Modelling the Geometric Configuration of a Spacecraft in Dyreqt

Justin P. Coleman*, Bradford E. Robertson†, and Dimitri N. Mavris‡
Georgia Institute of Technology, Atlanta, GA, 30332, USA

The Dyreqt framework is a Python tool for the conceptual design of spacecraft. It allows a user to combine custom models of spacecraft subsystems with a mission in order to rapidly size a vehicle and perform trade studies. However, Dyreqt currently has no built-in method for representing the geometric configuration of a spacecraft, limiting its ability to consider this in analysis. This paper introduces a framework that allows for the geometric modelling of an arbitrary spacecraft within Dyreqt. The framework utilizes the methods of primitive instancing and constructive solid geometry to map each Dyreqt subelement to a physical body. This representation can then be passed to a set of models which calculate geometric properties, allowing new information to be computed within Dyreqt, including bounding dimensions, moments of inertia, and projected area. These in turn enable Dyreqt to utilize higher fidelity subsystem models, to include geometric constraints in optimization, and to produce three-dimensional visuals of its results. These capabilities were demonstrated by using the framework to compare different potential configurations for a lunar cargo lander.

I. Introduction

Dyreqt is a framework for the conceptual design of space systems based on OpenMDAO in Python. It was originally developed by Georgia Tech's Aerospace Systems Design Lab and now is co-maintained by ASDL and the NASA MSFC Advanced Concepts Office [1]. This tool allows for the rapid sizing and comparison of spacecraft concepts by connecting various models of spacecraft subsystems. These subsystems are represented as a collection of Dyreqt subelements making up a vehicle (or element). Each of these contains a model for some spacecraft component (e.g. solar panels, propellant tanks, engines, etc.). The spacecraft's mission is represented as a set of events. When the code is run, the subelements are sized to the needs of the mission. Each subelement and event can be custom tailored to a problem allowing a great deal of flexibility. As a result, Dyreqt is capable of modelling a wide variety of different space systems including landers, Mars exploration vehicles, and surface systems [2-4].

Despite this flexibility, Dyreqt is still limited in how it represents the geometric configuration of a space system. While individual subelements can include their own geometry in their calculations, there is no framework for modelling the overall arrangement of subelements in the vehicle. As a result, there is no way to compute the overall geometric properties of the vehicle such as its dimensions, moment of inertia, or volume. This hampers analysis as these values provide important constraints on the vehicle and are crucial to designing certain subsystems. In addition, the lack of a geometric representation limits Dyreqt to providing numerical outputs with no built-in way to visualize the spacecraft it models. This makes it harder to communicate Dyreqt results.

Some past Dyreqt studies have created their own systems for geometric modelling [2]. However, they have been limited in scope; only able to model a limited number of geometries and provide a small number of geometric properties. The goal of this project was to create a more general geometric framework which can model any configuration of components, and which can provide that model in a form that can be utilized by many types of geometric calculations. This framework was integrated with current Dyreqt subelements, allowing them to use geometric information for increased fidelity. Additionally, this system has enabled Dyreqt to automatically generate visual models of its results.

* Graduate Researcher, School of Aerospace Engineering, AIAA Member
† Senior Research Engineer, School of Aerospace Engineering, Senior AIAA Member
‡ Regents Professor, School of Aerospace Engineering, AIAA Fellow

II. Background

A. **Applications of Geometric Modelling in Spacecraft Design**

This section presents a review of how geometric information is used in different areas of spacecraft design and modelling. This served to guide the development of capabilities for the geometric framework.

1. Attitude Control

One major subsystem which relies on knowledge of spacecraft geometry is attitude control. This is because the agility performance of a spacecraft depends on its moment of inertia. For a given requirement on slew rate and time to accelerate, the spacecraft's moment of inertia determines the required torque (and possibly angular momentum capacity) of the attitude control actuators [5-6]. This in turn determines the mass and power requirements of the actuators and influences the choice of what types and how many actuators should be used. Furthermore, certain types of attitude devices, such as thrusters or solar reflectors, have performance dependent on their position on the spacecraft. Without knowing the size or shape of the spacecraft, these can not be designed in any detail.

Beyond maneuvers, geometric information is also needed to estimate the magnitude of disturbances on a spacecraft's attitude. In the case of gravity gradient torques, this depends on the differences in the principal moments of inertia of a spacecraft and how these axes are aligned with the orbit. For disturbances from solar radiation pressure or atmospheric drag, the necessary parameter is the projected area of the spacecraft and how its geometric centroid is positioned relative to its center of mass [6]. Knowing these disturbance torques is important for sizing control systems, particularly for finding the momentum storage requirements of reaction wheels or the size of magnetic torque rods [5, 7]. These estimates of solar and atmospheric forces can also be used to more accurately model spacecraft trajectories and to consider coupled orbit-attitude propagation [8].

2. Structural Modelling

Another application of geometric modeling is the estimation of spacecraft structural mass. An example of this is presented by Heineman [9]. In this paper, a regression was used to find structural mass as a function of design envelope volume: the bounding volume around the spacecraft. Through such a regression, it is possible to obtain a more accurate estimate of the structural mass of a spacecraft than the more typical method of a simple mass fraction. This technique has been applied in Dyreqt in past ASL projects where it was used to find the structural mass of a lunar cargo lander. By incorporating this method, it was possible to compare different lander configurations on the basis of their structural mass. To enable, this sort of analysis, the framework will need to be able to calculate the envelope volume around a spacecraft.

3. Bounding Dimensions

An important constraint in spacecraft design is the size of the available launch vehicle fairing. This places constraints on the overall radius and length of a vehicle at launch. For large vehicles, these constraints can be a major driver of design in terms of how subsystems are arranged and what types of missions are possible. This can have knock-on-effects on other systems such as limiting the size of cargo that can be carried on a lander. For large space campaigns, these constraints can dictate what architectures are feasible and what technologies are necessary to achieve mission goals [10]. To address this, the geometry framework will need to compute the bounding cylinder around a spacecraft. It will also need to account for such deployable components, such as solar panels, which can have multiple configurations.

B. **Representing Three-Dimensional Objects**

Past efforts to add geometry to Dyreqt have relied on a layered representation of the spacecraft [2]. Each layer is a particular, mostly 2-dimensional arrangement of major components such as propellant tanks, engines, and payloads. By stacking different layers, it is possible to represent an overall vehicle. Geometric properties are found by deriving an expression for the values of each layer and then combining the results (e.g., adding the height of each layer for overall length or finding the maximum radius of the layers). This technique is quite simple to implement and use but has some limitations. First, the range of geometries that can be considered is limited by the fact that layers can generally only be stacked in one direction with no complex intermeshing between them. Second, deriving expressions for each layer becomes complicated with larger layers and does not allow for more complex calculations on the whole spacecraft. Finally, this representation cannot be easily be incorporated into other models or tools as there is no way to convert it into a standard CAD format.

