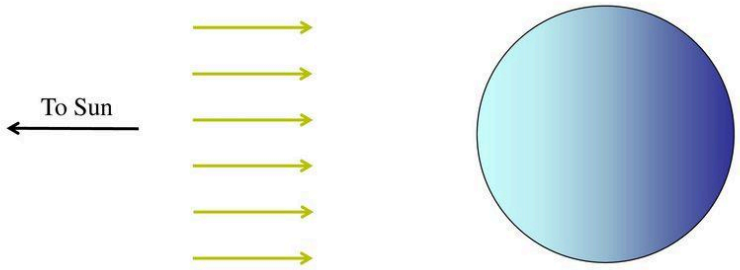
Model drag and SRP forces on a satellite

A satellite may experience drag and SRP (solar radiation pressure). We will consider different models for a satellite.

Cannonball model

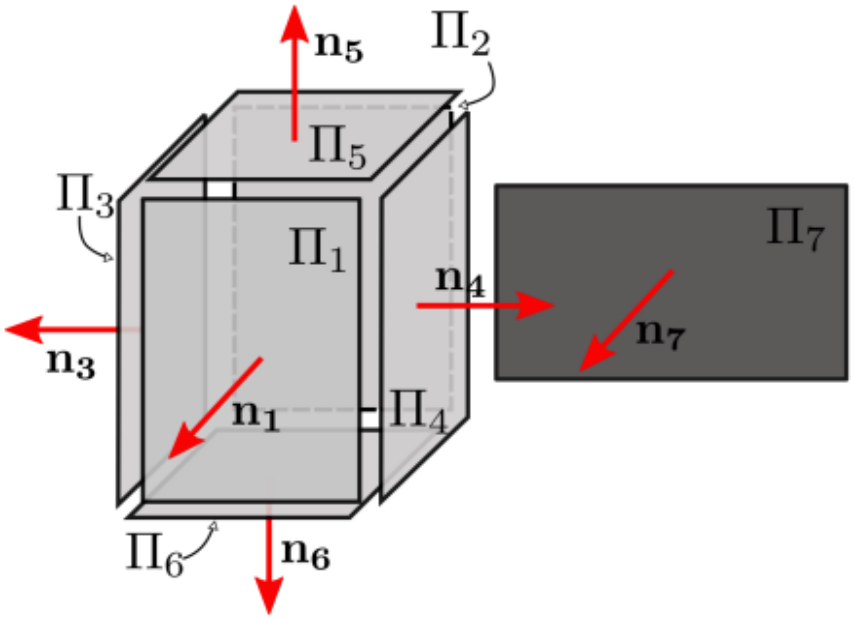
Models satellite as a sphere, and only depends on area to mass ratio and surface reflectivity properties. However it is not very realistic or accurate of a satellite geometry.

(insert SRP acceleration eq)

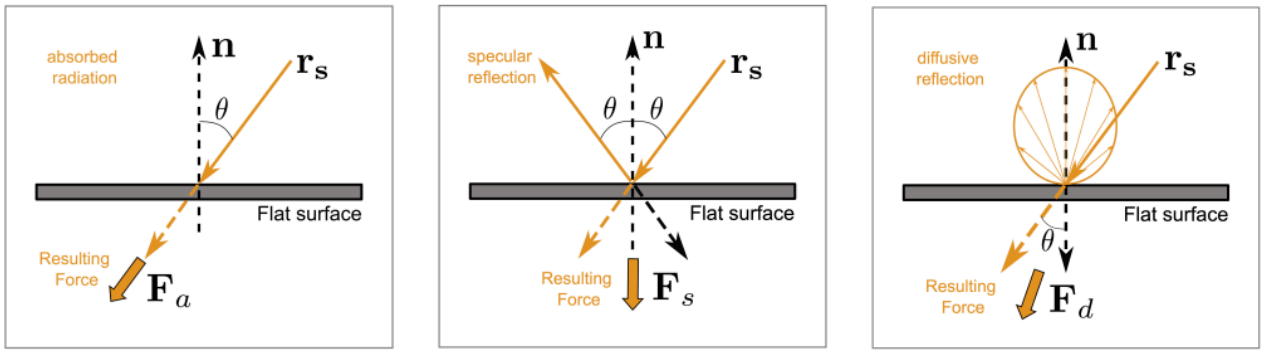


N-Plate model

Models satellite into flat plates on its surface. We can calculate force and torque for each plate. Does not consider parts that are shadowed (auto-occultation). One of the advantages of this approach is that the SRP acceleration depends on the satellite's attitude. More representative of satellite geometry.



SRP acceleration can be calculated by considering the reflection behaviour of particles in a free stream contacting the flat plane.

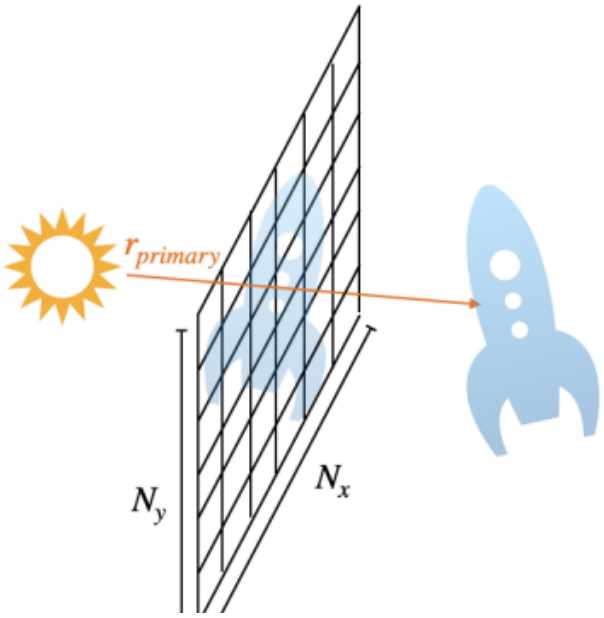


We can calculate the forces produced from absorption, specular reflection and diffuse reflection
(insert forces and SRP acceleration).

