WEST VIRGINIA UNIVERSITY

Response Surface Modeling – Test Particle Monte Carlo

Version 2.0.0

User Manual

Developed by:

Logan Sheridan

Dr. Guillermo Avendano Franco

Dr. Smriti Paul

Advisor:

Dr. Piyush M. Mehta

CONTENTS

1 Background **7**
1.1 Test Particle Monte Carlo (TPMC) . . . 7
1.2 Stereolithography(STL) Files . . . 7
1.2.1 STL Orientation . . . 8
1.3 Original RSM Suite . . . 9
1.3.1 Automated RSM . . . 9
1.3.2 Projected Area . . . 10
1.3.3 Model Evaluation . . . 10
1.4 West Virginia University's RSM Software Package . . . 10
1.4.1 Extension to Complex Geometries . . . 10
1.4.2 Check Water-Tightness . . . 11
1.4.3 Regression Model Creation and Evaluation . . . 11
1.4.4 Compilation/Installation of Code . . . 11

2 Setup and Navigation **11**
2.1 Setup and Installation . . . 11
2.1.1 Modules Required . . . 12
2.1.2 Compilation . . . 12
2.2 Navigation . . . 13
2.2.1 Overview . . . 14
2.2.2 Inputs . . . 14
2.2.3 Outputs . . . 14
2.2.4 RSM_TPMC . . . 15

3 Simulation Execution **16**
3.1 Define Inputs . . . 16
3.1.1 Simulation Inputs . . . 16
3.1.2 STL_Files . . . 18
3.1.3 STL_Rotation_Inputs . . . 18
3.2 Code Execution . . . 20

4 Outputs **20**
4.1 TPMC Simulation . . . 20
4.2 Regression Model . . . 21
4.3 tempfiles . . . 21
4.3.1 Latin Hypercube Sampling . . . 21
4.3.2 Mesh_Files . . . 22
4.3.3 temp_variables.txt . . . 22
4.4 Regression Model Output . . . 22
4.5 Model Evaluation Outputs . . . 22

5 Model Evaluation **22**
5.1 Model Evaluation Inputs 23
5.1.1 Model_Evaluation.json 23
5.1.2 Model_Input_Data.csv 24
5.2 Execution of Model Evaluation 25
5.3 Outputs of Model Evaluation 25

6 Example **25**
6.1 Creating an STL File 26
6.2 Example Inputs 29
6.3 Example Execution 31
6.4 Example Outputs 32
6.5 Example Model Evaluation 33

LIST OF FIGURES

1.1 Example STL File, Slice of a Cube 8

1.2 Example of proper orientation of STL File, CHAMP 9

2.1 Top Level Directory of WVU RSM 13

2.2 Inputs Directory of WVU RSM 14

2.3 Outputs Directory of WVU RSM 15

2.4 RSM_TPMC Directory of WVU RSM 16

3.1 Example of the Simulation.json file 17

3.2 Correlation between STL_Rotation_Input Files 19

3.3 Example of the parent.txt file 19

3.4 Example of the hinge_points1.txt file 19

3.5 Example of the hinge_points2.txt file 20

4.1 Visualization of Model Performance 21

5.1 Example of the Model_Evaluation_Inputs.json file 23

5.2 Code from model_evaluation_script.py, calculation of the fractional surface coverage of adsorbed oxygen with different ADS models. 24

6.1 Structure and Mesh of example Cube-Sat with Solar Panels 26

6.2 Body Component of Example Cube-Sat 27

6.3 Left Solar Panel Component of Example Cube-Sat 27

6.4 Right Solar Panel Component of Example Cube-Sat 28

6.5 Mesh for Body Component of Example Cube-Sat, GMSH 29

6.6 Simulation Input for Example 30

6.7 Example Rotation Inputs 30

6.8 Simulation Input for Example 31

6.9 Rotation Direction for Example 31

6.10 RSM Logo and Initial Output 32

6.11 Thirty Files for the Example Regression Model 32

6.12 Example Model_Evaluation.json file 33

6.13 Example CSV file 33

6.14 Example Results 34

ACKNOWLEDGMENTS

A special thanks Dr. Andrew C. Walker of Los Alamos National Laboratory. His research and creation of the TPMC code was the basis for this software. Furthermore, his assistance in the development of this new software was pinnacle. His support is very much appreciated.

A very notable thanks to Dr. Guillermo Avendo-Franco, without whom this project would not have reached its fullest potential. His teachings and support assisted in the timely completion of the software and helped it reach a reputable stature.

Thanks to Dr. Smriti Paul as well. His assistance with this software was invaluable and his patience was unyielding. His research into regression models and evaluating the test and training data of this software helped expedite the project immensely.

Thank you to Dr. Piyush M. Mehta. As an expert in this field, his expertise and guidance drove this project to success. His support, patience, and faith in this project brought this team together as well as served as the foundation for this project to grow to fruition.

Thanks to the West Virginia University super-computing systems, which are funded in part by the National Science Foundation EPSCoR Research Infrastructure Improvement Cooperative Agreement #1003907, National Science Foundation Major Research Instrumentation Program (MRI) Award #1726534, the state of West Virginia (WVEPSCoR via the Higher Education Policy Commission) and West Virginia University.

- Logan Sheridan, West Virginia University

*This project was completed in partial fulfillment of the requirements for the degree of Masters of Science in Aerospace Engineering for Logan Sheridan at West Virginia University

1 BACKGROUND

This software package is used to estimate the drag force on an object in the free molecular flow regime. The free molecular flow regime is recognized as a flow with a Knudsen number greater than ~10, typically occurring at ~150km altitude and above. This value is defined by the following equation where λ is the mean free path and L is a characteristic length of the object.

$$Kn = \frac{\lambda}{L}$$

1.1 TEST PARTICLE MONTE CARLO (TPMC)

Test Particle Monte Carlo is a monte carlo simulation technique, first proposed by D. H. Davis in 1961 [1]. This method has been implemented in several software packages before and has proven to be a computationally efficient and accurate method to simulate flow in the free molecular flow regime [3, 4].

Test particles represent real molecules of different species of gas. The particles are sequentially fired into the computational domain with a constant free-stream bulk velocity and a determined thermal velocity. The test particles interact with the surface but do not undergo intermolecular collisions making the TPMC method ideal for free molecular flows but not for transition flows. The TPMC method can model the effects of multiple reflections caused by complex concave geometries and flow shadowing. The TPMC method can be used with different gas-surface interaction models for energy and momentum exchange as well. In the case of this software, Cergani-Lampis-Lord (CLL) and Diffuse Reflection with Incomplete Accommodation (DRIA) are supported.

1.2 STEREOLITHOGRAPHY(STL) FILES

The STL file is the focal point of the simulation process for this software. Typically STL files come in two formats; ASCII and binary. For this software it is imperative that the ASCII format is used, binary is not currently supported. ASCII STL files use a very intuitive and very specific formatting. The premise behind the formatting of STL files is that an object can be made up of a series of triangular facets. These triangular facets make up a mesh of the object. In order to create the body and the mesh of the object to the users desired specifications, a computer-aided design (CAD) software and/or a meshing software is needed. An example of how these are used is shown in the Example section of this manual.

Once the object has been created, the ASCII STL file takes on a specific format as shown in the figure below. Note each facet is defined by three vertices in the x-y-z Cartesian coordinate frame and a vector normal to the facet. The keywords "outer loop", "endloop", and "endfacet" define where the facet definition begins and ends. The keywords "solid" and "endsolid" define where the file begins and ends. "solid" has the name of the object located next to it. This name is unimportant and can be changed without having any effect on the software.

```
solid Cube
      facet normal 0.0 -1.0 0.0
        outer loop
          vertex 0.0 0.0 0.0
          vertex 1.0 0.0 0.0
          vertex 0.0 0.0 1.0
        endloop
      endfacet
      facet normal 0.0 0.0 -1.0
        outer loop
          vertex 0.0 0.0 0.0
          vertex 0.0 1.0 0.0
          vertex 1.0 0.0 0.0
        endloop
      endfacet
      facet normal -1.0 0.0 0.0
        outer loop
          vertex 0.0 0.0 0.0
          vertex 0.0 0.0 1.0
          vertex 0.0 1.0 0.0
        endloop
      endfacet
      facet normal 0.577 0.577 0.577
        outer loop
          vertex 1.0 0.0 0.0
          vertex 0.0 1.0 0.0
          vertex 0.0 0.0 1.0
        endloop
      endfacet
    endsolid
```

Figure 1.1: Example STL File, Slice of a Cube

1.2.1 STL ORIENTATION

***Note: The orientation of the STL for the software is very important**. The RSM does not follow a typical aerospace reference frame. Typically in the aerospace community, vehicles travel along the x-axis with the flow going in the negative x-direction and the z-axis pointing positive downwards. For the RSM, the opposite is true. The flow is along the x-axis with the object traveling in the negative x-direction and the z-axis pointing positive upwards. An example of the correct orientation is shown below. The example shows the satellite CHAMP. CHAMP's boom is the forward of the spacecraft, traveling against the flow.