To achieve the goals of this project, a more general form of geometric representation is needed. There are numerous schemes available of which two are particularly useful here. The first is the method of primitive instancing.

This represents different shapes in terms of a set of primitive classes (e.g. a cube, cylinder, etc.). Each dimension of the shapes can be set to produce an instance of the primitive with the desired properties. A general body can be represented by combining a set of primitives. In general, this technique of combining simple shapes is called constructive solid geometry (CSG) [11]. Each shape in the arrangement is described by three basic parameters: its dimensions, a translation representing its position in space, and a rotation describing its orientation. Because each primitive only requires a small number of parameters, this method provides an efficient way to represent the overall body, and it is commonly used in computer vision applications [12]. Indeed, it has recently been applied to modelling the shape of a spacecraft for visual guidance during rendezvous [13]. Since each shape can be mapped to a physical object, this method also provides an efficient way for calculating certain physical properties such as moment of inertia.*

The second method of 3D modelling to consider is the polygon mesh method, a form of boundary representation. This method models the surface of an object in terms of a set of vertices and faces between those vertices. This form of representation is useful for generating computer graphics and for performing certain calculations which are difficult to do directly in terms of the primitives. Crucially, it is straightforward to convert from a primitive model to a mesh model allowing both to be utilized in tandem [11]. The type of mesh model used here is the STL format which utilizes triangular faces. This format is the primary modelling standard used in 3D printing and thus enjoys a good deal of programming support [14]. This format has also been used in previous applications of spacecraft geometric modelling such as in reference [8].

C. Coordinate Transformations

The position and orientation of a rigid body (such as a geometric primitive in CSG) can be described by a coordinate transformation between a global frame and the body frame [11]. This transformation can be broken into a translation and a rotation. If a set of points have known locations in both frames, these two transformations can be found through the following optimization problem:

$$\min_{\mathbf{R},\mathbf{t}} \frac{1}{N}\sum_{i=1}^{N} \|q_i - (\boldsymbol{R}p_i + \boldsymbol{t})\|^2 \tag{1}$$

where **R** is a rotation matrix, **t** is a translation vector, and p and q are the points expressed in each frame. There is a unique solution for this problem when N=3 [15]. The rotation matrix alone can be found by aligning two vectors expressed in the two frames (e.g., the vectors from one of the points to the other two) [16]. This is known as Wahba's problem and can be expressed as:

$$\min_{\mathbf{R}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{q}_i - (\boldsymbol{R}\boldsymbol{p}_i)\|^2 \tag{2}$$

Once the rotation is known, the translation can be found through the following equation:

$$\boldsymbol{t} = p_i - \boldsymbol{R}q_i \tag{3}$$

where the rotation matrix is used to align the position vector in the second frame with the axes of the first frame [16].

Once the orientation of a body is known, it can be used to relate properties in the body frame to the global frame. Positions are transformed simply using the inverse of the expressions above:

$$\boldsymbol{r} = \boldsymbol{R}\boldsymbol{r}_{\boldsymbol{b}} + \boldsymbol{t} \tag{4}$$

Vectors representing directions need only be rotated without the translation. Second order tensors (such as the inertia matrix) can be rotated into axes parallel to the global frame using:

$$\boldsymbol{I}' = \boldsymbol{R}\boldsymbol{I}_{\boldsymbol{b}}\boldsymbol{R}^T \tag{5}$$

From here, the parallel axis theorem can be used to find the moment of inertia about the global origin (J) using [17]:

$$J_{ij} = I'_{ij} + M\left(\delta_{ij}\boldsymbol{t}^2 - t_i t_j\right) \tag{6}$$

III. Approach

The basic modelling approach for this framework is to represent each component of the spacecraft individually in the geometric definition. Compared to the layer method used previously, this allows for much more flexibility in the types of geometries that can be created and in the way the geometry can be mapped to Dyreqt. It also produces a geometric representation which can more easily be used in a variety of calculations. The primary representation will

* These techniques are commonly used in video games as they allow physical properties to be calculated much more efficiently than using the visual model of a game object (see: https://www.gamedev.net/tutorials/programming/math-and-physics/capsule-inertia-tensor-r3856/).

be implemented using the method of primitive instancing and CSG. Each primitive instance is called a shape. As in reference 11, each shape will be described by its dimensions, location, and orientation in space. In addition, each shape will possess a mass which is involved in calculations such as moment of inertia. This mass is obtained by mapping the geometric component to a Dyreqt subelement (or group of subelements). Additional information may be provided by Dyreqt if it is modelled by the subelement (e.g. dimensions are included in the existing propellant tank model and surface area is computed by the solar panel and radiator models).

The challenge with this approach is determining the location and orientation of each component. Since the spacecraft sizing is combined with the geometric modelling, the physical sizes of each component are not known in advance. This means it is generally not possible to prespecify the location or orientation of shapes in the model since these parameters depend on the sizes of the other components. As a result, the geometric definition set by the user cannot be specified in these terms as it would be in other applications of these modelling techniques. Instead, the framework uses a definition of the shapes' location relative to each other. This is done by specifying a hierarchy of physical connections between shapes starting with a single shape at the origin.

Mathematically, these connections are represented using a set of connection points associated with each shape. These are points on the surface of a shape which compute their coordinates in both the shape's frame and the global frame. The location of each shape is specified by setting one of its connection points to coincide with a point on an upstream shape. To find orientation, each connection point has an associated pair of vectors: one normal to the surface and one tangent. Again, these are defined in each frame. A shape's orientation is the coordinate rotation which aligns its vectors with those of the upstream shape.

The process to compute location and orientation of a shape is as follows (figure 1): First, the shape takes as input the location and vectors of the upstream connection point (in the global frame). Next, the shape computes the rotation necessary to align its tangent vector and the negative of its normal vector (in its own coordinates) with the upstream vectors. This is an application of Wahba's problem and gives the rotation necessary for the shapes to be tangent at their connection. Using this rotation, the shape can then transform its internal location vector for the connection point into the global frame. Subtracting this from the vector for the upstream point will produce the translation locating the shape's origin in the global frame. Lastly, the shape will find the global position of each of its other connection points and provide them to downstream shapes. This process repeats for each shape, building out the model like a tree until the location and orientation of every shape is known.