Ray tracing method

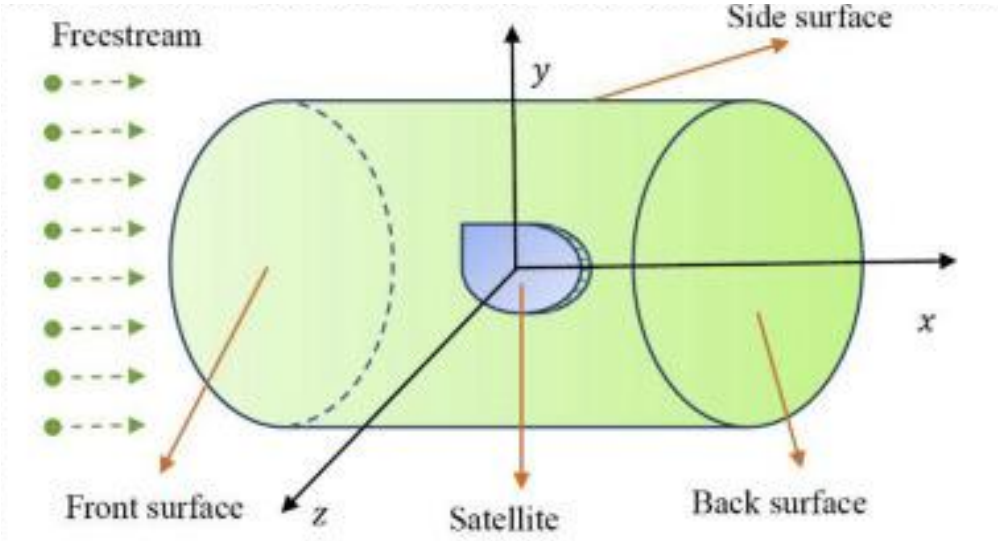
Models satellite in high quality. Uses ‘light rays’ to shine on satellite and detect light/shadow surfaces of satellite onto image plane. Then use the n-plate (or finite element) method to calculate force on only light parts.

We consider a plane at a certain distance from the satellite (a pixel-array) that represents the solar flux. Then we take a grid of points in this pixel-array and from each point we project a ray and check if it intersects any of the triangles/polygons on the satellite’s surface. In this way we determine which parts of the satellite are illuminated.



Monte Carlo method

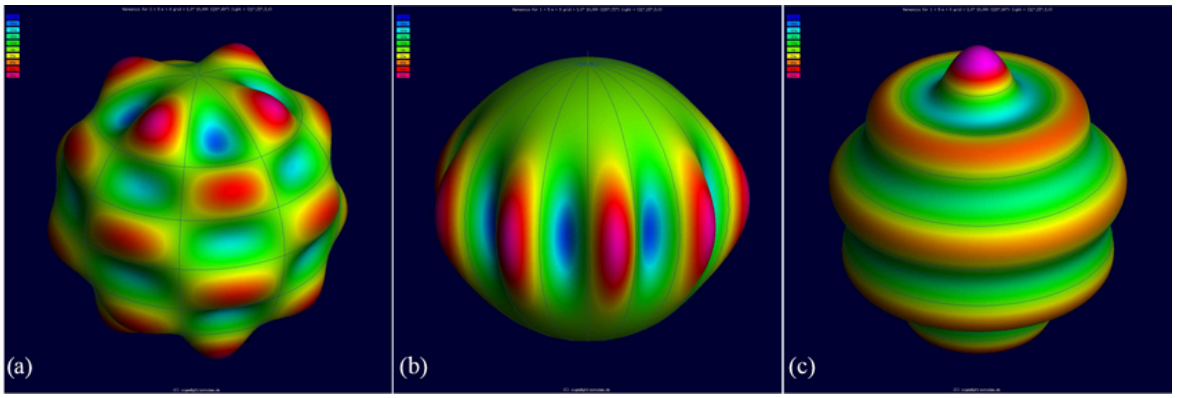
Like the Ray tracing method but uses ‘particles’ to project on satellites. Each test particle represents thousands of molecules. Different methods can let the particles interact with each other or not to simulate different flows.



Spherical harmonics

We can use spherical harmonics to find SRP acceleration as an alternative to the n-plate method. This can be used when the attitude of the satellite depends on two angles.
The Spherical Harmonics are a set of orthonormal functions defined on the sphere, and can be used to represent functions defined on the surface of a sphere.

They can model the shape of a complex geometry by overlapping each other to create more intricate shapes. When we reach a very complex shape such as a satellite, it
it becomes very difficult to do this.



Despite this, it is significantly quicker than the finite element method as the computationally expensive calculations only need to be made once and it reduces error as spherical harmonic degrees increase

We will use the ray tracing method. This is computationally expensive so will develop a neural network to speed it up.

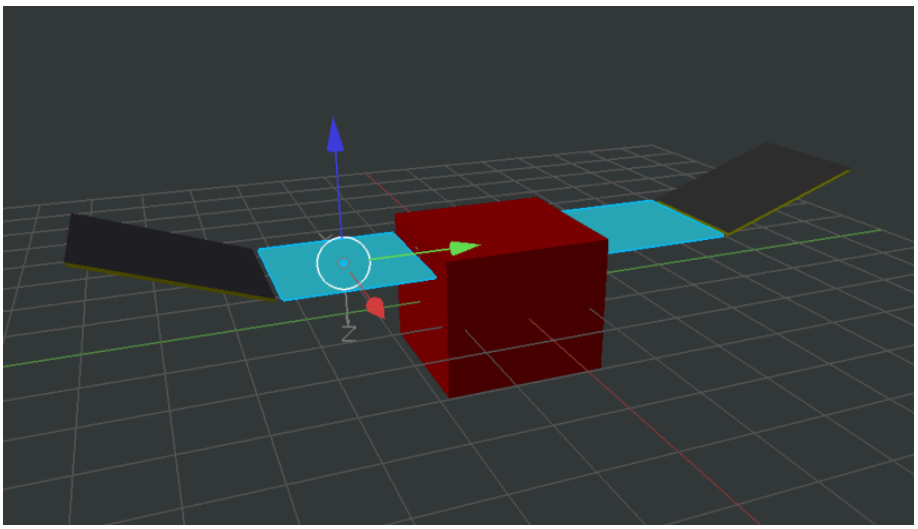


Table or run times for 3D CAD model of MAVEN

Table 4. For MAVEN, different SH degrees and the N-plate model. Top: run time for the computation of 1369 different attitudes. Bottom: maximum error of approximation.