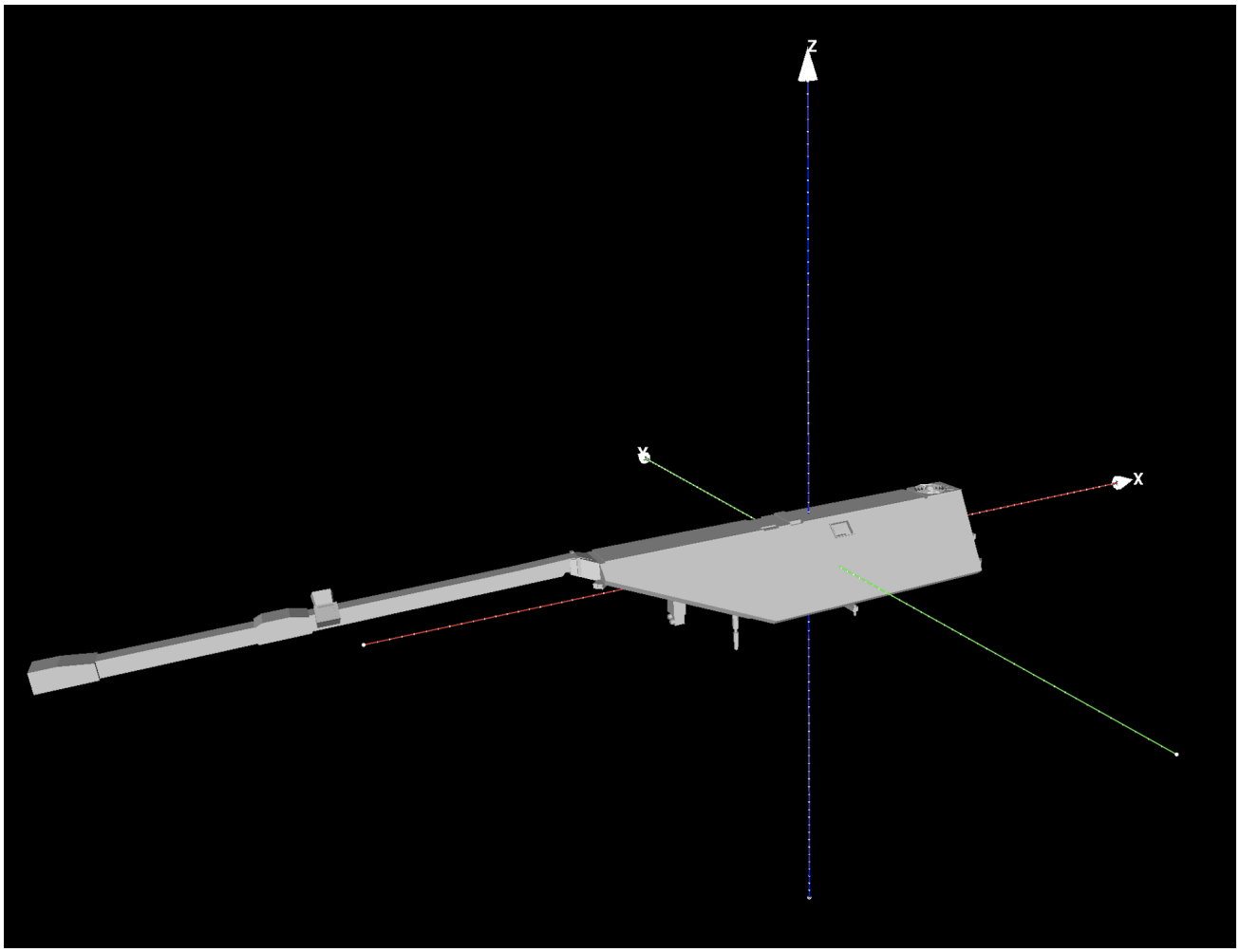


Figure 1.2: Example of proper orientation of STL File, CHAMP

1.3 ORIGINAL RSM SUITE

The original Response Surface Model (RSM) suite was developed by Mehta, et al. The code was written by Andrew Walker et. al at Los Alamos National Laboratory [2]. The suite is the basis for this software package. A brief explanation and understanding of its structure and functionality may help the user to navigate the different sections of this software package. The suite utilizes three essential codes that allowed the user to perform simulations and create a regression model, calculate projected area of the object, and perform evaluation of the model with user inputs. ***Note: this is not the structure of the WVU RSM software, this is the code the WVU RSM software package is based on.**

1.3.1 AUTOMATED RSM

This folder includes a Python execution code that allows the user to perform simulations using TPMC. The TPMC code is written in C programming language. A regression model for multiple species of gas is created after the simulations are complete. The regression model uses

a Gaussian Process Regression model created in MATLAB®, also developed at Los Alamos National Laboratory. The regression model models drag coefficients based on multiple inputs such as velocity, temperature, mole fraction, etc. Furthermore, the code is capable of running the simulations in parallel with multiple processors

1.3.2 Projected Area

The original RSM suite uses geometry of the object to calculate the projected area of a given yaw and pitch, written in C. A range of yaw and pitch is given with a certain resolution of each that ultimately creates a lookup table of areas. This table can be bi-linearly interpolated to obtain a projected area that is needed to find the ballistic coefficient of the object.

1.3.3 Model Evaluation

The final section of code is used to evaluate the model created from the Automated RSM. This code was written in C as well. The user can input the precise values for each input and receive a single drag coefficient as well as ballistic coefficient. This code evaluates the multi-variate input with a Langmuir or Freundlich adsorption model taking into account fractional surface coverage of adsorbed oxygen. There is a level of uncertainty in the model. The use of interpolation in the RSM as well as the lack of knowledge regarding the GSI leads to uncertainty.

1.4 West Virginia University's RSM Software Package

The original RSM suite was developed into a software package with extended capabilities into complex satellite geometries. The software has also been developed around providing a new user with an easily navigable environment. Furthermore, all MATLAB® code has been removed and replaced by a Python Gaussian Process Regression model via Python's Scikit-learn module. This is beneficial as it is less computationally expensive, more efficient, and open source. This new suite allows the user to not only execute and automatically develop RSM models, but also to execute individual scripts independently such as; projected area calculations, STL rotation and generation, model creation, and model evaluation.

1.4.1 Extension to Complex Geometries

The largest addition to the RSM suite is the ability to evaluate complex geometries. By complex geometries it is meant objects with rotating components such as, but not limited to, solar panels. Allowing for rotating components to be a part of the regression model creates an n-dimensional number of inputs. This proves to be a challenge with a lot of the original code. To overcome this new file types were introduced to the suite such as $JSON$ and $HDF5$ files. The readability of these, along with metadata attributes, make them ideal for information exchange within the software package.

The actual implementation of component rotation requires an understanding of how ASCII STL files are structured. The user must also have a strong understanding of the size and structure of their object as well as spatial awareness of each component. Further detail of how and why this is needed will be explained under the execution section of the manual later.

The rotation code in the new software uses an algorithm developed by James A. Tancred at the Air Force Research Laboratory [5]. The algorithm takes each individual component of

the satellite and rotates it independently. The algorithm was extended to combine all of the components to form the whole structure of the object. This process creates a new STL that is used for a single simulation. If there are multiple simulations being performed, that many STL files will be created. For instances, 1000 ensemble points will generate 1000 STL files for simulation in the TPMC code. To make sure there are no errors with this method, a "Check Water-Tightness" function was created and is performed on each STL file.

1.4.2 Check Water-Tightness

Another addition to the package is the ability to check water-tightness of the object. Also sometimes referred to as manifold surfaces, the surfaces of the object can not have any gaps or holes in them. Breaks in the structure can cause particles to become trapped and bounce around inside the object giving false data. To overcome this, a basic function was created to make sure each side of a triangular facet is shared by exactly two triangles. If a mesh is not watertight then the program well stop execution with an error message.

1.4.3 Regression Model Creation and Evaluation

The regression model creation and evaluation codes were both changed to python scripts that are more user friendly, efficient, and open source. It should be noted that the model creation was extensively tested and compared to MATLAB® models as well as data from other drag simulators.

The evaluation code was changed from C to Python for user accessibility as well as the compatibility with the Python generated model. The difference in performance between the two languages has shown to be negligible. The new evaluation script takes user inputs as before but now also includes the component orientations. Using a CSV file, the user can input the appropriate inputs as a row of data. Implementing multiple rows of input data allows the evaluation script to return drag information for multiple instances and/or time series data.

1.4.4 Compilation/Installation of Code

A large difference between the original RSM suite and this software package is the build of the package. To elaborate, the suite is now a software package in the sense that compilation and installation of the package is backed by a series of scripts that evaluate the users environment to make sure execution will be successful. This is still a high level package that is not recommended for desktop environments. Instead, it is recommended to be used with high performance clusters utilizing multiple nodes and processors. That being said, this software is still capable of running on a desktop environment. The package is also capable of giving the developers debugging information in case any issues arise. How to perform the installation will be discussed further in the next section of the manual.

2 Setup and Navigation

2.1 Setup and Installation

The software package can be obtained by the provided $GitHub$ repository link. Once downloaded, the package needs to be setup. This is a one time procedure the user must perform to

compile the appropriate TPMC code. ***Note: Whichever modules the user uses to compile the code must also be used to execute the code as well.**

2.1.1 MODULES REQUIRED

In order to compile, make sure the following module types are accessible and loaded on your desktop, cluster, and/or node. Also, make sure your Python modules are backed with $numpy$, $pandas$, $h5py$, $scipy$, and $sklearn$. All other modules used are a part of the Python standard libraries. Another requirement is that GSL, used for random number generation, and MPI, used for parallelism, be included with your C compilers. If any of these are not present, they can be downloaded from the internet or speak to a cluster specialist about including them in your Python packages and C compilers. The main compilers needed are:

1. C code compiler, with MPI implementation
2. Python 3.x interperter
3. HDF5 library for C

Examples of the appropriate packages can be found under the $load\_sources.sh$ script found in the RSM_TPMC folder. This script can also be modified and used for the user's own purposes. Currently, as downloaded, the script can be called with the following command and output the following modules:

```
$ source load_sources.sh intel19
$ module list
lang/gcc/9.3.0 lang/python/intelpython3_2019.5
lang/intel/2019 libs/hdf5/1.12.0_intel19
```

***Note: These are the modules and directories used on the developers cluster. Load in the modules via your own cluster instructions/directories. This is for example purposes only.**

2.1.2 COMPILATION

There is only one code that needs to be compiled before use and that is the TPMC code. This code is the file called $tpm.c$. This file is found under the $RSM\_TPMC/tpm/src$ working directory. In order to compile follow the following instructions. Any errors that may arise during compilation will be noted by the autotools build system.

1. Navigate so your current working directory is $./RSM\_TPMC/tpm/$.
2. Create a folder for compilation, this is not mandatory, but it is a good practice to separate the compiled code from the sources.