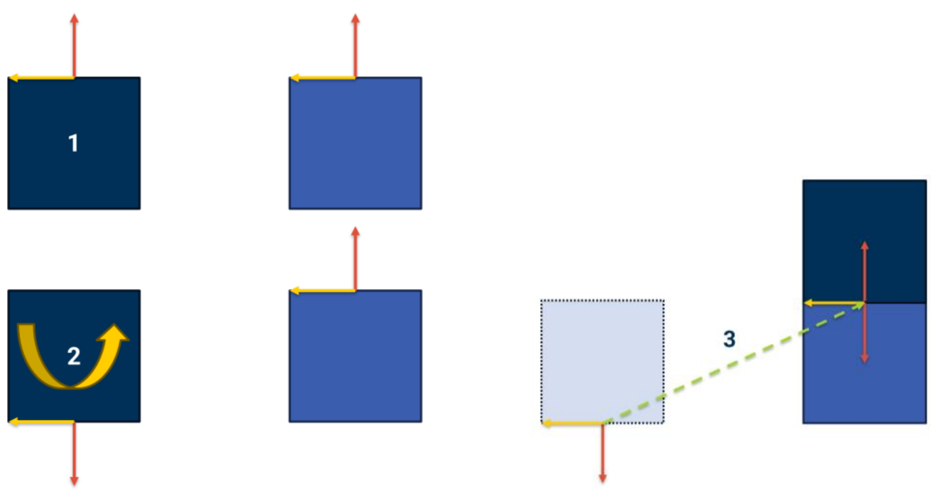


Fig. 1 From 1-3, the process of computing the location of a shape.

While the entire geometry can be defined in one large tree, the framework also supports grouping shapes together. These are simply called shape groups. This is useful for organization since complex subsystems can be placed into their own groups. It is also useful when working with multiple spacecraft docked together, allowing geometric calculations to be performed on both the overall geometry and on each individual spacecraft. Within the framework, each group functions as its own tree, and its constituent shapes find their location relative to the group's coordinate system. From here each group can be added to a higher-level group where it will function as a single complex shape. For this purpose, the connection points of the shapes within the group become connection points for the group as a whole (now transformed twice from the shape's coordinates to the group's coordinates and then to the parent's coordinates).

One final level of customization of the model is the use of alternate configurations. These mark certain shapes within a group to be excluded from geometric calculations. The excluded shapes will still calculate their location and orientation as normal, but they will not contribute to any other geometric properties when further calculations are performed. This is very useful for modelling components with multiple states. For example, a solar array could be represented with two shapes: one for the deployed state and one for the stowed state. Configurations allow for both shapes to be in the same group (so that the rest of the spacecraft need not be duplicated) but for calculations to consider only one of the shapes at a time (and thus one of the two states).

A. **Code Implementation and Problem Structure**

The framework was implemented in Python using OpenMDAO: a library for multidisciplinary design analysis and optimization which Dyreqt is built on [1]. This has the advantage that it simplifies the integration of the geometric tool with Dyreqt. However, beyond integration, OpenMDAO provides an excellent basis for implementing the modelling method described above. Each shape can be made into an OpenMDAO component which will handle both the computation of the shape's internal properties and the sharing of information with other shapes to form the complete model.

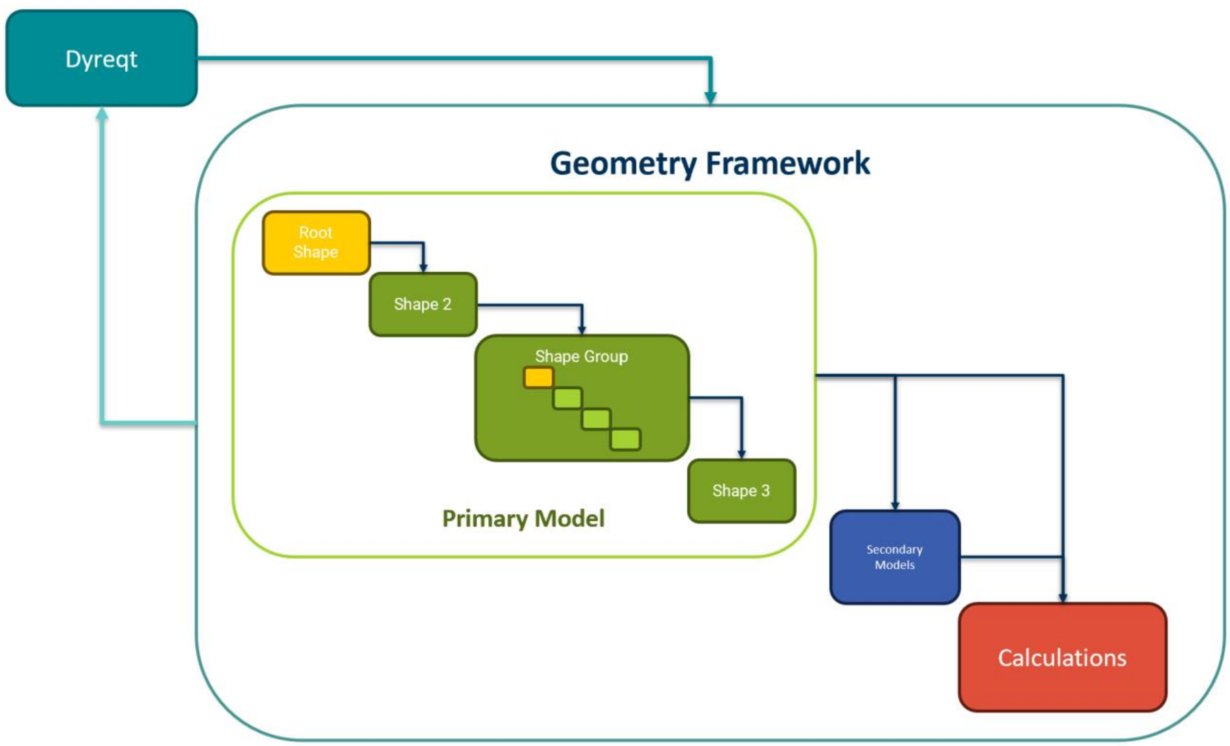


Fig. 2 Problem N^2 Diagram

Within the OpenMDAO problem, the geometry model will form a separate block acting in a feedback loop with Dyreqt (figure 2). This is similar to other Dyreqt addons developed at ASDL such as cost estimation with NASA's Project Cost Estimating Capability tool (PCEC). To build the problem, a recursive function is used which reads the definition of each shape group and adds a component for each shape and subgroup. In order to ensure there are no unnecessary feedback loops within each group's computation, the shapes are sorted based on their connections before being added to the problem. This ensures that each shape will only calculate its position after the upstream shape it is connected to. This sorting process also serves as a way to check for errors in the geometry definition. If a group contains a circuit of connections, no or multiple origin shapes, or other invalid connection setups, it will raise an error to alert the user. After the primary model components, additional components can be added to the problem to create secondary models and to perform calculations. The major classes of components are described in the following sections.