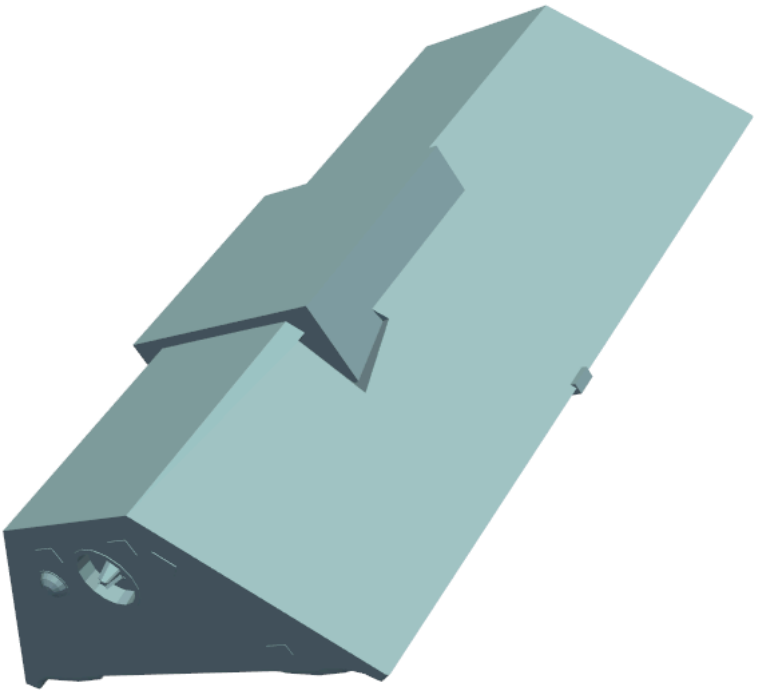
	FE	N-Plate	SH deg 4	SH deg 8	SH deg 16
Run Time	2m 22.297s	0m 0.007s	0m 0.006s	0m 0.009s	0m 0.014s
Max Error	–	9.0995	4.1791	2.1514	2.0483

Modelling

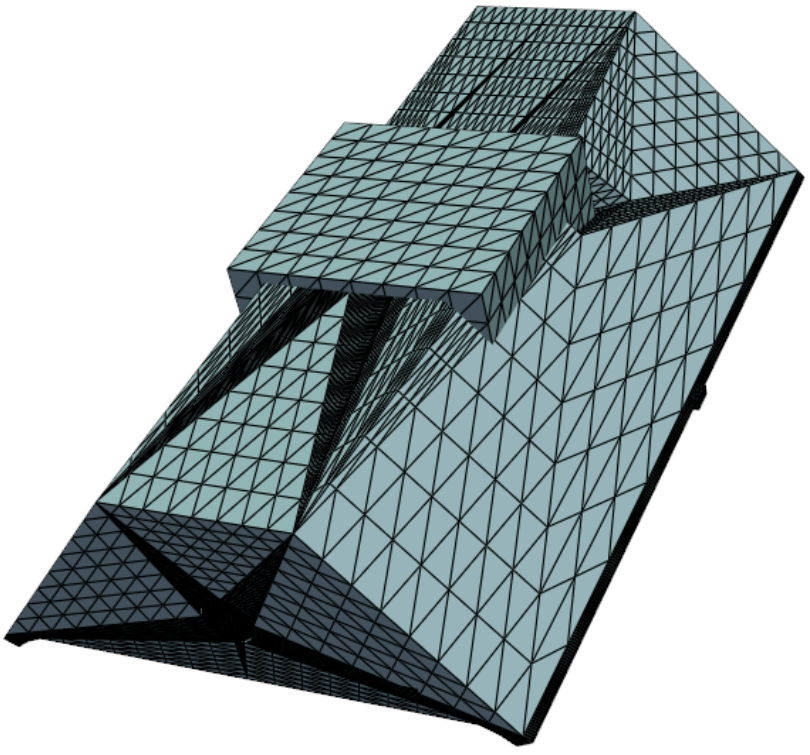
I used a model of the GRACE satellite and created a triangular mesh to be the surfaces we measure using the ray tracing method.

We considered using the light function on Pyvista instead of computing ray tracing directly. However the software was not able to provide the data we need about light and shadow of the satellite.

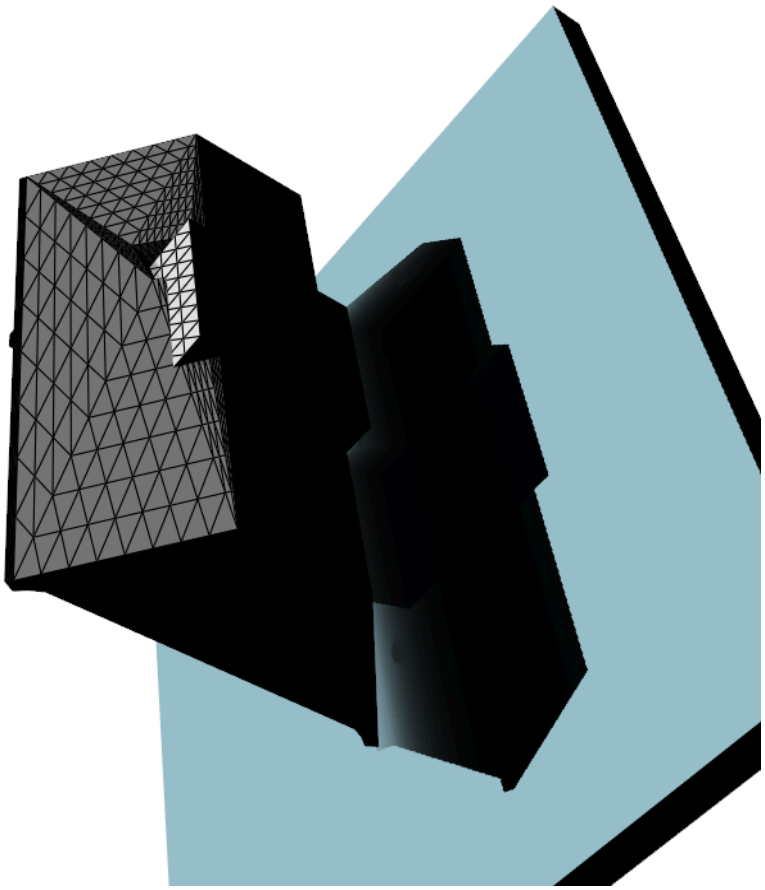
Therefore I computed ray tracing directly. We can see that from a pixel array, we can trace rays to simulate the sun. We can compute data about the rays that make contact with the surface, in order to find the force and torque on the satellite.