   ```
   $ mkdir build
   $ cd build
   ```

3. Execute the configure with appropriate flags. $mpiicc$ is the compiler wrapper over Intel compilers and $mpicc$ is the Intel MPI wrapper over GCC compilers. The code needs HDF5 and GSL as well, for example, using Intel compilers:

   ```
   $ CC=mpiicc ../configure
   ```

The configure will figure out the right values for HDF5 and GSL libraries, otherwise you can also explicitly insert them like this:

```
$ CC=mpiicc LIBS="-lhdf5 -lm -lgsl -lgslcblas" ../configure
```

4. Execute make to build the code

   ```
   $ make
   ```

5. After compilation the executable will be located at: $build/src$ with the name "tpm"

   You can use this binary directly or you can install it. For installing the code you can decide a good place for installation with:

   ```
   $ CC=mpiicc LIBS="-lhdf5 -lm -lgsl -lgslcblas" ../configure
   --prefix=$HOME/.local
   ```

   and after make, install the code with:

   ```
   $ make install
   ```

   Otherwise, this executable code can be used to run the package. For the rest of the manual the binary file will be used directly and can be found in the directory: $RSM\_TPMC/tpm/build/src/tpm$.

All of the Python scripts should be executable upon downloading from the repository. If for some reason the scripts are not, *$ chmod +x* needs to be performed on the following scripts to make them executable. If all are executable, the user is ready to begin utilizing the software.

- model_evaluation_script.py
- projected_area.py
- regression_model.py
- rotate_stl.py
- rsm_run_script.py

2.2 NAVIGATION

This section will illustrate to the user how to navigate the software package. The following diagram illustrates the top level directory of the software package. Blue colored blocks represent folders, green represent a script or code.

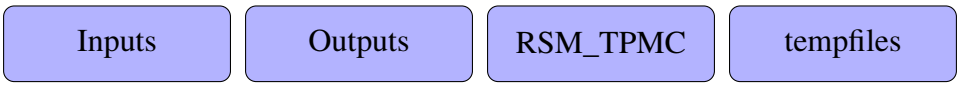


Figure 2.1: Top Level Directory of WVU RSM

2.2.1 Overview

In the top level directory there are four folders; $Inputs$, $Outputs$, RSM_TPMC, and $tempfiles$. The Inputs folder is where the user will change values as they see fit. This is the only place where the user has to change anything in the software. The Outputs folder is where everything of importance is output by the various codes. *RSM_TPMC* is where all the code is contained. Finally, *tempfiles* is a folder where several temporary files are kept. The user never needs to change anything within here. For informational purposes though, the items contained within are variables that are passed from Python scripts to the tpm code, the STL files created by the rotation Python script, and the ensemble information created by the LHS. The STL files placed within are not deleted automatically.

2.2.2 Inputs

How to use the $Inputs$ folder will be further discussed under the execution section of this document. This is an illustration and explanation of the structure of the $Inputs$ folder. The Inputs folder has the following items within:

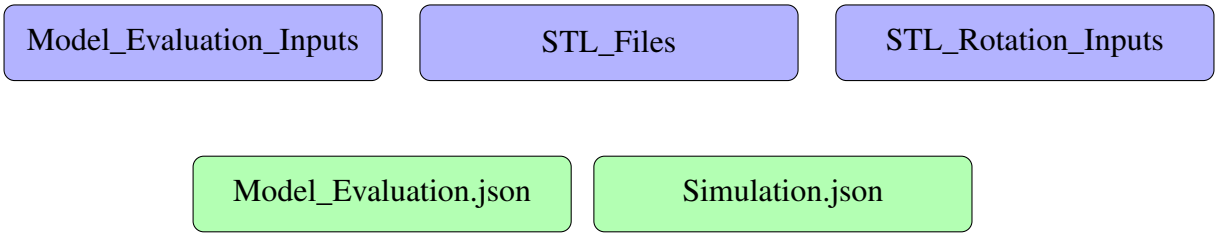


Figure 2.2: Inputs Directory of WVU RSM

The $Model\_Evaluation\_Inputs$ folder contains two items necessary to produce the drag coefficient results. First is the $Model\_Input\_Data.csv$ file used to calculate the drag coefficient. The second is the $Regression\_Models$ folder. This folder is where the user places the regression models generated from the simulation scripts.

The STL_Files folder is where all STL file types go. Whether individual components or a single structure object, this is where the user will place their STLs. Note, this is not where the rotation script outputs STLs. Since that script outputs one STL per simulation, they are stored and deleted in the $tempfiles$ folder.

$STL\_Rotation\_Inputs$ consists of three files that are necessary to define how rotations will be performed. How to use these files will be explained under the execution section of this document.

The $Model\_Evaluation.json$ and $Simulation.json$ files define the different values and ranges used to create and evaluate the model. Their structure will be explained later in this document.

2.2.3 Outputs

The $Outputs$ folder is where all pertinent information from the scripts is output.

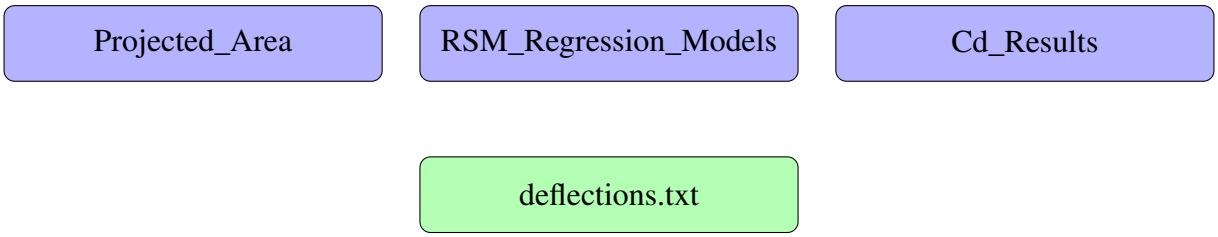


Figure 2.3: Outputs Directory of WVU RSM

$Projected\_Area$ contains the file produced by the tpm code with the projected area of each simulation. This file is called $Aout.dat$. The user does not need to do anything with this file, but the $projected\_area.py$ code does use it to create a model for projected area calculation. Also, the final projected area results from $model\_evaluation\_script.py$ are output to this folder as a text file.

$RSM\_Regression\_Models$ is where the regression models from *regression_models.py* are output. After each run, there will be thirty files added to this folder.

$Cd\_Results$ is where the drag coefficients from $model\_evaluation\_script.py$ are output as a text file.

The deflection angle is the angle of rotation of a single component of the object. The $deflections.txt$ file is created from the Latin Hypercube Sampling (LHS). During the LHS sampling the deflection angles for each ensemble point are created and saved to this text file. This is for reference purposes only. The actual $deflection.txt$ is not used by any other scripts.

2.2.4 RSM_TPMC

This is the section of the software package that contains all of the code. There are many parts of it that are not useful to the user. Only the relevant directories will be discussed. A diagram of the folder is shown below. There are multiple sub-directories that contain the code being used.

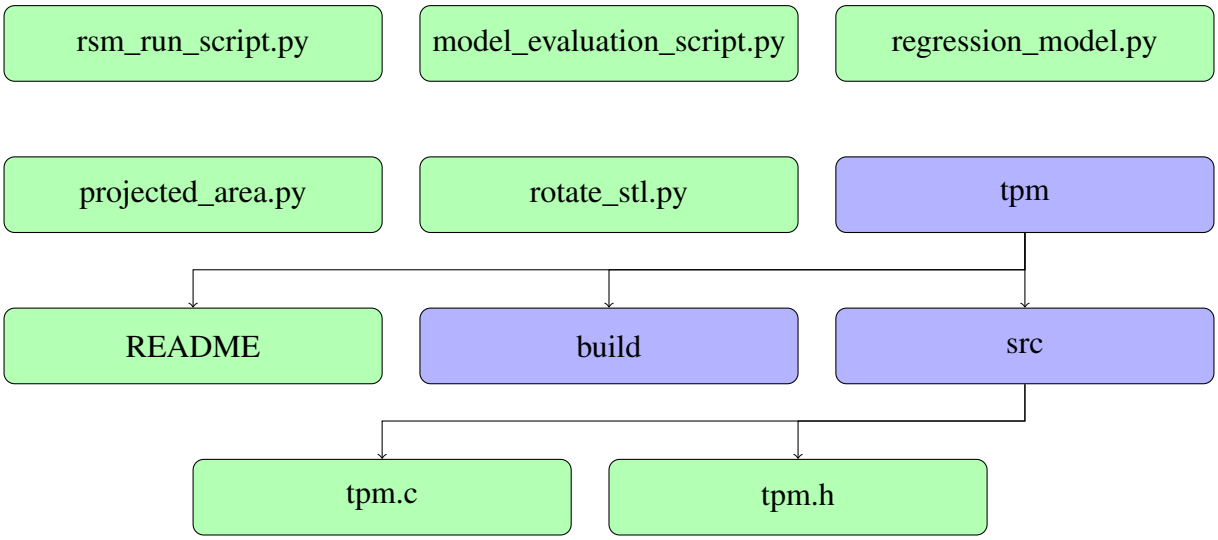


Figure 2.4: RSM_TPMC Directory of WVU RSM

The main scripts that the user will execute are the $rsm\_run\_script.py$ and the *model_evaluation_script.py*. Both of these are executable scripts that the user will have to execute in order create the model then evaluate it. The $regression\_model.py$ and the $rotate\_stl.py$ are ran automatically from the $rsm\_run\_script.py$; however, these can also be ran independently if the user wishes to just rotate an STL or create a regression model.

The tpm folder is the folder that contains the TPMC code. In this folder, the three important items are the $README$, $build$, and src. The README file has the aforementioned steps to compile the code. $build$ is a folder created during installation where the executable TPMC is kept. src contains the $tpm.c$ code and its header file, $tpm.h$. This is the simulation code that is compiled and then ran by the $rsm\_run\_script.py$

3 Simulation Execution

This section of the manual will go over how to setup, execute, and evaluate the code to return drag coefficients and projected area.

3.1 Define Inputs

The software requires multiple user defined inputs to operate, all can be found in the $Inputs$ folder.