B. **Shapes**

Each shape in a geometry model is represented by an OpenMDAO explicit component. The computation for each shape occurs in three stages. First the shape will determine its internal properties. These methods are set for each specific shape class (i.e. a cube, cylinder, etc.) and determine the mass (either as a user input or from Dyreqt), the dimensions of the shape, the location of the shape's connection points in its internal coordinates, and the mass properties of the shape (center of mass and moment of inertia) again in internal coordinates. The second step is to find the location and orientation of the shape using the process described above. These methods are stored in the base shape class and make use of the SciPy Rotation library. The locations of the shape's connection points in the group frame are output from this step. The final stage, also done by the base shape class, is to determine the contributions of the shape to the mass properties of its parent group. For center of mass, this is the first moment of the shape's mass in the group coordinates. For moment of inertia, this is the shape's moment of inertia about the group's origin.

C. **Shape Groups**

Shape groups are represented with subsystem groups in OpenMDAO. These contain the components of their constituent shapes as well as any subgroups. Similar to subelement groups in Dyreqt, each shape group is automatically given a utility shape which handles the calculation of the overall group properties [1]. This sums the first and second mass moment contributions of each shape in the group and uses them to find the group's center of mass and its moment of inertia about that center of mass. The utility shape also takes as input the connection points of each shape in the group and treats them as its own. From here, the computation follows that of the base shape class, allowing the group to be added into a higher-level parent group.

D. **Secondary Models**

In some cases, the primary geometric model (using CSG) is unsuited for a particular application. This can occur if a calculation is difficult to perform directly in terms of the geometric primitives. For example, when finding bounding dimensions, it is much easier to work with a set of points than directly in terms of a set of bodies. This problem also occurs when using other tools (e.g., for visualization) which require the geometry to be represented in a particular structure or file format. To address this, the framework allows for the creation of secondary models. In the code, these take as input the dimension, location, and orientation parameters of each shape and add them to a new model expressed in another form. Depending on the application, this can be done in-the-loop of the problem or as a post-process. The current implementation of this feature is using STL mesh models generated with the Python library: NumPy-STL. This format is useful for calculations involving bounding dimensions and cross-sections. It can also be used to produce visuals using Matplotlib as shown in figure 3.

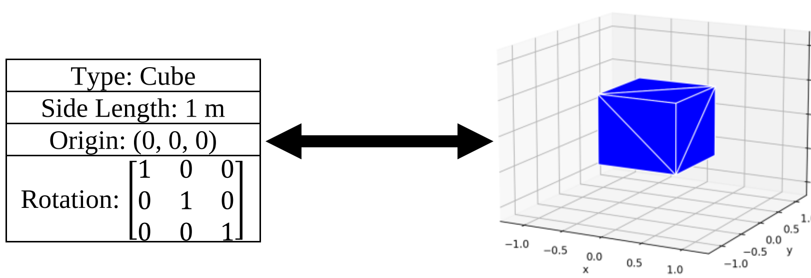


Fig. 3 Relationship between primary and secondary models.

E. **Calculations**

Once the model or models of a spacecraft have been computed, the last step is to perform calculations of geometric properties. These can take two forms: First are additive properties. These are simply the sum of the properties of each shape in a group (such as the mass properties already discussed). As these properties involve each shape individually, they are calculated within the shapes themselves. The sums are computed by the shape groups (as discussed above) for the main spacecraft configuration. They may also be computed at the end by a separate component for alternate configurations.

The second group are properties which require the overall geometric model. These relate to the spacecraft as a whole and cannot be determined by the individual shapes. They are added as separate components at the end. These components can either utilize the primary geometric representation, taking as input the dimensions, locations, and orientations of each shape, or a secondary model. In this case, the secondary model object will be passed as an input. The following sections details the calculations which have been implemented so far:

1. Mass Properties

This group of calculations determines the mass, center of mass location, and inertia tensor for each shape group. This is an additive property and is handled by the shapes themselves using the method above. By default, only a single value of each output is provided, corresponding to the maximum mass of a Dyreqt-linked shape. This results in the maximum moment of inertia which forms the limiting case for sizing attitude control systems [5, 6]. However, in certain situations, such as for detailed modelling of attitude maneuvers throughout a mission, it is desirable to know how mass properties change over time. For shapes and simple groups, this is straightforward and only requires repeating the calculation for each mass value in the linked subelement's mass history. However, it is more complicated when using subgroups corresponding to different Dyreqt elements (e.g., when having multiple spacecraft docked together). In this case, the elements may not spend all their time docked together and thus may have mass histories corresponding to different time steps. To address this, each shape group is assigned to a Dyreqt element. Whenever a group is assigned to a different element than its parent, it will compare the missions of the two elements to produce a mapping between their time states. When the group utility shape then produces its contribution to the parent's mass properties, it will use this mapping to match its time steps to the parent's.

2. Bounding Dimensions

These calculations find the dimensions of a bounding shape around a geometry group. This information is important to determine if a spacecraft will fit in a given space such as the fairing of a launch vehicle. The calculation is performed by finding the minimum size shape which can enclose all the vertices of the STL secondary model. This can either be done as a cylinder (appropriate for launch vehicle constraints) or a box (more relevant for spaces like cargo holds).

In the box case, the bounding dimensions can be determined by finding the maximum and minimum coordinates of vertices along each coordinate axis. The difference between these gives the dimensions of the box needed to enclose the group. For the cylindrical case, the height can be found by the same method, taking the difference between the maximum and minimum coordinate value from the list of points along the axis of interest. Finding the radius can be done in two ways. If it is known that the axis of the spacecraft (or at least the axis of the bounding cylinder) will be centered on the origin, it is enough to find the point with the maximum 2D distance from the origin (i.e. distance in the plane perpendicular to the axis). On the other hand, for a general set of points, it is necessary to perform an optimization to find which center gives the smallest bounding radius. This is known as the bounding circle problem. It can be solved relatively efficiently using various algorithms but is still more complex than for a centered set of points [18]. This code uses an implementation of this problem from the Shapley Python package.

3. Volume

This is another additive property computed within the shapes and groups themselves. Each shape class has an analytic formula for its volume, and the group utility shapes sum these to determine the overall volume of the group. Along with the volume of the bounding shape components, this calculation enables the use of structural models based on spacecraft volume as in reference 9.