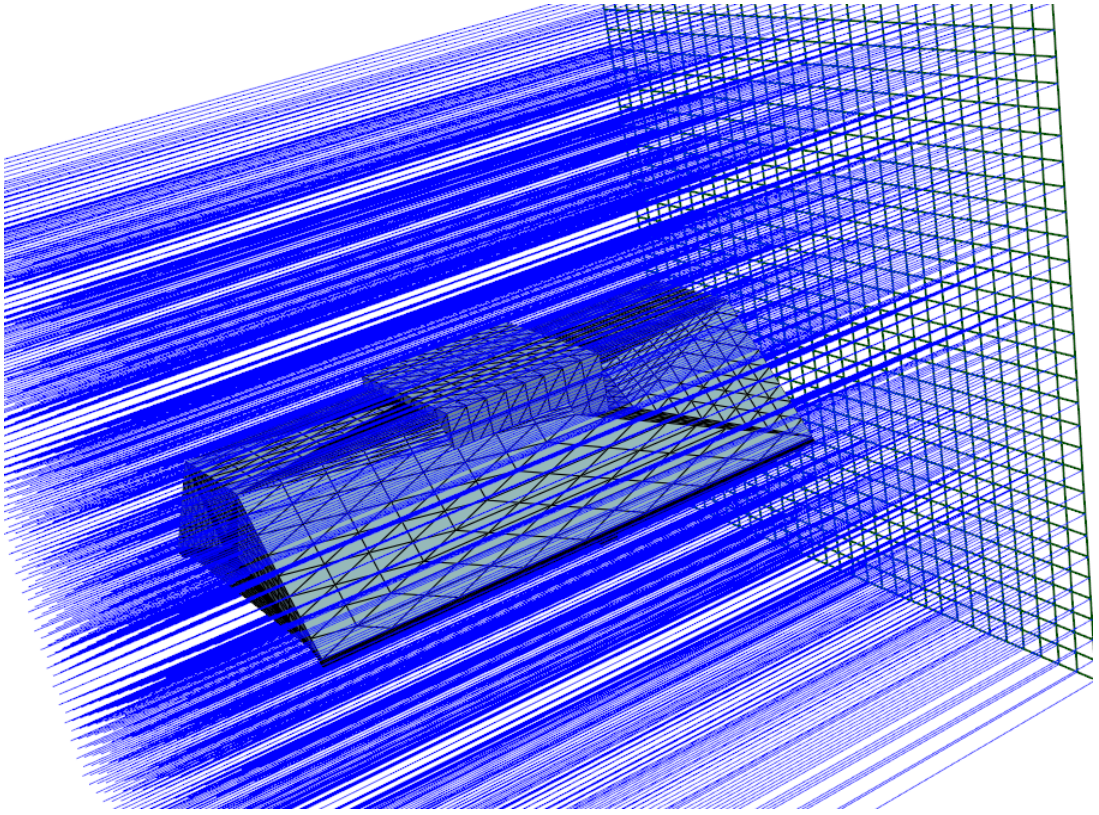


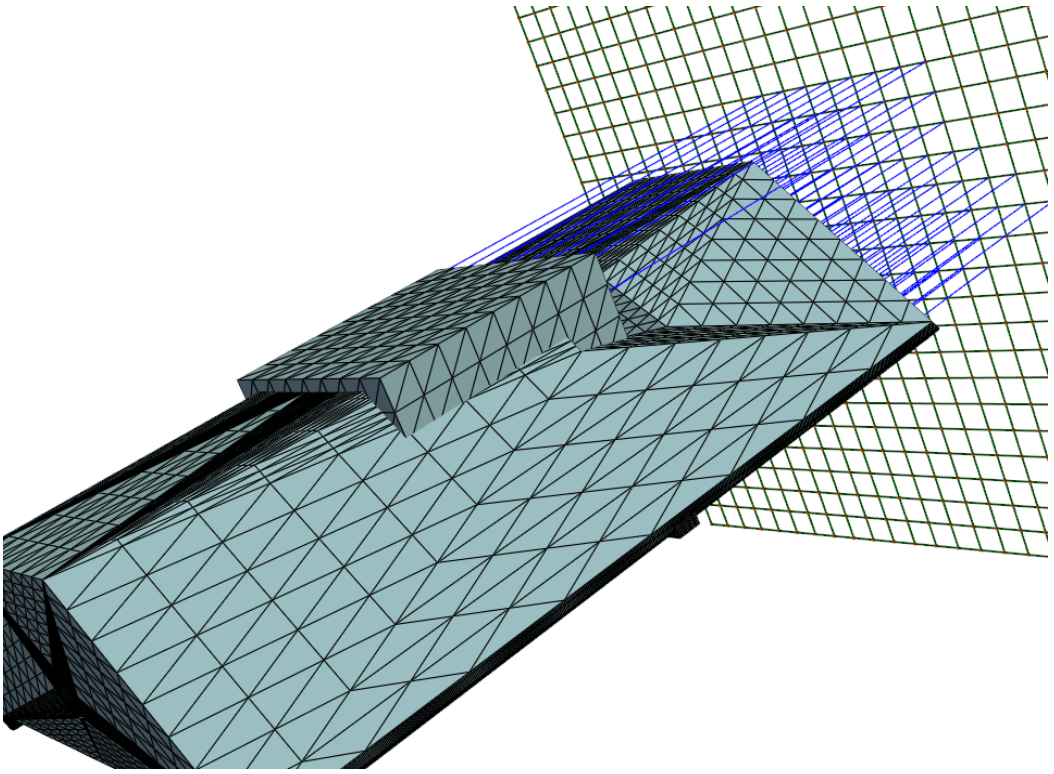

Y
X
Z

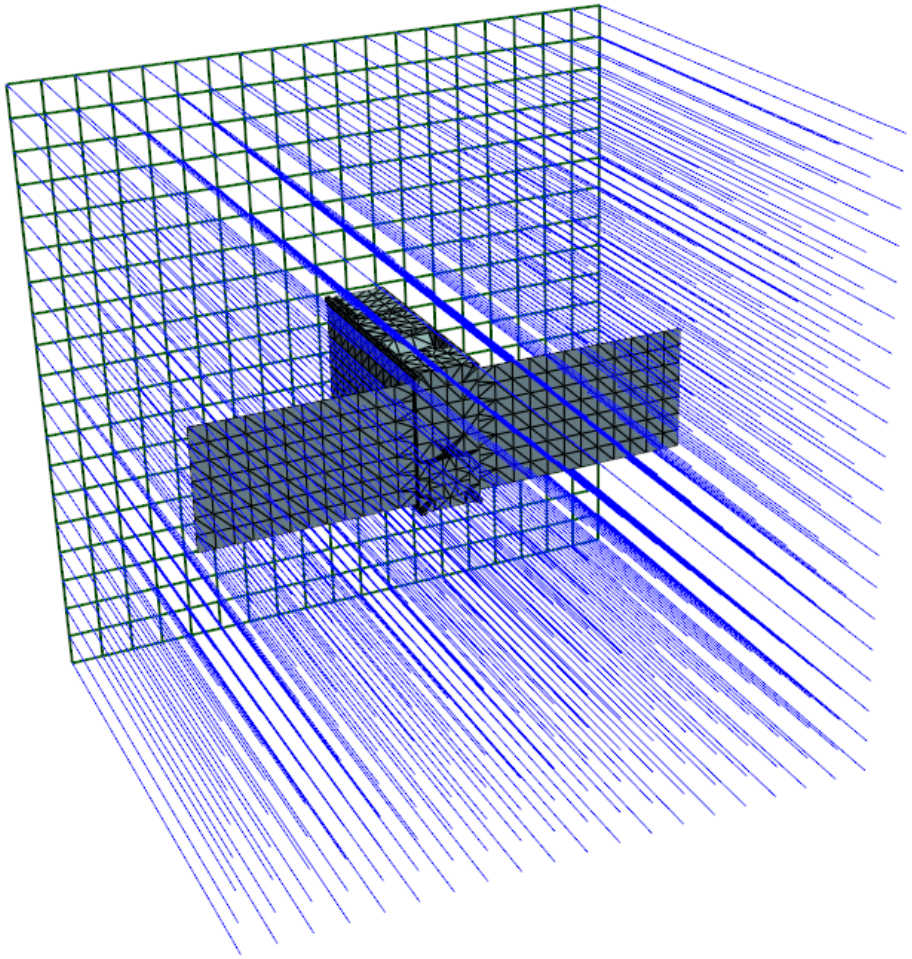


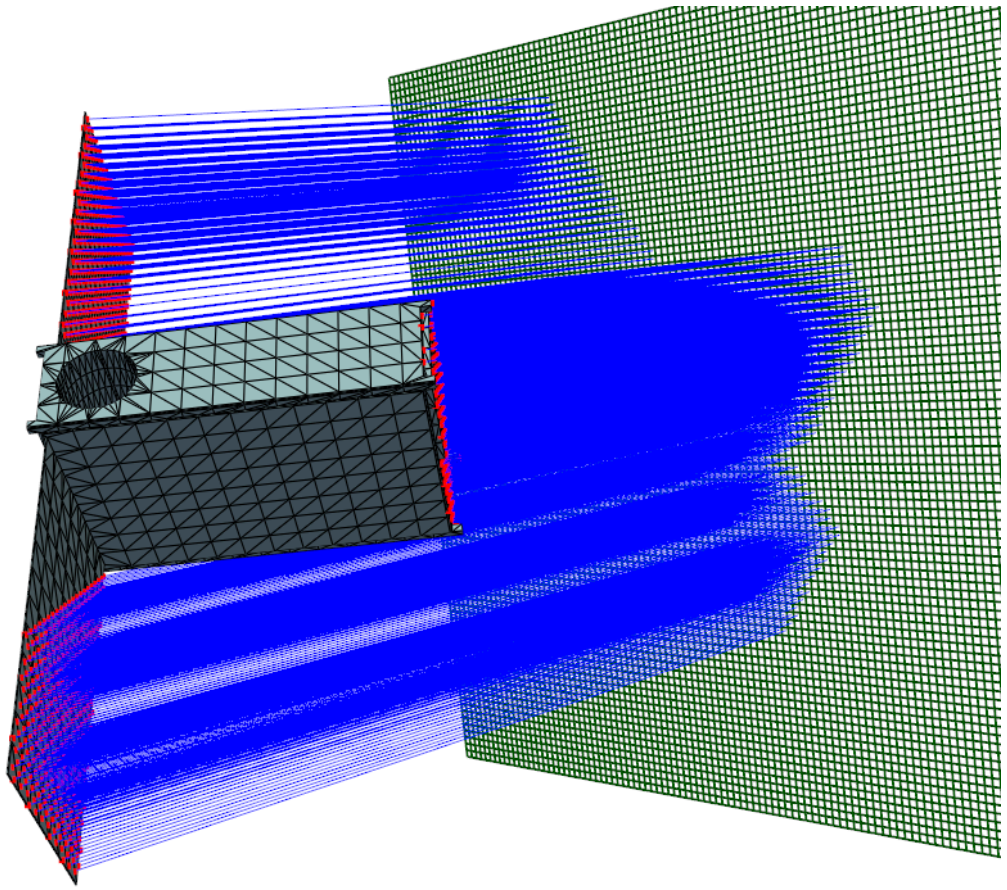

Y
Z
X







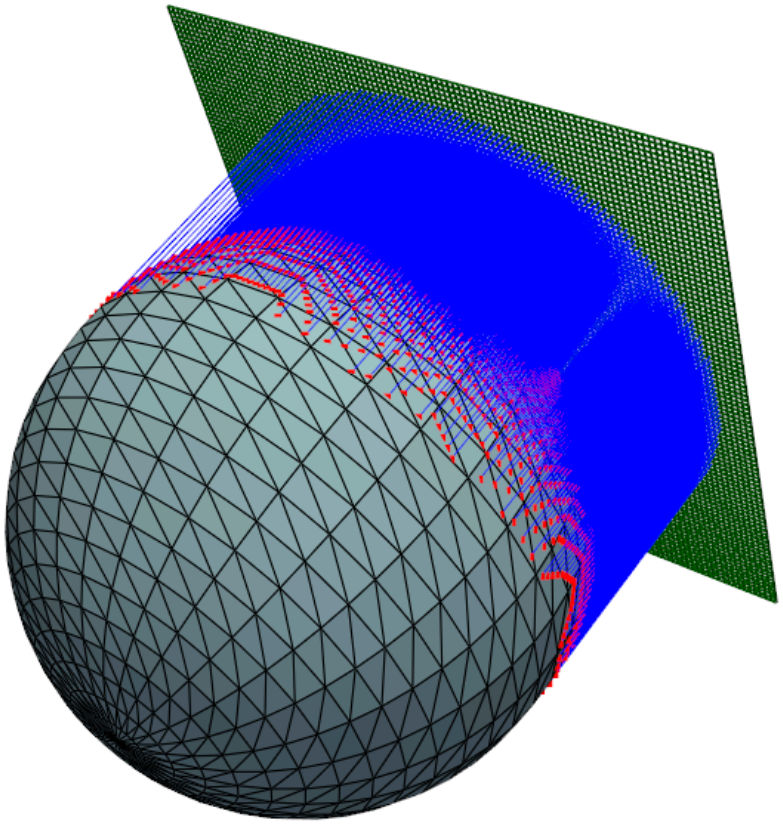




```
In [15]: A
Out[15]:
array([[[1.        , 1.        , 1.        ],
        [0.27505094, 0.24526394, 0.2579056 ],
        [0.12420079, 0.11083056, 0.11541523]],

       [[3.63569012, 4.07724019, 3.87738775],
        [1.        , 1.        , 1.        ],
        [0.45155558, 0.45188279, 0.44750958]],

       [[8.05147874, 9.02278257, 8.66436805],
        [2.21456683, 2.21296322, 2.23458901],
        [1.        , 1.        , 1.        ]]])

In [16]:
```

After completing ray tracing and optimisation, I computed the force and torque for each pixel that hits the satellite surface. Then we need to compare the accuracy of different resolutions to find the optimal pixel resolution for our model, also considering the time it takes to calculate.