3.1.1 Simulation Inputs

In order to define the inputs used for the simulations, the user needs to change the *Simulation.json* file. This file is a JSON file to provide the user with flexibility in inputting the relevant information. For instances, line order and values don't matter. The only thing that matters is the

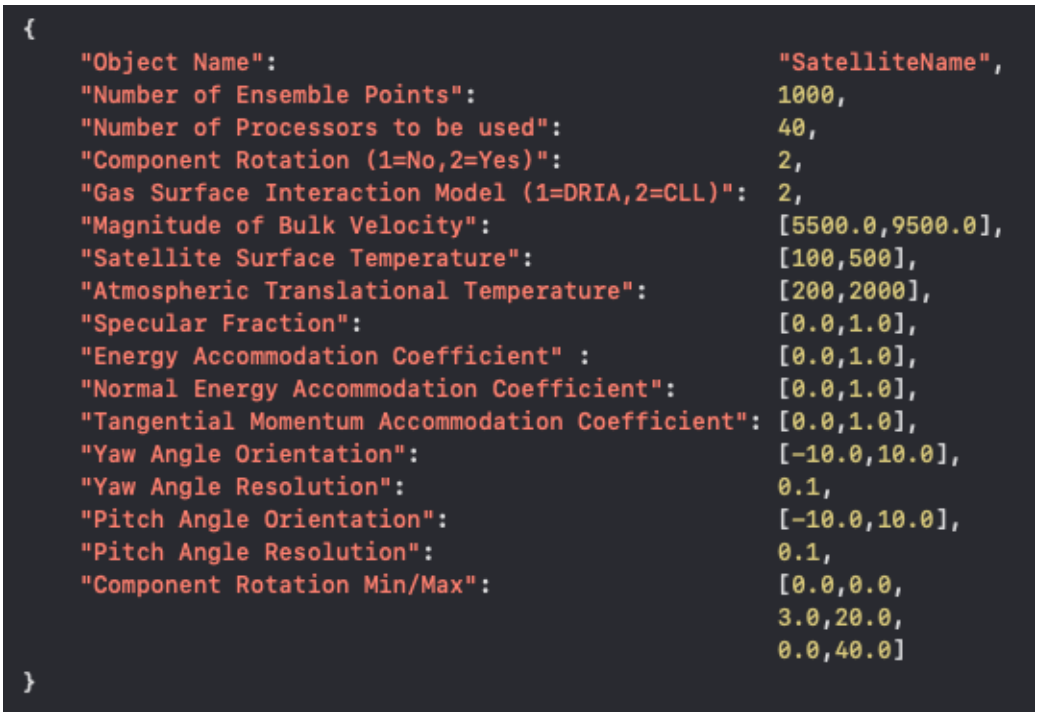


Figure 3.1: Example of the Simulation.json file

format of the file and that the option/variable names be kept the same. Below is an example of the file.

A description of the inputs are as follows:

1. Object Name: This is the name that will be given for the regression model. If not performing rotation, the filename of the STL must be used.
2. Number of Ensemble Points: This is how many simulations will be performed by TPMC to create the regression model.
3. Number of Processors to be used: processors used to run simulation in parallel.
4. Component Rotation: A flag to tell the software if the user wishes to have components of the satellite rotate.
5. Gas Surface Interaction Model: Flag for user to define GSI model.
6. Magnitude of Bulk Velocity: Min/Max value of Velocity sampling; in meters per second.
7. Satellite Surface Temperature: Min/Max value of Surface Temperature sampling; in Kelvin.
8. Atmospheric Translational Temperature: Min/Max value of Atmospheric Temperature sampling; in Kelvin.
9. Specular Fraction: Min/Max value of Specular Fraction; unitless.
10. Energy Accommodation Coefficient: Min/Max value of EAC; unitless.
11. Normal Energy Accommodation Coefficient: Min/Max value of normal EAC; unitless.
12. Tangential Momentum Accommodation Coefficient: Min/Max value of momentum accommodation coefficient; unitless.

13. Yaw Angle Orientation: Min/Max value of yaw sampling; in degrees.
14. Pitch Angle Orientation: Min/Max value of pitch sampling; in degrees.
15. Component Rotation Min/Max: An array for the Min/Max value of each component being rotated. Each row is a component that corresponds to the order of $parent.txt$; in degrees.

Formatting Notes: Formatting of the JSON file is very important. It should be noted that the file requires opening and closing brackets and commas after every value except the last one. The "Min/Max" inputs need to be in brackets and the component rotation array has commas after each number except the last value. This array was put in this form for aesthetic reasons. The number of rows should match the number of components. If the user desires to not rotate a certain component, use zero for the maximum and minimum value.

Note: If the user is not performing a component rotation, use the filename of the mesh for the "Object Name". For example:*"Object Name": "Satellite.stl"

3.1.2 STL_FILES

This folder is where the user should put all of the relevant STL files. Components for rotation and whole objects can be placed in this folder. The appropriate file name will be called either by the *parent.txt* file with rotation or by the *Simulation.json* file object name if no rotation is being performed. Extra STL files in this folder will not affect the operation of the software.

3.1.3 STL_ROTATION_INPUTS

There are three important files in this folder; $hinge\_points1.txt$, $hinge\_points2.txt$, and *parents.txt*. Each of these is a text file that needs to be altered with the evaluation of new structures and components. The user needs a high level of understanding of the structure of their object for this. An example of how this is done is shown later in this document. At the end of this section is an example of each file.

To start, the $parents.txt$ file is how the user defines the component STL file being used. ***Note: The order of the files is very important.** The order the files are listed in this text file dictates the order of the hinge point definitions as well as the order of the "Min/Max" of the component rotation definitions in $Simulation.json$.

The hinge point files define the line of rotation for each component. The files are structured using an x-y-z Cartesian coordinate system where each row is a point corresponding to the component in the same row of $parent.txt$. Each of the two files define a point in space that corresponds to the other file. For instance, $hinge\_points1.txt$'s first row corresponds to $hinge\_points2.txt$'s first row. Theses two points draw an imaginary line about which the component rotates. The line follows the right hand rule. Going from point 1 to point 2, if the user were to point their right thumb in the direction of this vector and curl their fingers, that curl would denote the positive direction of rotation for that component. An example of how these values are used is given under the example section of this manual.

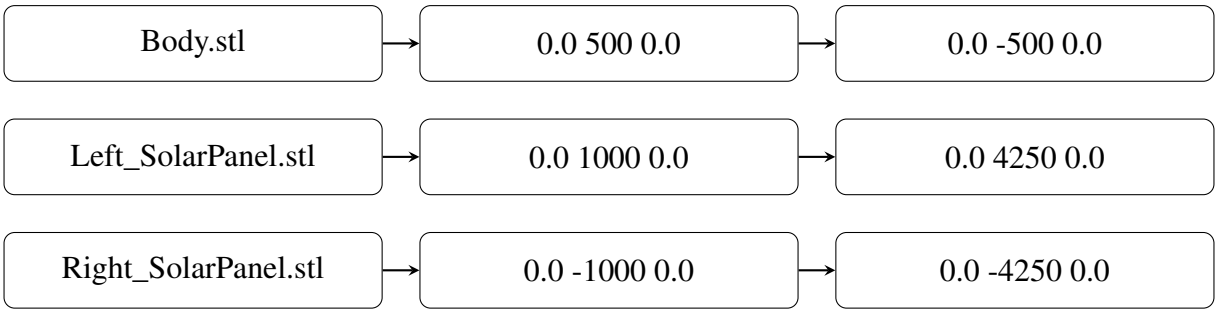


Figure 3.2: Correlation between STL_Rotation_Input Files

```
Body_Cube.stl
LeftPanel_Cube.stl
RightPanel_Cube.stl
```

Figure 3.3: Example of the parent.txt file

```
0.0 500 0.0
0.0 1000 0.0
0.0 -1000 0.0
```

Figure 3.4: Example of the hinge_points1.txt file

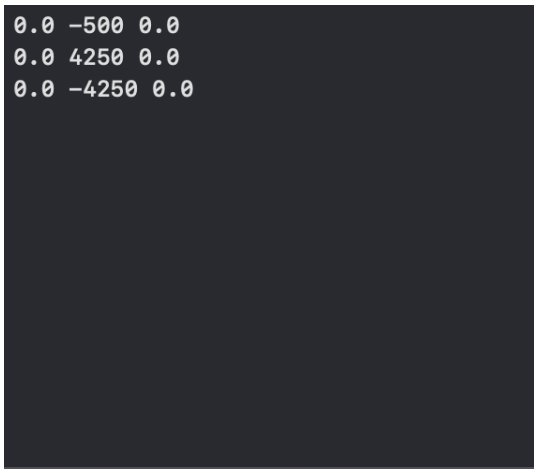


Figure 3.5: Example of the hinge_points2.txt file

3.2 CODE EXECUTION

Executing the code is a simple process. Once the user has loaded their modules and defined their inputs, all that is left to do is execute the software. ***Note: code execution must be done from the top level directory.** Execution needs to be performed from the top level directory so there are no issues with file navigation within the code. To execute input the following line:

```
$ ./RSM_TPMC/rsm_run_script.py --tpm RSM_TPMC/tpm/build/src/tpm
```

A condition with this execution line is that the path of the tpm code needs to be defined. The path to the compiled code is stated after the `--tpm` flag because the user places the compiled code in their desired and personally defined path. Once the code begins the user will see the Response Surface Model logo along with several print statements. Depending on the mesh of the STL file, the number of simulations being performed, and the number of processors being used, the completion of simulations can take anywhere from a matter of hours to several days. A very complex satellite with a fine mesh and an immense number of simulations can even take several weeks to complete.

4 OUTPUTS

In this section, the different outputs from the TPMC simulations as well as regression model will be discussed.

4.1 TPMC SIMULATION

The $tpm.c$ code creates several outputs. First, it creates training and test data for the regression model. With 6 species of gas, there will be 6 training and 6 test files. These 12 files are placed in the following directory: *./Outputs/RSM_Regression_Models/data*. The test and training data reflect the variables created from Latin Hypercube Sampling for each simulation as well as the calculated drag coefficient.

