the earth input are shined on the satellite perpendicular to the side surface and the satellite will radiate energy with 2 side surfaces and 1 top surface as the other 2 side surfaces are used by solar panel and the bottom surface usually have payload or other components that observe the earth mounted.

Earth energy input

Solar energy input

1. Most if all the constant properties I used is extracted from SMAD differentiated with worst case hot and worst case cold (maximum or minimum value). When the worst case cold case is considered, it’s assumed that there’s no solar energy input and hence no earth reflected solar energy.
2. The emissivity and absorbance values are represented by white paint after 5 years of use so the values are pretty terrible (α=0.5 and ε=0.9) compared with the reality. This makes sure that the estimation from this script will literately be the “worst possible”. But sometimes such rough estimation may not be needed or ideal. The properties can be modified as seen fit and the model can be improved when more detailed information about the orbit can be obtained.
3. The mounting method is determined based on the radiation area needed. If the area needed is greater than the area the satellite can offer (2 sides and 1 top), the radiator will be deployed and if the area needed is less, the radiator will be a thin face-mount sheet of Aluminum (or just a layer of coating if the structure of the satellite housing is a metal box). The option of using structure panel as radiator is not explored because with current information available on the satellite, it’s difficult to find structure panel let alone utilizing it.

\*I had a script determining the reasonable temperature range that accommodate certain percentage of the components on board after disregarding some components with temperature outliers (i.e. as the majority has low limit as -20 degree but one component has -400 degree). But the code got unnecessarily complicated whereas the results it returned weren’t ideal. Hence I got rid of the code completely and hard coded the range [5,35]. It is reasonable and close to what’s commonly seen on satellites (even manned satellites).

**Thermal.m**

This code is the main thermal code called in the satellite level code. The script run the estimation in the following order: get all the input values needed (as for right now a lot of them are just hardcoded based on my hypothesis from the satellite we are using to test the code)🡪 run thermal estimation on the satellite level and get radiator information output 🡪 wrap all the components with corresponding paint and appropriate ones with MLI, figure out if active components are needed and what power is needed on the active components 🡪 sum up the mass, cost, power from the codes run and output those value as thermal.\_\_\_\_ to aid the satellite level estimation. \*

\*I kept the SMAD cost and mass estimation in the main thermal code as comparison and sanity check with the values returned from running the estimation codes. I named the mass, power, cost values I got as \_\_\_cjf for comparison purpose. When seen fit, those values can replace the original estimations from SMAD when running the entire program.

\*\*Most of the result I get from my estimation are reasonably close with the reality with my estimation tending to be larger in radiator size. I expect the estimated mass higher than the current one I have because I essentially haven’t added all the active components and still have not a complete thermal subsystem structure.

**selection.m**

Since the material I mentioned at the beginning of this write up all have somewhat detailed material properties listed in the table, materials commonly seen used on satellite. I coded up this simple material selection that can be integrated in the user interface in the future to let the user input how important each factor is in making thermal design decisions and the script will return in the specific situation indicated by said user, what are the most reasonable options of material.

This is in no way a complete list nor did I exhaust the factors that a user might consider. The selection algorithm has a lot of room of improvement. I just thought this would be a good head start so leave in the folder for future reference.

The code selects the material with the highest performance score and display the name of it on the screen. After user indicates how important each factor is, the script calculate a weighted average that will be filled under field name “Performance”.

There are 2 situations to consider, one is when lowest property is considered ideal (the coating with lowest density will generally be favored over the heavier ones) and the other is when highest property is considered best fit (the material that can survive space contamination for 100 is better than the one of only 5 days).

When “low” is desired, the material’s performance on that specific property is calculated as the ratio between the lost value in the database and the value of each individual material (so the best possible result will be 1 and everything else will have performance lower than one and the heavier the material is the lower the performance score will be).

When “high” is desired, it works the other way around the and performance is based on the ratio of individual value and the maximum value out of the database.

After summing up the performance on all the criteria (i.e. cost, mass, durability, emittance and so on), the material with the highest score will just be the user’s “best” option.