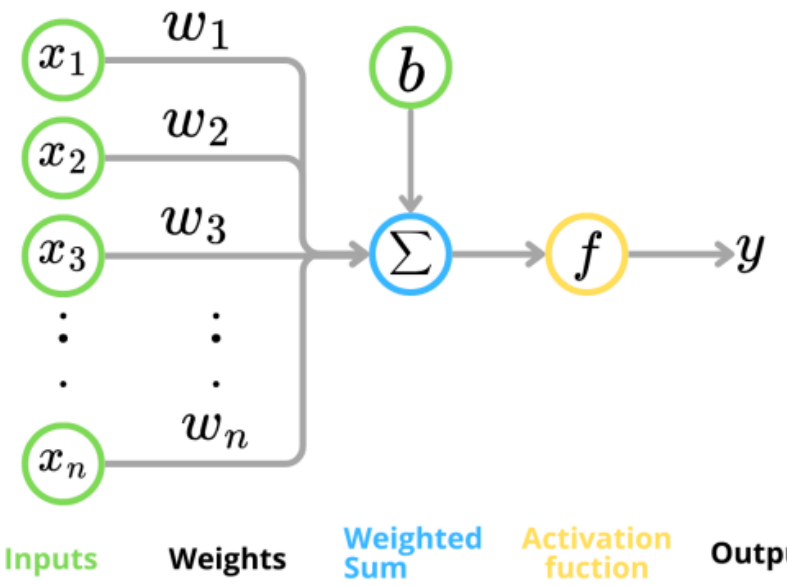
I check if the model works by computing the same method of a sphere and comparing the error between our estimated force and the ‘perfect force’ .

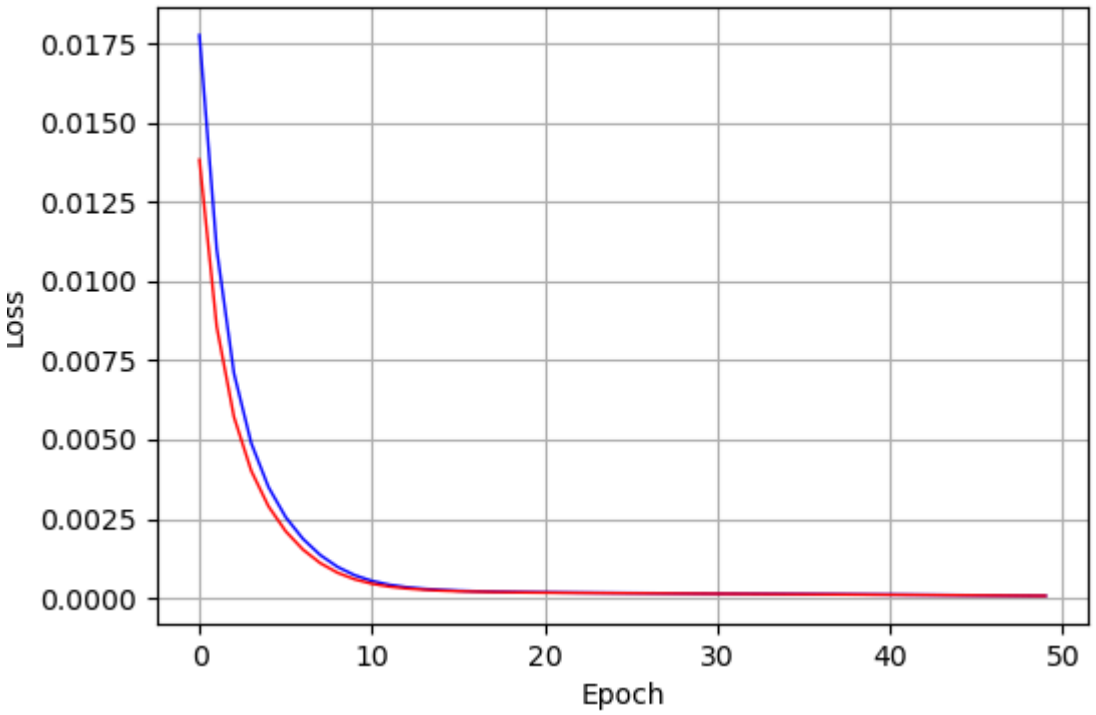


Results from the spherical model: Ran at 2000x2000 and real drag force of [0.00000000e+00 0.00000000e+00 -9.55672485e-07], we get an estimated drag force of [3.09147064e-13 2.62249789e-14 -9.44414648e-07] (error of 0.011780016008091232)

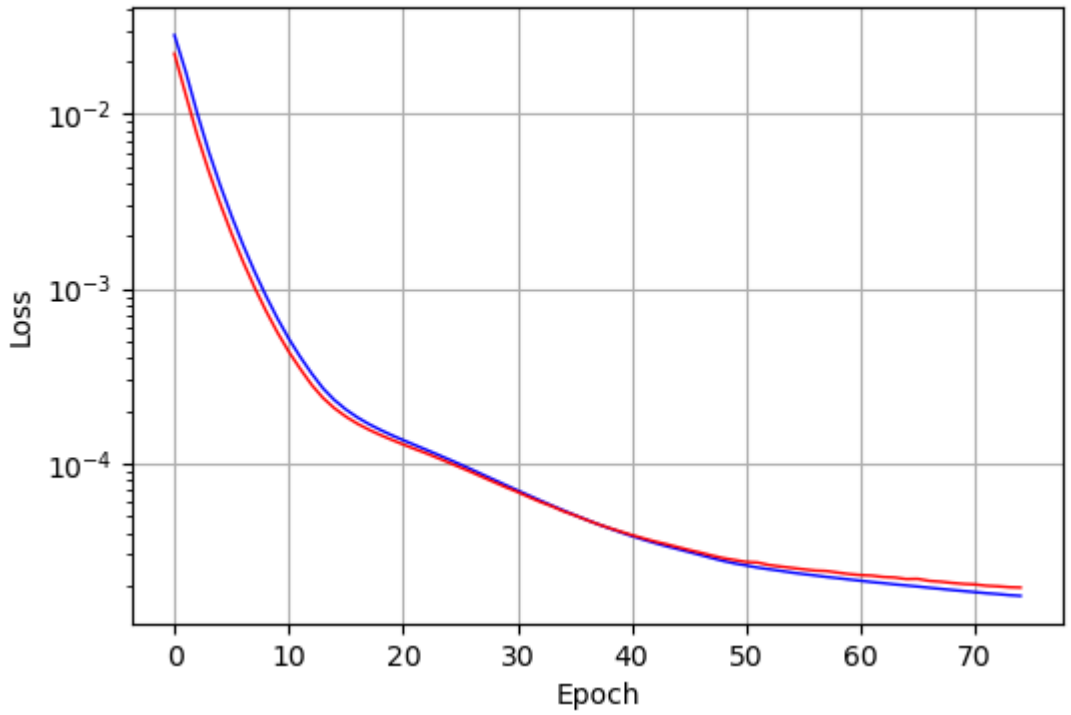
We have run the simulation for 20,000 vectors at a resolution of 2000 x 2000. This data will be used to train the neural network.