Furthermore, the $tpm.c$ code outputs the projected area of each simulation. The area is saved to the $Aout.dat$ file located in the $./Outputs/Projected\_area$ directory. This file is then used by the area lookup code to calculate the projected area given by the user inputs.

4.2 Regression Model

The regression model code uses the training and test data files to develop and test the model. Verification of successful model creation is shown through several plots placed in the following directory: *./Outputs/RSM_Regression_Models/Plots_Output*.Below is an example of the O2 model created for the satellite GRACE.

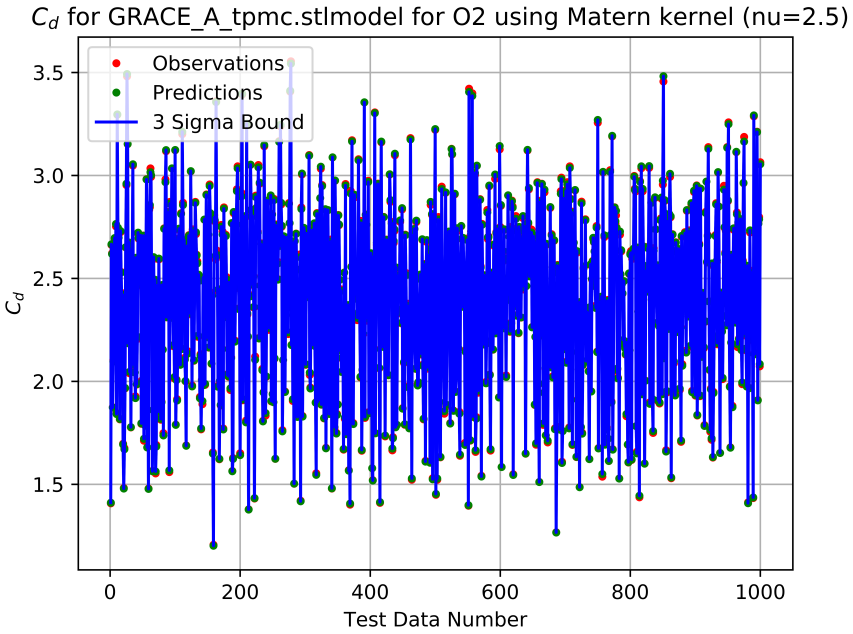


Figure 4.1: Visualization of Model Performance

4.3 TEMPFILES

4.3.1 Latin Hypercube Sampling

Latin Hypercube Sampling is used to make randomized data points for the TPMC simulation. The output of this process is worth noting even though the user does not need to interact with these outputs. In the $tempfiles$ folder, two files are created; $tpm.ensemble$ and $tpm.ensemble.h5$. These files are the same but the HDF5 file is used particularly for the metadata included in the file type. The HDF5 is not visible without using the `$ h5dump` command or an HDF5 viewer. The user can easily view the tpm.ensemble file with any common text editor or reader though. If the user wishes to see the the deflections of each component further they may reference the $deflections.txt$ file in the $Outputs$ folder. This file contains similar information as the ensemble files; however this structure is more intuitive. The order of this file also correlates to the order of the mesh file created during rotation. For instance, if the first mesh file

created was Satellite0.stl and the first line of $deflections.txt$ was "10 20 30". Then opening and physically viewing Satellite0.stl would show the first component rotated 10 degrees, the second 20 degrees, and the third 30 degrees.

4.3.2 MESH_FILES

As the rotation script creates STL files for the $tpm.c$ code, they are placed in the $Mesh\_Files$ folder. The folder is not automatically cleaned after the software is finished. For now, the files are not deleted so the user can view the STLs if they wish to do so. Since there are so many STLs created for a large number of simulations, the user should clean this folder periodically. The user also does not need to interact with these files. Each one is checked for water-tightness as it is created. If there are any issues, the code will automatically exit and notify the user of the error.

4.3.3 TEMP_VARIABLES.TXT

$temp\_variables.txt$ is used to essentially pass variables from the Python code to C code. The variables that are passed are the following: Object Name, Gas Surface Interaction flag, Species Mole Fraction, and the Component Rotation flag. The user does not need to interact with these and changing them manually will have no effect on how the software operates.

4.4 REGRESSION MODEL OUTPUT

The regression model has a lot of outputs that the user will need to interact with. Once the model is created the appropriate outputs are placed in the *./Outputs/RSM_Regression_Models* folder. There are thirty files created from this process in total, each sharing the object name used in the $Simulation.json$ file. Each species requires two files for mean and standard deviation, as well as a Gaussian Process Matern model. The only interaction the user must perform with these files is to move them to the *./Inputs/RSM_Regression_Models* folder in order for the model evaluation script to use the correct regression model.

4.5 MODEL EVALUATION OUTPUTS

After the user executes the model evaluation code, there are only two outputs; the drag coefficient and the projected area. These are named according to the $Model\_Evaluation\_Inputs.json$ file. Drag coefficient is placed in the $Cd\_Results$ folder and projected area is placed in the $Projected\_Area$ folder. These are the final results for the user. They can be visualized through plotting and/or used for orbit propagation as the user sees fit.

5 MODEL EVALUATION

Evaluating the model is the pinnacle of the user obtaining results. The user has a couple of action items to perform before being able to do so. First are the inputs; there are three input items to execute the evaluation. Once these are set, the user just needs to execute the code similarly to the simulation execution. Finally, the two outputs are placed in the $Outputs$ folder.

5.1 Model Evaluation Inputs

Under the $Inputs$ folder, there are three inputs that need to be set before execution; the JSON file, the CSV file and the regression models. The regression model is a simple setup. The user must take the regression model created in the simulation execution steps and move them to the following directory: $./Inputs/Model\_Evaluation\_Inputs/Regression\_Models$

5.1.1 Model_Evaluation.json

This file is very similar to the $Simulation.json$ file. The structure requires the same JSON formatting as before; however, there are no arrays in this file. This is worth noting as the user should be wary that the last value has no comma after it. The following is an example of the file.

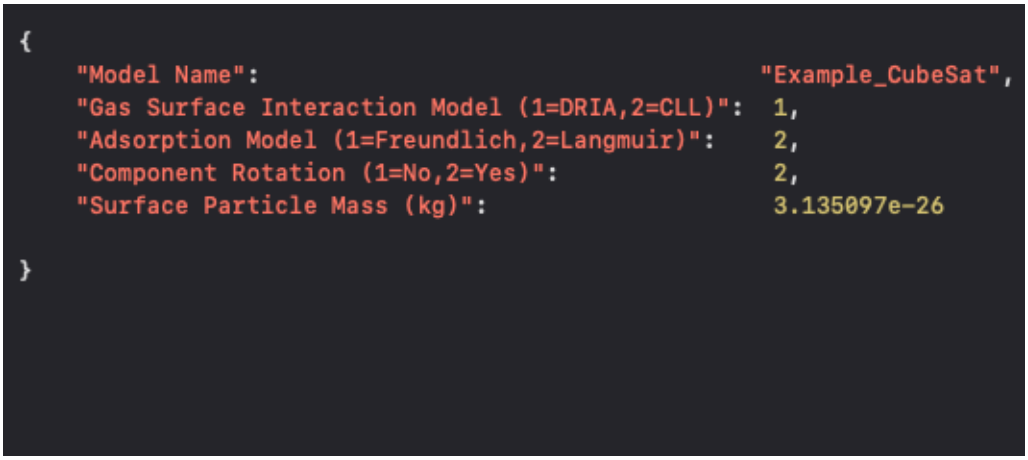


Figure 5.1: Example of the Model_Evaluation_Inputs.json file

The first input of this file is the "Model Name". This is not a name the user is assigning to anything. This is the model name the precedes all of the regression model files. For example, if the user had regression models name $Fun\_wTPMC\_GP\_reg\_H\_matern2.pkl$, $Fun\_wTPMC\_x\_train\_H\_mean.pkl$, $Fun\_wTPMC\_y\_train\_O2\_mean.pkl$, etc. then "Model Name": "Fun_wTPMC" would be the appropriate input. If the user performed simulation without rotation, then the caveat of using the .stl extension carries over to this input file. For example, the user could have an input such as "Model Name": "Satellite.stl".

The next input is the GSI model. This must correlate with the GSI used when the simulation was preformed. If these do not match then the program will throw an error. Similarly, the Component Rotation input here must also match what was used for the model. These two inputs are important because they determine the number of variables as well as the number of dimensions used, respectively.

The adsorption model input is at the users discretion. Using either of these determines how the fractional surface coverage of adsorbed oxygen is perceived. A snippet of the code that calculates the fractional surface coverage is shown below.

```
print('Calculating Cd with ADS model')
#Compute partial pressure of oxygen
Po[:,0] = n[:,0]*mole_frac[:,4]*kB*Inputs[:,2]
if ads_model == 1: #Freundlich
    fsc = kF_CLL*(Po**aF_CLL)
    if fsc>1.0:
        fsc = 1.0

elif ads_model == 2: #Langmuir
    if GSI_MODEL == 1:
        fsc = (kL_DRIA*Po)/(1+kL_DRIA*Po)
    elif GSI_MODEL == 2:
        fsc = (kL_CLL*Po)/(1+kL_CLL*Po)
else:
    print('Enter valid ADS Model Flag')

Cd_TOTAL = fsc[:,0]*Cd_ads[:,0] + (1.0-fsc[:,0])*Cd_srf[:,0]

return Cd_TOTAL
```

Figure 5.2: Code from model_evaluation_script.py, calculation of the fractional surface coverage of adsorbed oxygen with different ADS models.

The last input is the satellite surface particle mass. The surface particle mass is used for the calculation of the energy accommodation coefficient for clean surfaces based on Goodman's (1966) empirical formula. This is a value that can vary from satellite to satellite and is dependent of the satellite's material and finishing.