4. Projected Area

This component finds the area and centroid of the projected area of a spacecraft along an axis. This again makes use of the NumPy-STL secondary model. The axis of projection can be set as an option to the OpenMDAO component. The calculation works by first projecting each of the faces from the STL mesh onto the plane perpendicular to this axis. This results in a set of overlapping triangles. The Shapely Python package is then used to compute the union of these triangles which is the projected shape of the spacecraft. This is an application of CSG in two dimensions. Shapely is then able to find the area of this shape as well as the location of its centroid. The model returns these values as well as the projected distance between the centroid and the spacecraft center of mass. This information can be used by Dyreqt's models of attitude disturbances.

5. Distance

This component is used to find the distance between two points in the model. These can be connection points, a group's center of mass, or any other point with an accessible output. To give the correct value, the points' locations must be provided in the appropriate coordinate system (i.e. both points must be provided in the same group). The calculation returns both the vector between the points and the magnitude of the distance. An option is provided to also return the distance projected onto some plane. This can be used to determine values such as the moment arm of a thruster for rotation about a given axis.

IV. Examples

Two case studies were conducted to demonstrate the capabilities of the geometry framework. The first recreated a real spacecraft in the framework while the second utilized it in a design exercise.

A. Lunar Reconnaissance Orbiter (LRO)

In order to verify that the geometry framework is functioning correctly and to validate the accuracy of its moment of inertia calculations, the framework was used to create a model of the Lunar Reconnaissance Orbiter. This utilized a simplified layout of the spacecraft split into 6 parts: the main bus, two instrument clusters, two pieces of the solar array, and the high gain antenna assembly (HGA). Parameters for this model were obtained from references 19-21. The main bus of the spacecraft consisted of a box shape with dimensions 1.2 m x 1.2 m x 2.4 m and a dry mass of 826 kg. On one side were the two instrument clusters. Their masses were 37.1 kg and 23.7 kg, respectively. The solar array was located on the opposing face of the bus and has a total mass of 87 kg. A mass of 30 kg was allocated to the panels themselves, modelled as a thin rectangular plate, based on their area and a typical area density for solar arrays [6]. The remaining mass was allocated to a cylindrical mounting arm. The HGA was modelled as a 44.7 kg cylinder extending from the third face of the bus. These components were modeled both in their operational configuration and in a stowed launch configuration. Figures 4 and 5 show the resulting models. These can be compared to an official CAD model in figure 6.