We need to find the best activation function for the neural network (we don't want ones which work with categorical data). So we choose an output activation function of Tanh because this returns continuous values between 1 and -1.

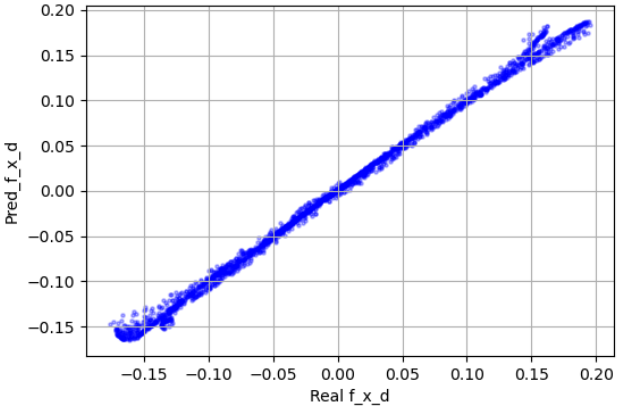




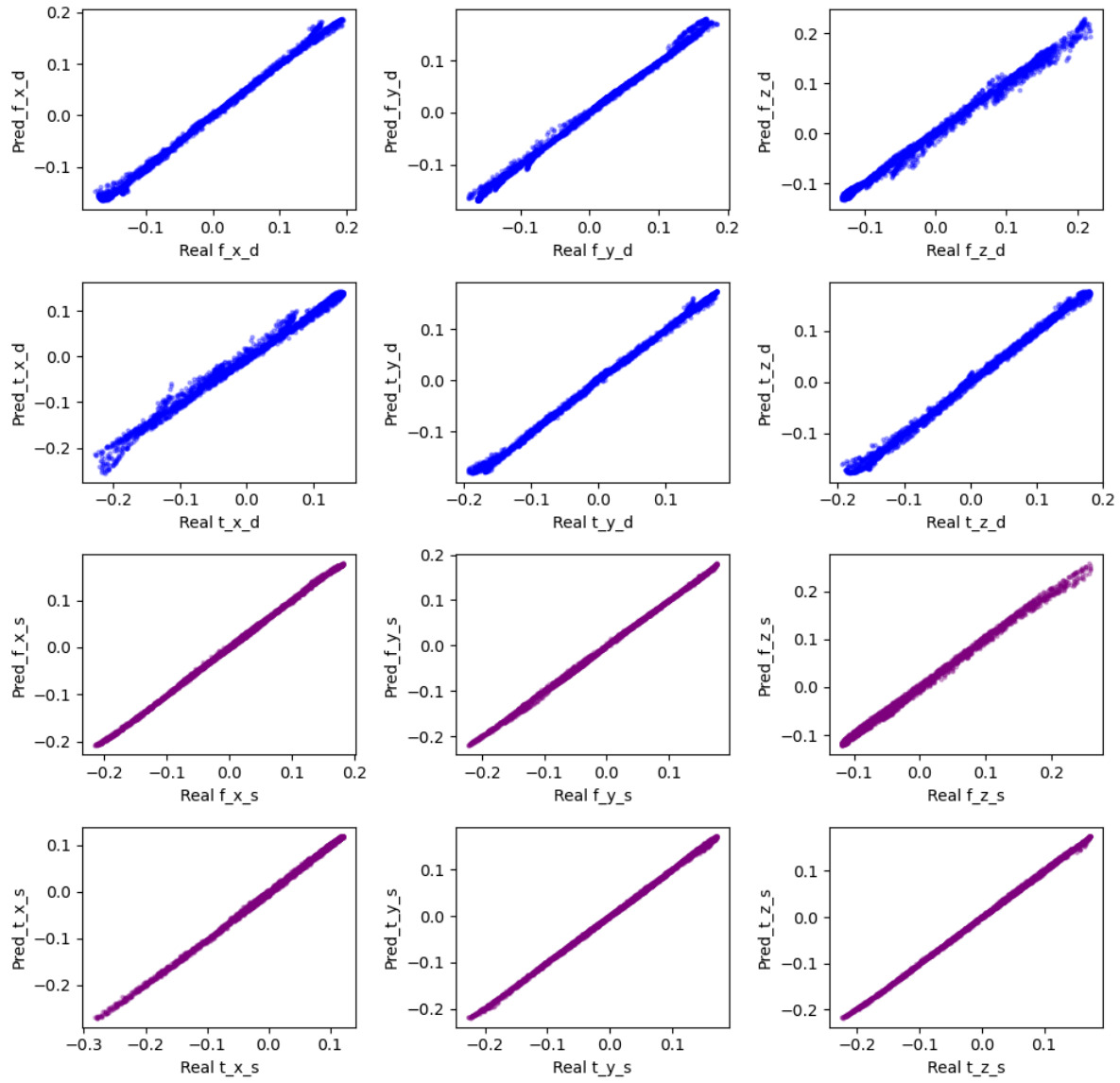
This shows how quickly the neural network is trained over 50 epochs (iterations). This is with batch processing. Blue in the training data, red is the validating data. Altered neurons, learning rate and epoch to avoid overfitting.



Now we use the test data to see if the model is accurate.