5.1.2 MODEL_INPUT_DATA.CSV

This file is where the user may input as many data points as they desire to find the drag coefficient at any given instance. A *csv* file was chosen to allow the flexibility of inputting a single instance, multiple instances, or time series data as well as the ability to allow for n-dimensions from the component rotation. Each column of the file is an input and denoted in the header of the file. The inputs are as follows:

- Velocity (m/s)
- Surface Temperature (K)
- Atmospheric Temperature (K)
- Yaw (radians)
- Pitch (radians)
- Number Density ($\#/m^3$, the number of particles per meter cubed)
- Mole Fraction (He)
- Mole Fraction (O)
- Mole Fraction (N2)
- Mole Fraction (O2)
- Mole Fraction (H)
- Mole Fraction (N)
- Component Rotation... (degrees)

Each row of the file is considered to be one instance of evaluation. The CSV file requires that each of these columns remain in this particular order also. The component rotation flag that is used in the $Model\_Evaluation\_Inputs.json$ tells the code if the columns after "Mole Fraction(N)" are relevant or not. If the flag is turned off then these columns will not be read. It is important that the number of "Component Rotation..." columns matches the number of components being used in the simulations. All of the inputs prior to the component rotation inputs are used to calculate the drag coefficient of the satellite. The component rotation inputs themselves are relevant to find the projected area of the satellite. ***Note: Formatting of the CSV file can leave erroneous commas with certain editors such as Microsoft® Excel. If an error arises, check the formatting with a plain text editor such as Microsoft® Notepad or TextEdit.**

5.2 Execution of Model Evaluation

Executing the model evaluation script is very similar to the RSM run script. A difference is that the evaluation does not require any C code. Therefor a simple Python execution is necessary. Similar to other code executions, the call must be made from the top level directory.

```
$ ./RSM_TPMC/model_evaluation_script.py
```

5.3 Outputs of Model Evaluation

There are two outputs to the model evaluation; the drag coefficient and the projected area of the satellite. Each of them are placed in the $Outputs$ folder in their respective folders; $Cd\_Results$ and $Projected\_Area$. The output files are text files with the output printed as a column of data in the same order as the CSV file used at input.

6 Example

This section of the user manual will be a walk-through of calculating the drag on a simple cube-satellite with rotating solar panels. It is assumed that the user has downloaded, installed, and setup the RSM software already. The satellite for evaluation is shown below.

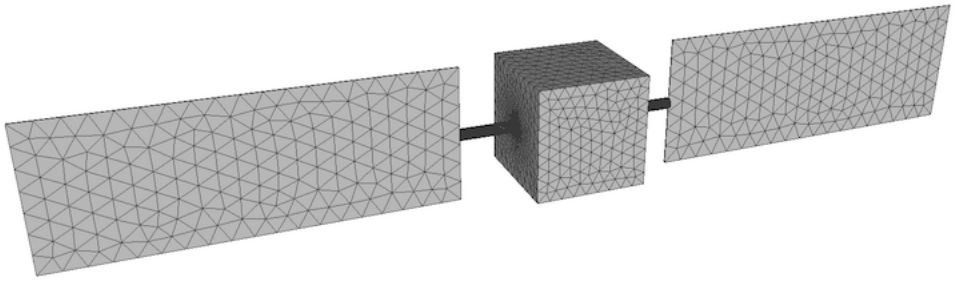


Figure 6.1: Structure and Mesh of example Cube-Sat with Solar Panels

This satellite is made up of three distinct components; the left solar panel, the right solar panel, and the body. The body is compromised of the cube and the cylindrical arms on each side. Because the cylindrical arms are rounded, the mesh has be made much finer in these areas. The entire structure is made up of 10025 facets in total.

6.1 Creating an STL File

In order to create the actual structure of the object a CAD software is necessary. To create this satellite DS Solidworks® was used. The user may use which ever software they desire to accomplish this task, but there are some notable steps to making sure the final structure is comprised of each component correctly.

The coordinate frame each component is made in is vital to making sure the final structure is built correctly. For the case of the satellite being built here, the center of the cube is (0,0,0). The user should also remember that the orientation of the reference frame is important for simulation (see section 1.2.1). For this example, the satellite is very symmetrical so the orientation is only important for the consideration of what direction the solar panels are rotating.