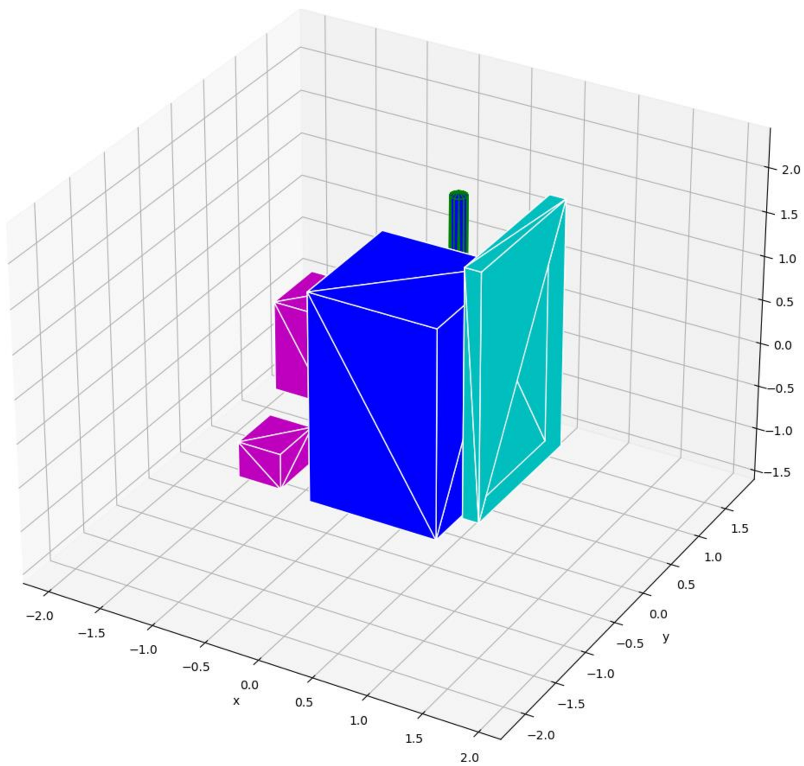


Fig. 4 LRO Launch Configuration

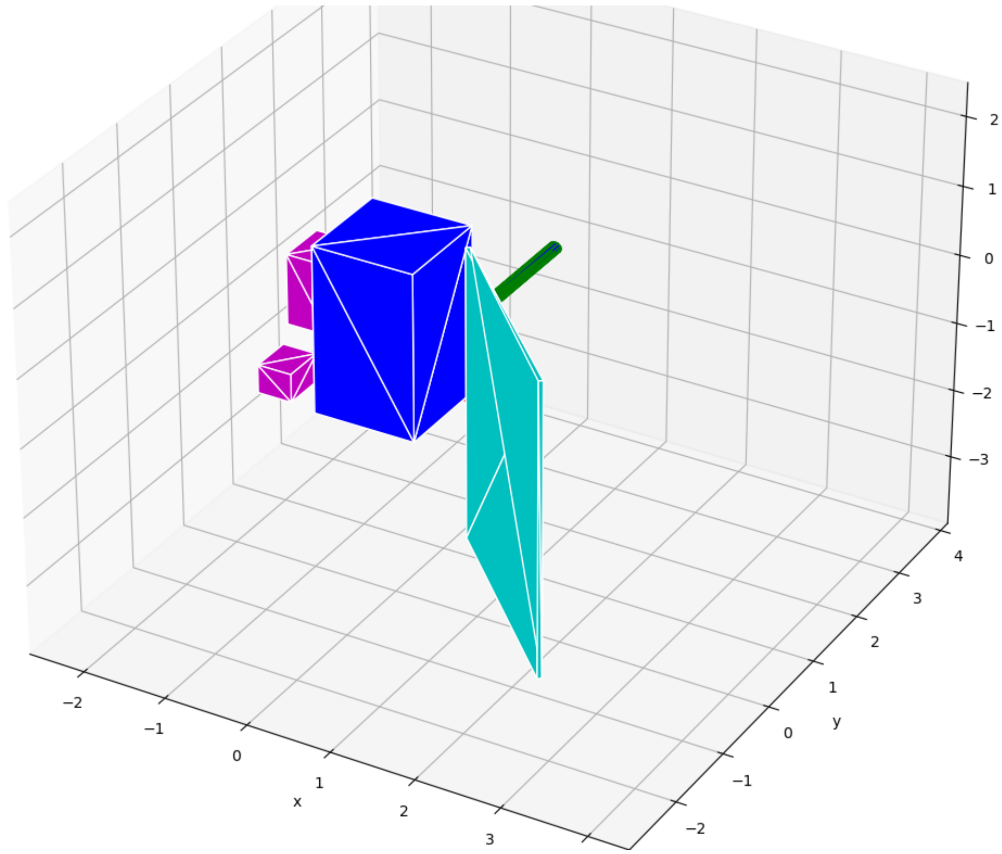


Fig. 5 LRO Deployed Configuration

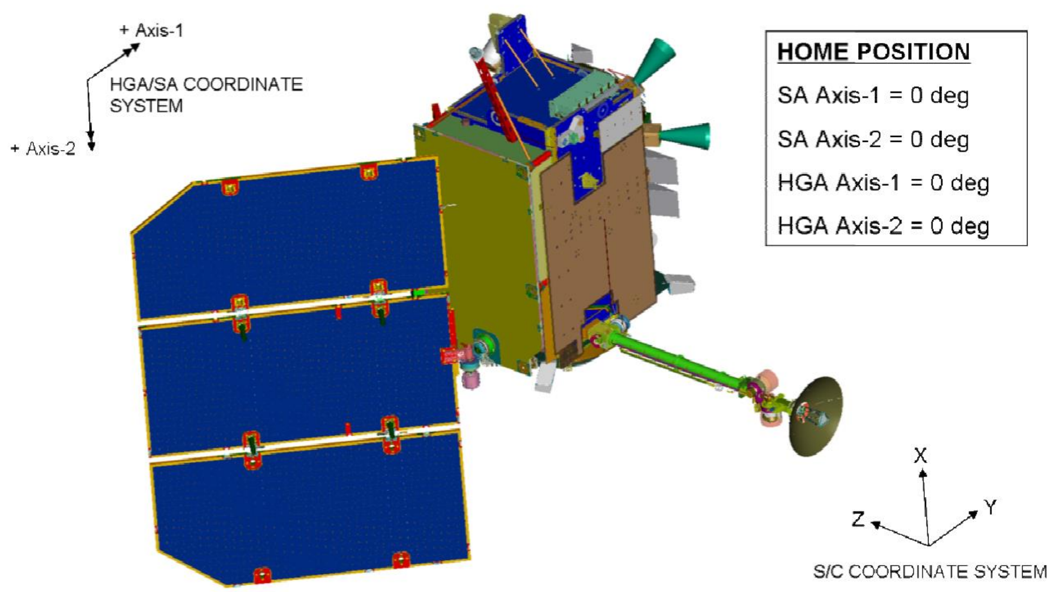


Fig. 6 LRO CAD Model [22]

The moments of inertia of the spacecraft model (as calculated by the geometry framework) are given in table I. A manual calculation produced the same results, verifying that the geometry framework is working correctly. The real inertia tensor for the LRO, measured near its end-of-operation (i.e., empty) state, is also given.

Table I: LRO Moments of Inertia

	Estimated Values	Reported Values [23]
Inertia Tensor (kgm^2)	$\begin{bmatrix} 649 & 22.8 & 66.4 \\ 22.8 & 823 & 33.1 \\ 66.4 & 33.1 & 873 \end{bmatrix}$	$\begin{bmatrix} 591 & 20.7 & 21.5 \\ 20.7 & 850 & 28.0 \\ 21.5 & 28.0 & 923 \end{bmatrix}$
Principal Moments (kgm^2)	908, 807, 630	935, 841, 588

Comparing the two shows the estimated values agree fairly well with the true result. The differences can be explained primarily by different positioning of the solar array and high-gain antenna when the real values were measured. Comparing the principal inertia values shows even closer agreement with less than 3% error for the major principal moment. This is significant as the major principal moment is the value typically used in the initial sizing of attitude control devices. Together, this demonstrates that the framework is able to model a spacecraft effectively and produce accurate mass properties.

B. **Lunar Lander Configuration**

To demonstrate the framework's capabilities for comparing potential configurations of a spacecraft, an example problem was created to design a large lunar cargo lander. The mission flown has three legs. First the lander will be launched towards the Moon where it will perform its own orbital insertion into near-rectilinear halo orbit (NRHO) to rendezvous with the Lunar Gateway space station. This is taken to be the rapid transfer to NRHO in figure 7. Here the lander will pick up the rover which is assumed to be launched separately. The lander will then descend to the surface. After unloading the rover, the lander will finally ascend back to Gateway, so it can be refueled and reused. This comprises a total delta-V of 6,095 m/s with the breakdown shown in figure 7. Between each maneuver, a lumped slew event of 90° is included to represent the use of attitude control thrusters. The payload is a notional pressurized rover based on past concepts. The dimensions are 4.5 m X 4m X 3 m based on the NASA Space Exploration Vehicle rover concept [25]. The mass has been increased to 7000 kg to reflect other concepts and the possibility of packaging the rover with additional cargo [10, 26].

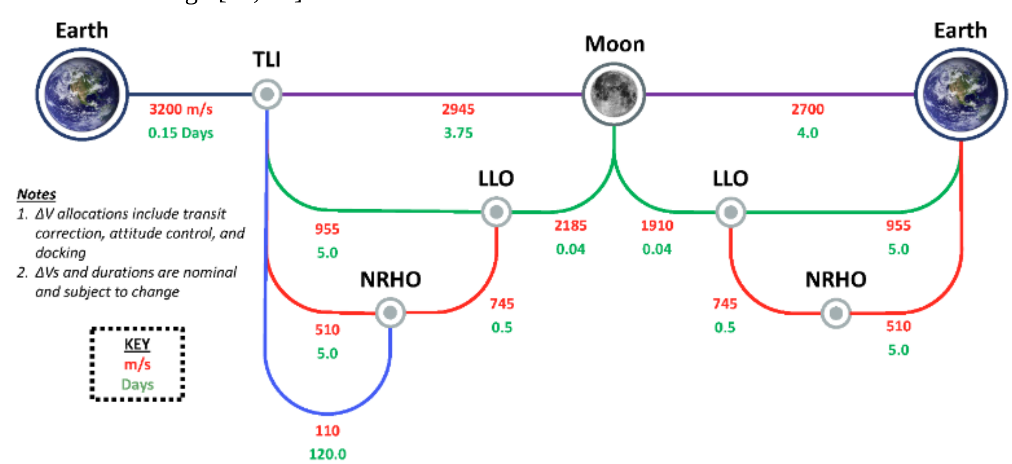


Fig. 7 Earth-to-Moon Delta-V Subway Map [24]