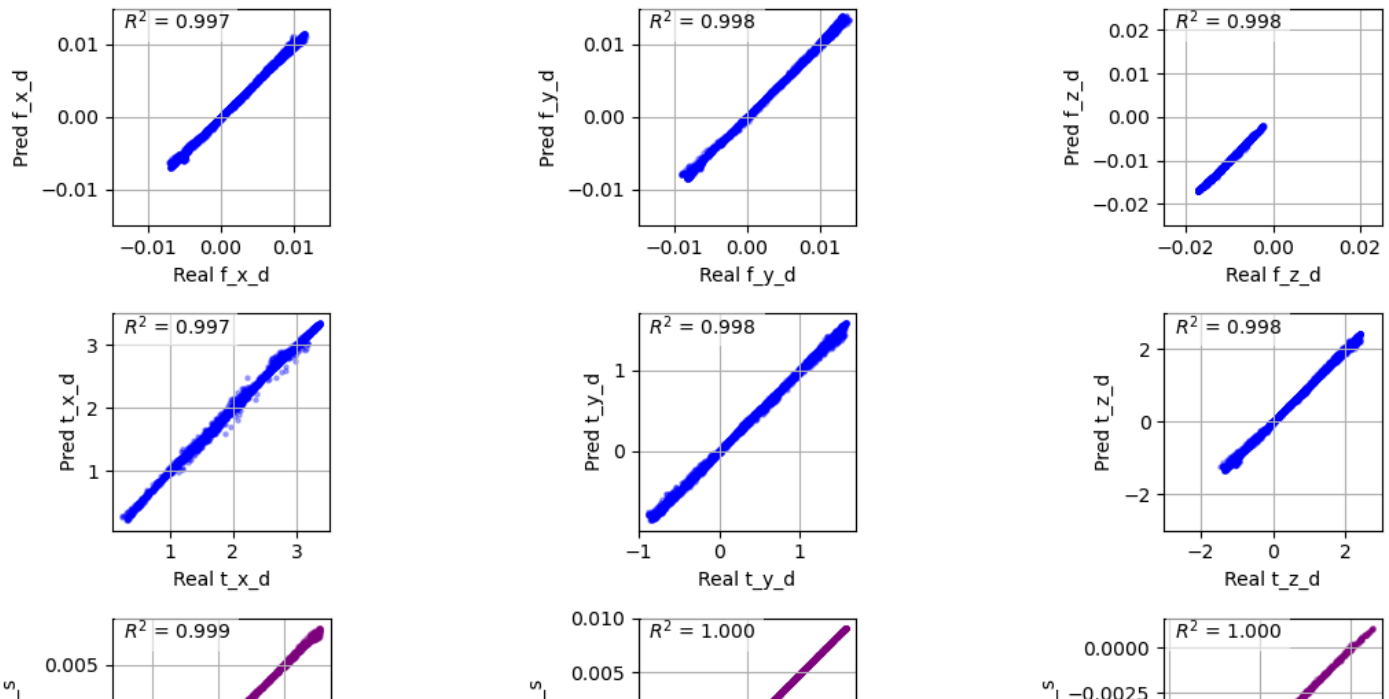


Plotting the x force component for drag against what the neural network predicts and what the actual force is (from the test inputs) shows a strong positive correlation, hence the network is working effectively.

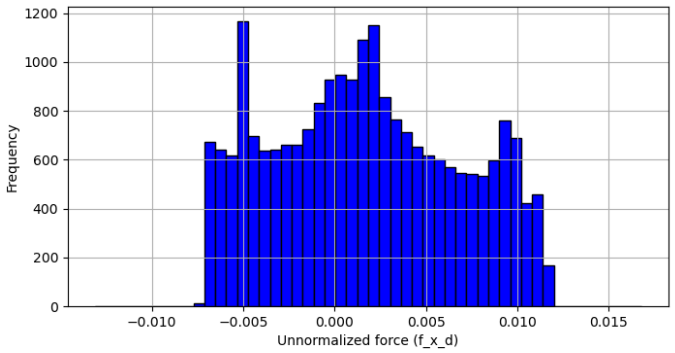


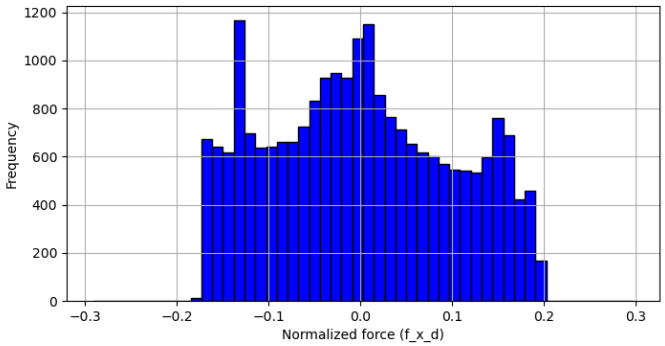
Note that forces and torques have been scaled to factor out constants, and z normalised to range between -1 and 1.

We reverse the z normalising to get,

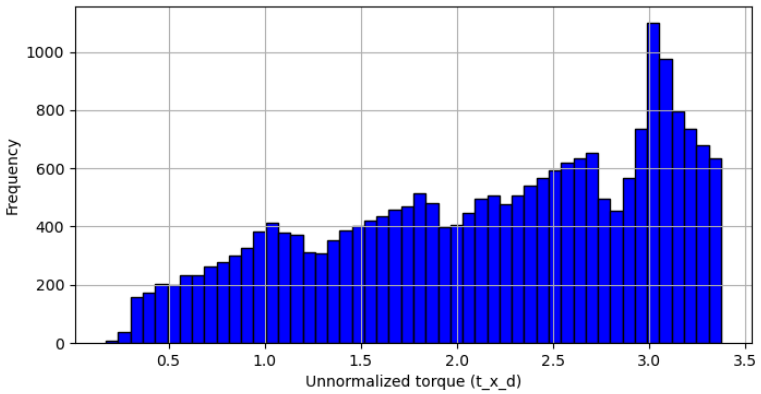


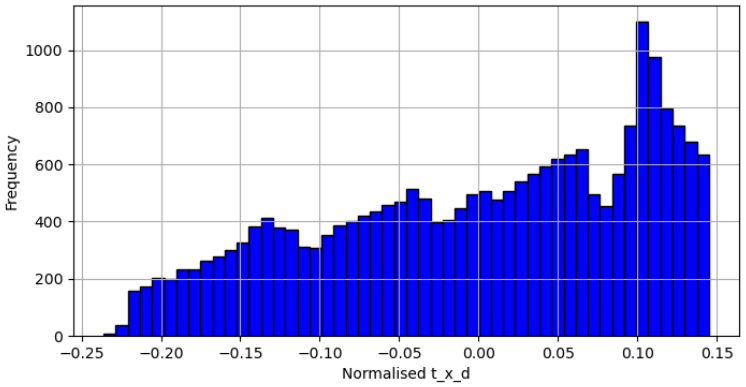
Now we can look at why we normalise all our forces and torques. Here is the data for f_x_d (factored out constants) unnormalised.





This scale is much more appropriate for the neural network as it will read very small values as 0 and by using the activation function Tanh, it will only output values between -1 and 1. But note that z-normalisation does not work very well if data is heavily skewed or contains lots of outliers.





I am trying to fix the sphere model (z coord out by x100) There was no problem with units so I looked at the locating cells method. Initially I had the code set to it filtering out multiple hits on a cell so the pixel area and force we calculated once for each cell. This is a problem because the total estimated area (cell area we calculate) will not be close to the real total area (of the cross section). So I changed my code to account for multiple hits on one cell. This made the total estimated area much more similar to the real area. However, there were still issues with the final force comparison as the x and y force components became a lot larger when they should be close to 0 , despite the z coord being more accurate. Then I tried using the actual area of the cells instead of using the area of the pixels to calculate the projected area. As a result, the total estimated area was the most accurate so far.

Elapsed time: 26.56 seconds to run the neural network before overfitting.