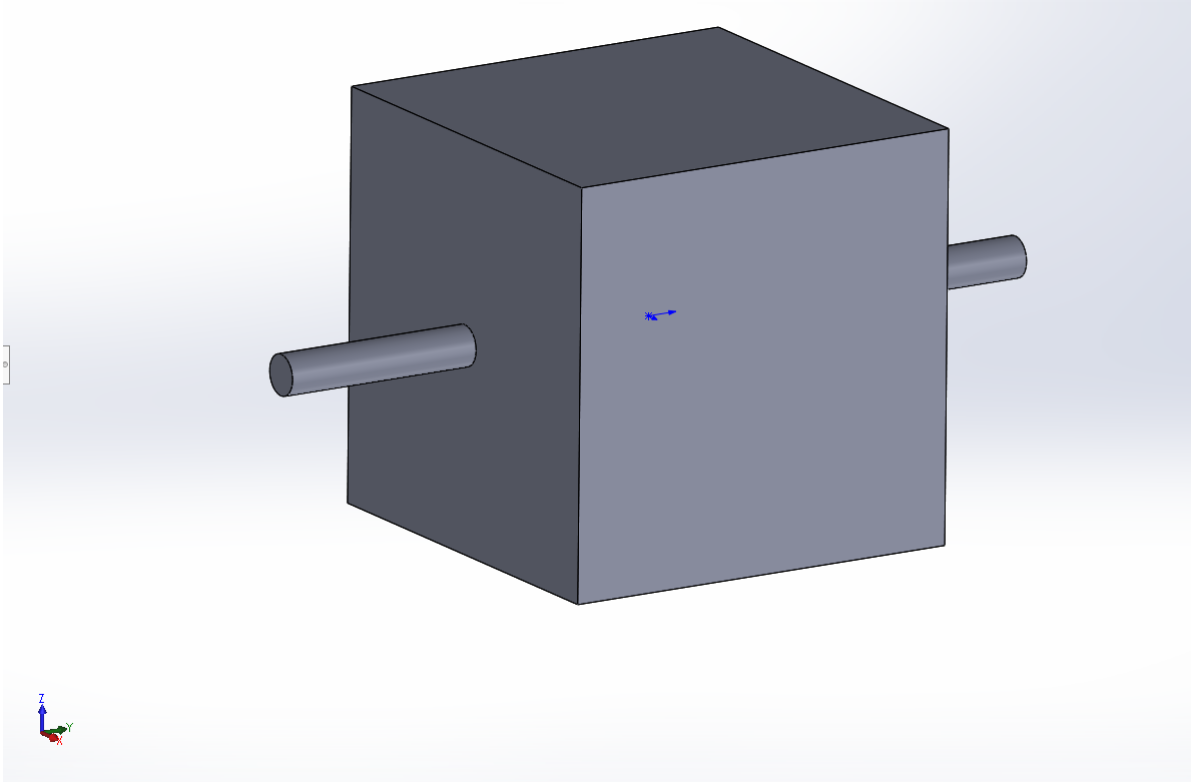


Figure 6.2: Body Component of Example Cube-Sat

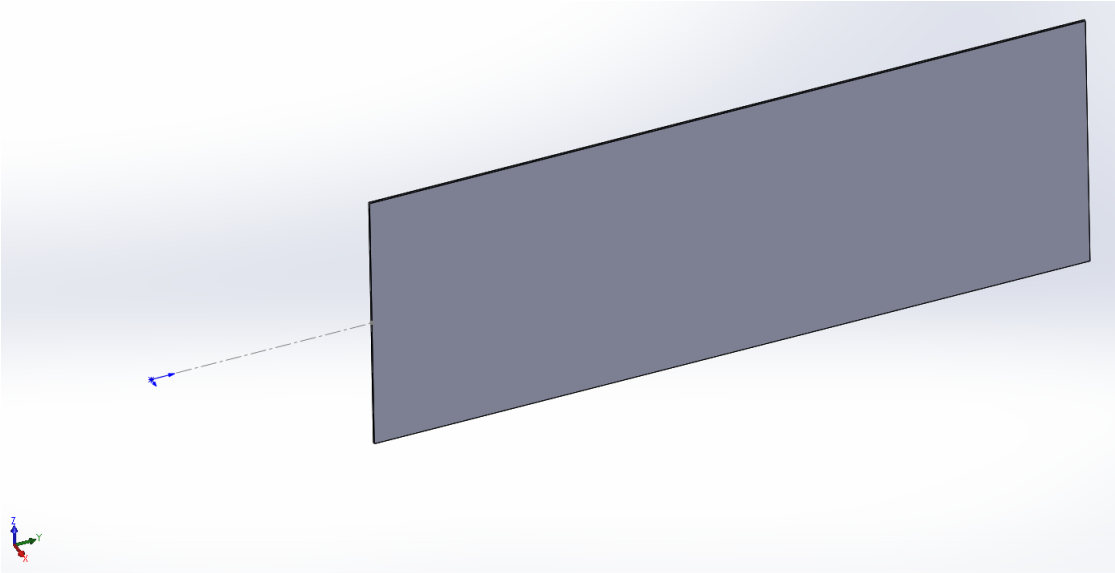


Figure 6.3: Left Solar Panel Component of Example Cube-Sat

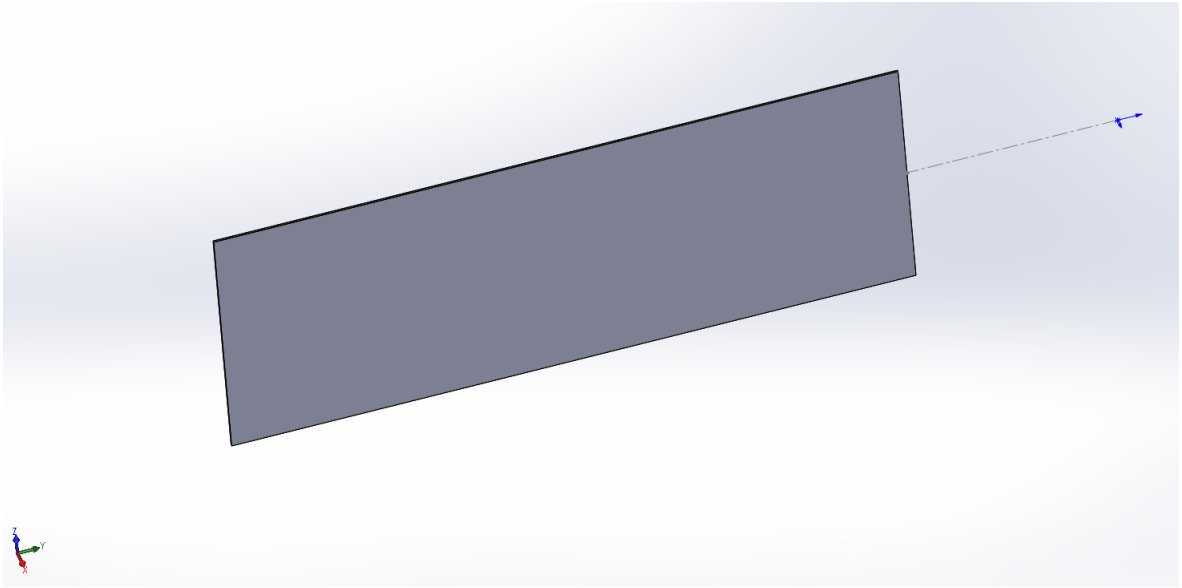


Figure 6.4: Right Solar Panel Component of Example Cube-Sat

Note that each of the solar panels is 1 meter from the center, as shown by the dotted line. Since the body is a cube that is 1x1x1m and each arm is .5m long, the left panel connects at (0,1000,0) and the right panel connects at (0,-1000,0).***Note: When exporting components from CAD software, some CADs will translate the component so that it is centered to (0,0,0), do not allow this**. Most CADs allow the user to turn off translation.

After the structure has been developed the body must be meshed. Most CAD softwares, such as DS Solidworks®, allow the user to export directly to an ASCII STL file. The user should be wary of this as the STLs created from CAD are not necessarily high fidelity meshes. To be sure of an even and controlled mesh the user should use a meshing software. Here, the open source software GMSH was used. An example of the mesh generated for the body of the satellite is shown below.

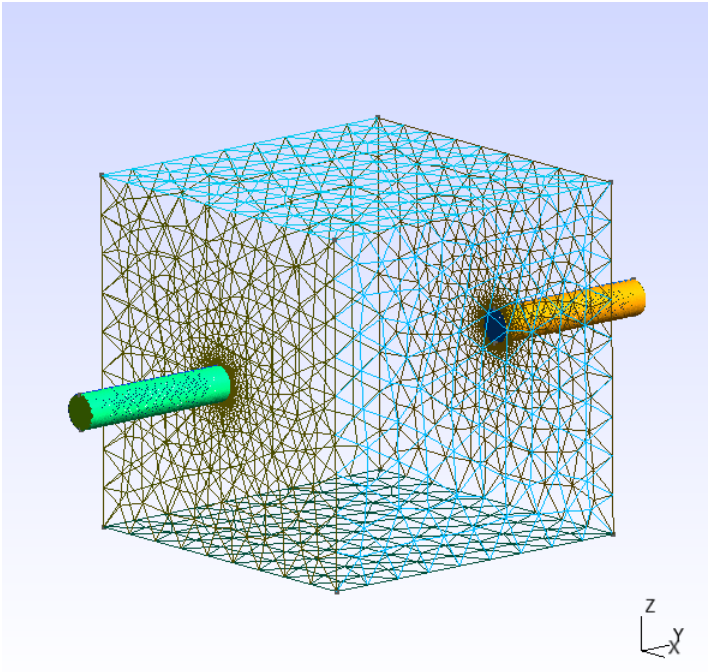


Figure 6.5: Mesh for Body Component of Example Cube-Sat, GMSH

6.2 Example Inputs

After creation and meshing of the components. The user can perform the setup for the simulations. Navigate to the $Inputs$ directory. First, place the components that were created into the STL_Files folder. Secondly set up the $Simulation.json$ file to the desired ranges. For this example, the $Simulation.json$ is shown below. 1000 ensemble points will be used with 200 processors. The sampling for yaw and pitch are $\pm$ 10, the component rotation is 0:0 for the body and 0:20 for each panel.

```
{
    "Number of Ensemble Points":                    1000,
    "Number of Processors to be used":              200,
    "Component Rotation (1=No,2=Yes)":              2,
    "Gas Surface Interaction Model (1=DRIA,2=CLL)": 2,
    "Magnitude of Bulk Velocity":                   [5500.0,9500.0],
    "Satellite Surface Temperature":                [100,500],
    "Atmospheric Translational Temperature":        [200,2000],
    "Specular Fraction":                            [0.0,1.0],
    "Energy Accommodation Coefficient" :            [0.0,1.0],
    "Normal Energy Accommodation Coefficient":      [0.0,1.0],
    "Tangential Momentum Accommodation Coefficient": [0.0,1.0],
    "Yaw Angle Orientation":                        [-10.0,10.0],
    "Yaw Angle Resolution":                         0.1,
    "Pitch Angle Orientation":                      [-10.0,10.0],
    "Pitch Angle Resolution":                       0.1,
    "Component Rotation Min/Max":                   [0.0,0.0,
                                                    0.0,20.0,
                                                    0.0,20.0]
}
```


Figure 6.6: Simulation Input for Example

Next the user needs to define the rotation inputs. Recalling there are three input files for rotation, the files are shown as below.


```
Body_Cube.stl
LeftPanel_Cube.stl
RightPanel_Cube.stl
```


(a) parent.txt


```
0.0 500 0.0
0.0 1000 0.0
0.0 -1000 0.0
```


(b) hinge_points1.txt


```
0.0 -500 0.0
0.0 4250 0.0
0.0 -4250 0.0
```


(c) hinge_points2.txt

Figure 6.7: Example Rotation Inputs

The user should recall that the hinge points follow the right hand rule. An illustration of the

hinge point setup is shown below.

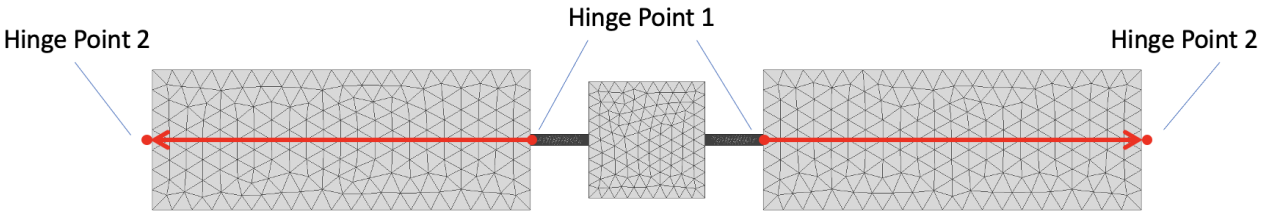


Figure 6.8: Simulation Input for Example

For this example, it was chosen that both inner points are the first hinge point and the outer points are the second. This is to illustrate the right hand rule. In the input file 0:20 was chosen for both, this will result in the solar panels rotating in opposite directions. The figure below shows the positive direction of rotation for each panel.

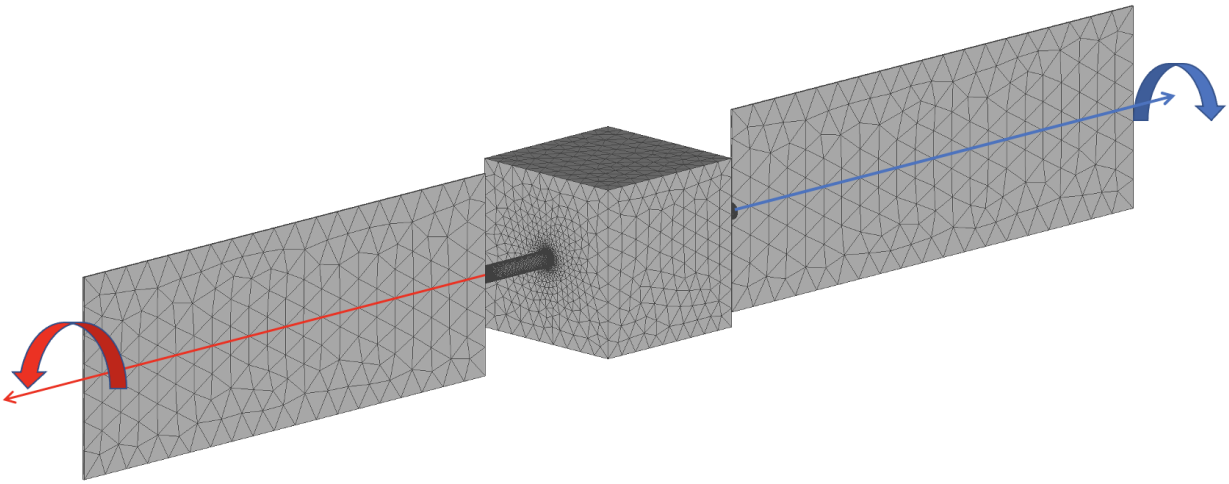


Figure 6.9: Rotation Direction for Example

6.3 Example Execution

For this simulation, GCC version 9.3.0 compilers were used along with Intel 2019 Python compilers, version intelpython3_2019.5. Refer to section 2 for module usage. The code is executed using the following command:

```
$ ./RSM_TPMC/rsm_run_script.py --tpm RSM_TPMC/tpm/build/src/tpm
```

Once executed the user will see the RSM logo along with some print statements. The user may also see some debugging information, this can be ignored and is for developer purposes. An example of the "Watertight" statement can also be seen in the figure below. When the rotation software creates the STL files for the TPMC simulation they are named with an integer; e.g. Satellite0.stl. The function that checks water-tightness instead outputs the angle deflections, in order of each component, attached to the name; e.g. Satellite_0.0_-5.0_10.235.stl. This can be useful if the user is having a certain rotation that is throwing an error.