The subsystems for the vehicle were modelled using existing subelements in Dyreqt. These have been developed as part of other ASDL projects. The main propulsion system for the vehicle uses a liquid hydrogen-liquid oxygen engine. The engine is based on the RL-10 engine planned for use on the Altair lander. This is modelled as having an Isp of 460 s, a thrust-to-weight ratio of 40, and operating at an oxidizer-to-fuel ratio of 5.5 [27]. The thrust of the engine was rubberized to provide an overall vehicle thrust-to-weight ratio of 0.4 (in standard Earth gravity). This value is based on an average of several past lunar lander studies from reference 28. Attitude control is accomplished with MMH-NTO RCS thrusters based on the Apollo LM lander. This system consists of 16 thrusters in clusters around the spacecraft with an Isp of 280 s, a thrust-to-weight ratio of 19, and an OFR of 2.1. As with Apollo, the system is required to produce a maximum slew rate of 20°/s around any axis. Based on the Apollo limit of 30 seconds continuous firing for most of its thrusters, the system is required to reach maximum slew rate in 15 s [29, 30]. The Dyreqt RCS thruster subelement sizes the thrusters to provide the necessary torque to achieve these requirements. This calculation incorporates the moment of inertia of the lander and the moment-arm of the thrusters provided by the geometry framework. The propellant consumed by both propulsion systems is used to size their respective propellant tanks. The tank models use the density and pressure of the resource they contain to determine their volume and the necessary thickness of their walls. The length-to-diameter ratio of the tanks is fixed at 1 for the RCS propellant tanks (i.e., the tanks are spherical) while the values for the MPS tanks are varied between 5 possible combinations. There are two of each type of RCS tank while the number of MPS tanks depends on the geometric configuration.

The power for the lander is provided by a set of solar panels and lithium-ion batteries. These are based on models from reference 6. The power load comes from a notional suite of avionics components with additional margin giving a total of ~1 kW. A radiator panel is modelled to handle the waste heat of the electronics. Lastly, the structural mass of the lander is taken to be 31% of the dry mass. This is again based on the results of past lander studies from reference 28.

The geometric configurations considered fall into three categories based on the positioning of the payload on the lander. These basic options were identified in reference 31. The first is top-loading. Here the payload is placed on top of the other lander components. This is the most traditional design and was used by the Apollo LM. The second concept is mid-loading. These place the payload in the middle of the stack with engines below and other components above. This places the payload closer to the surface while still allowing a single central engine. An example of this design is the Blue Origin HLS concept. The last concept is a bottom-loading lander. This places the payload at the bottom of the lander with engines located on either side. An example is the Dynetics HLS concept. This design has the advantage that the payload can be directly dropped onto the surface without additional handling equipment.

For this example, the top-loader was modelled as having a single hydrogen tank below the payload with either 2 or 4 oxygen tanks around it. The RCS tanks were clustered at the bottom of the hydrogen tanks while the solar panels were mounted to the oxidizer tanks. The mid-loader configuration was modelled with a single hydrogen tank above the payload and a cluster of 4 oxygen tanks around the engine below the payload. RCS tanks and panels are mounted to the hydrogen tank. For the bottom-loader two configurations were considered, one with both sets of main propellant tanks on the side and one with the hydrogen tank horizontally mounted above the payload. RCS tanks were mounted to the inboard side tanks while the panels were mounted to the outboard tanks. Unlike the other configurations, these are assumed to be loaded sideways onto the launch vehicle rather than upright. Combined with the 5 options for tank LD ratios, there were a total of 25 configurations examined. The results of the analysis are given in appendix A. Appendix B shows a matrix of images for each concept. Three interesting cases are highlighted below.

1. *Case 1: Minimum Mass*

The lowest mass configuration is the bottom-loader design with a top-mounted hydrogen tank and all spherical tanks (figure 8). This is to be expected, since keeping the tanks spherical and using the minimum number of tanks will minimize the surface area needed to hold a certain volume of propellant. As a result, this minimizes tank mass. In addition, this design results in relatively small moments of inertia while still having a long moment arm for the RCS thrusters. As a result, these thrusters can be fairly small, making them lighter and reducing the propellant needed for attitude maneuvers. The cost of this performance is that this design has the largest bounding radius of any considered. As a result, even though by mass it could be launched on a smaller rocket, only a very large launch vehicle could accommodate its size.

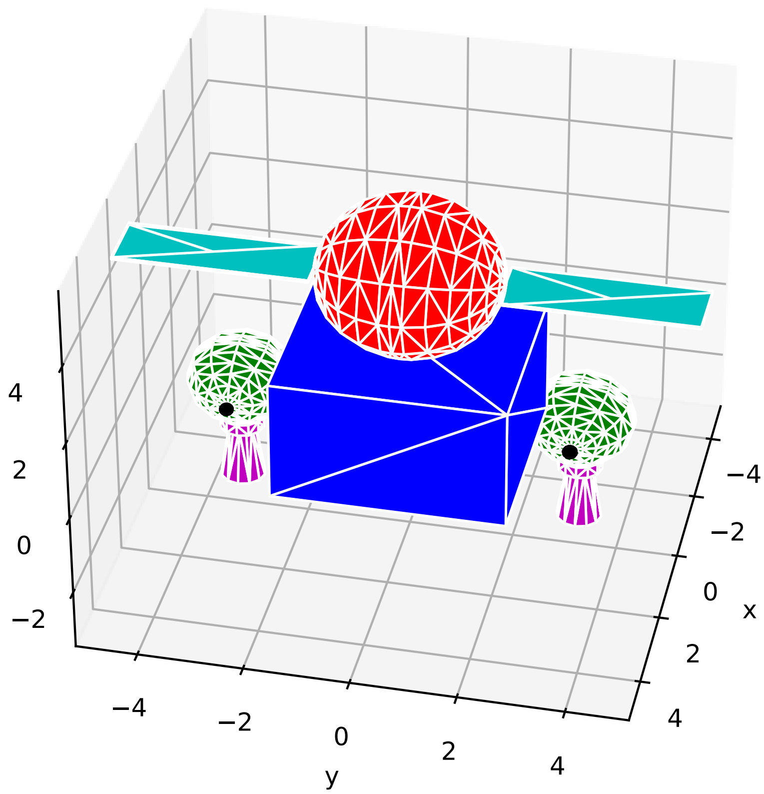


Fig. 8 Minimum Mass Concept

2. Case 2: Minimum Launch Radius

In contrast, the design with the smallest radius was the bottom-loading design with side-mounted propellant and 2:1 hydrogen tanks (figure 9). This small radius would enable it to fit in a much smaller fairing and thus on a wider variety of rockets. It should be noted that the length of the lander is much greater than before. However, launch vehicle fairings are generally much longer than their width, so this is less of a constraint. However, the penalty for this is an additional 1400 kg of mass over the previous configuration. This can be partially attributed to the tanks. By splitting the hydrogen tank in two and using a larger length to diameter ratio, more material is needed to form the tanks which increases mass. The remaining mass increase comes from the increased length of this design which gives it the greatest moment of inertia of the designs considered here. This large moment of inertia means this configuration needs larger RCS thrusters to meet the agility requirements and more propellant to make attitude maneuvers.