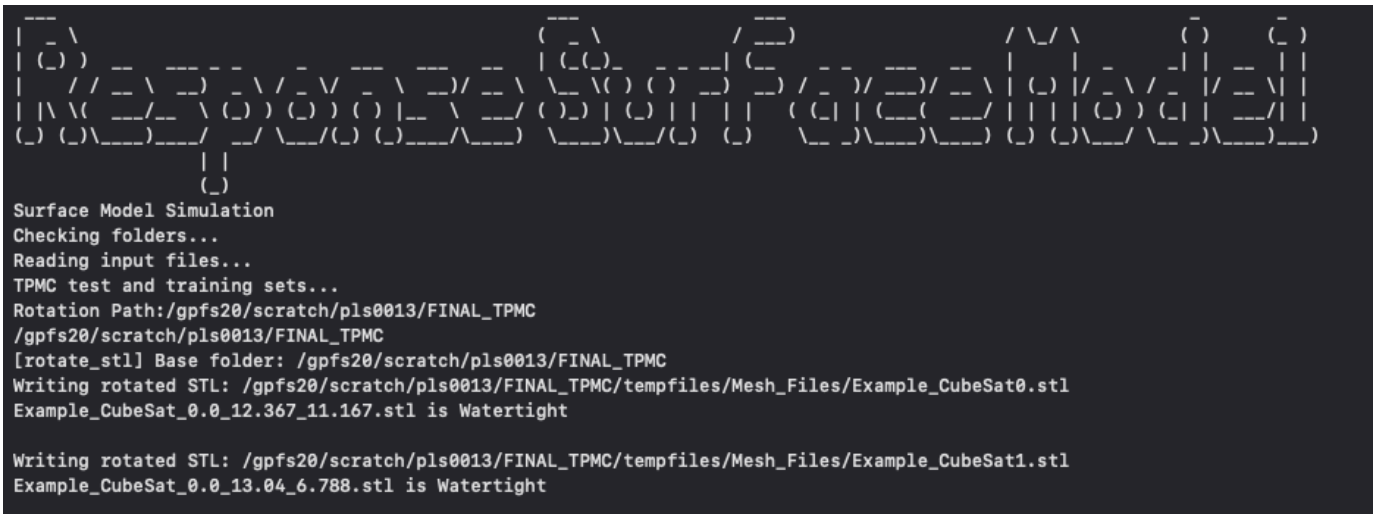


Figure 6.10: RSM Logo and Initial Output

6.4 Example Outputs

Once the RSM has finished running. The aforementioned outputs will have automatically been created. The user will need to move the regression models in the *Output* folder to the *Model_Evaluation_Inputs* folder.

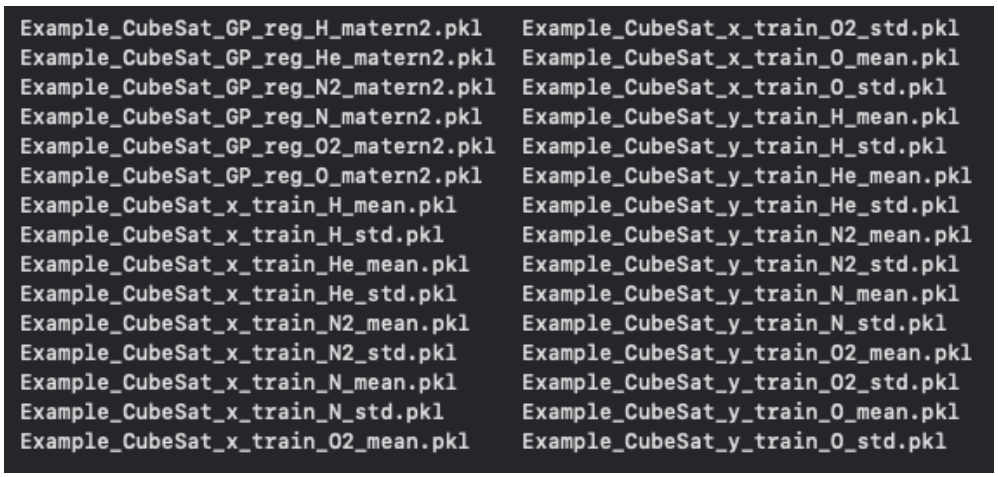


Figure 6.11: Thirty Files for the Example Regression Model

6.5 Example Model Evaluation

From here, the model is ready to be evaluated. The CSV file needs to be set up with the desired evaluation points and the JSON file needs to reflect the proper name and values. Headers in the CSV example are stacked for visual purposes only. The headers should be in a single row in the actual file.

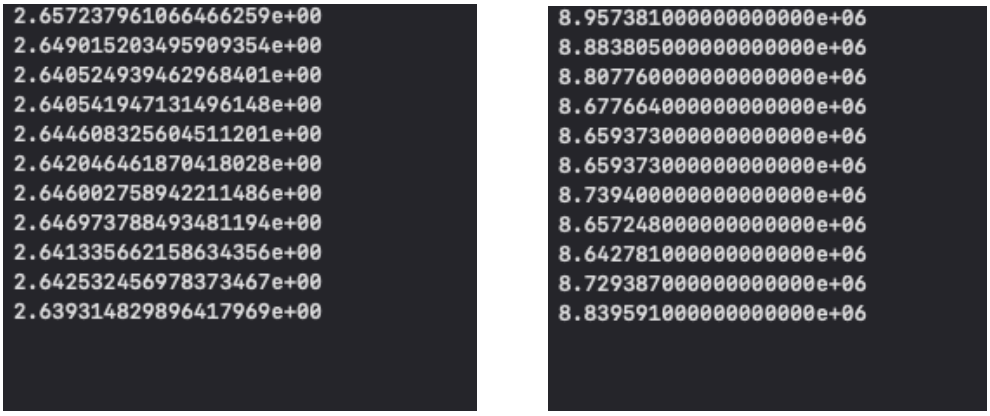

```
{
    "Model Name":                                      "Example_CubeSat",
    "Gas Surface Interaction Model (1=DRIA,2=CLL)":    1,
    "Adsorption Model (1=Freundlich,2=Langmuir)":      2,
    "Component Rotation (1=No,2=Yes)":                 2,
    "Satellite Mass (kg)":                             100.00,
    "Surface Particle Mass (kg)":                      3.135097e-26

}
```


Figure 6.12: Example Model_Evaluation.json file

Velocity	Temperature (K)	Atmospheric Temperature (K)	Yaw	Pitch	Density (#/m^3)	Mole Fraction: He	Mole Fraction: O	Mole Fraction: N2	Mole Fraction: O2	Mole Fraction: H	Mole Fraction: N	Component Rotations...:		
7665.73083	300	1196.0596	0.049679	-0.004151	3.44E+14	0.015457	0.889884	0.072458	0.001046	0.000151	0.021004	0	7	5
7665.90411	300	1195.1422	0.050152	-0.00426	3.44E+14	0.015585	0.890482	0.071902	0.001034	0.000152	0.020846	0	8	10
7666.07134	300	1163.8846	0.050376	-0.004166	3.19E+14	0.016939	0.894891	0.067694	0.000952	0.000183	0.01934	0	9	15
7666.23274	300	1163.1406	0.050544	-0.004006	3.19E+14	0.017074	0.895354	0.06723	0.000943	0.000184	0.019215	0	10	20
7666.38841	300	1162.4384	0.050843	-0.003989	3.20E+14	0.017211	0.895796	0.066782	0.000934	0.000185	0.019093	0	11	23
7666.53832	300	1161.7753	0.051026	-0.00384	3.20E+14	0.01735	0.896218	0.066348	0.000925	0.000185	0.018974	0	12	26
7666.68231	300	1161.1489	0.05144	-0.003941	3.20E+14	0.017491	0.896619	0.065928	0.000917	0.000186	0.018859	0	13	30
7666.82019	300	1160.5573	0.051785	-0.003992	3.20E+14	0.017634	0.897001	0.065522	0.000909	0.000187	0.018748	0	14	32
7666.95195	300	1159.9973	0.052181	-0.004078	3.20E+14	0.01778	0.897365	0.065127	0.000901	0.000187	0.01864	0	15	19
7667.07754	300	1159.4667	0.052597	-0.00421	3.20E+14	0.017928	0.89771	0.064744	0.000894	0.000188	0.018535	0	16	15
7667.19687	300	1158.9626	0.05296	-0.004292	3.20E+14	0.018079	0.898038	0.064373	0.000887	0.000189	0.018434	0	17	9.5

Figure 6.13: Example CSV file

Eleven points were used to evaluate the model, therefore there should be eleven projected areas and eleven drag coefficients output. In the *Outputs* folder there is now a file for each. In the projected area file below, it can be seen that each area is on the order of 10^6. This is because the STL file used values that are in millimeters.



(a) C_d Results (b) Area Results

Figure 6.14: Example Results

REFERENCES

[1] Davis, D.H. *Monte Carlo Calculation of Molecular Flow Rates Through a Cylindrical Elbow and Pipes of Other Shapes*. Journal of Applied Physics, Vol. 31, 1961, pp. 1169-1176.

[2] Mehta, P. M., Walker, A., Lawrence, E., Linares, R., Higdon, D., & Koller, J. (2014). *Modeling satellite drag coefficients with response surfaces*. Advances in Space Research, 54(8), 1590–1607. doi:10.1016/j.asr.2014.06.033

[3] Mehta, P.M., Walker, A., McLaughlin, C.A., Koller, J., 2014. *Comparing physical drag coefficients computed with direct simulation Monte Carlo using different gas–surface interaction models.* J. Spacecraft Rockets 51 (3), 873–883.

[4] Pilinski, M., Argrow, B., Palo, S., 2011. *Drag coefficients of satellites with concave geometries: comparing models and observations.* J. Spacecraft Rockets 48 (2), 312–325.

[5] James A. Tancred *ROTATESTL: A MATLAB ROTATION ALGORITHM FOR THE ANALYSIS OF COMPUTATIONAL MESHES IN STEREOLITHOGRAPHY FILE FORMAT* Air Force Research Laboratory, Aerospace Systems Directorate, Wright-Patterson Air Force Base, Air Force Materiel Command, United States Air Force.