3. Case 3: Compromise Design

A compromise between these two is the all-spherical mid-loader (figure 10). The tank layout of this configuration is intermediate in efficiency between the other two, resulting in a mass partway between them. However, by moving the oxidizer tanks and engine below the payload and mounting the hydrogen tank upright, it is able to keep its footprint within the boundaries of the payload, giving the smallest possible radius for a vertically launched design and keeping the radius less than the minimum mass case. In addition, mounting the tanks in this way gives a very small moment of inertia which results in the smallest RCS thrusters of any design.

One downside of this configuration is that it is a mid-loader rather than a bottom loader. As a result, unloading the payload will require either pre-positioned unloading equipment on the surface or equipment on the lander itself which would increase its mass from what was modelled here.

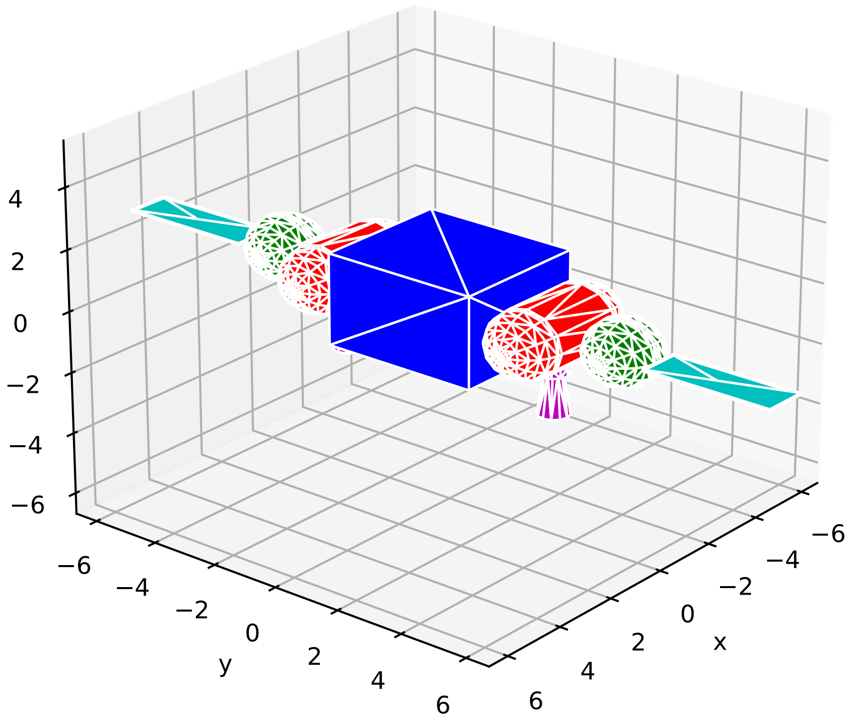


Fig. 9 Minimum Radius Concept

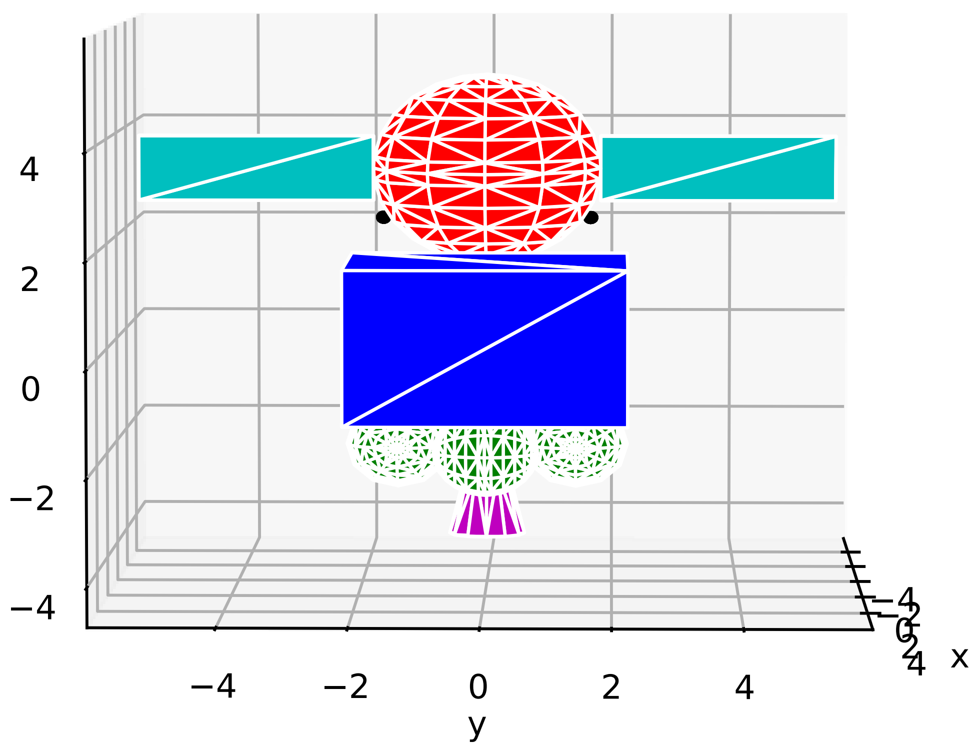


Fig. 10 Compromise Mid-Loading Concept

V. Conclusion

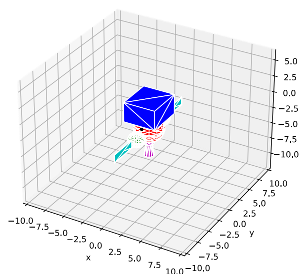
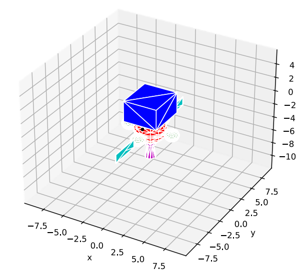
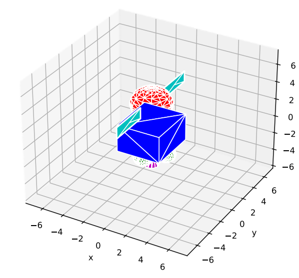
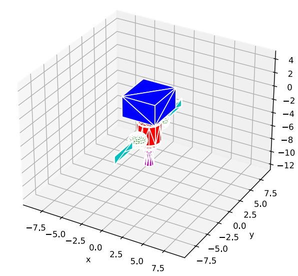
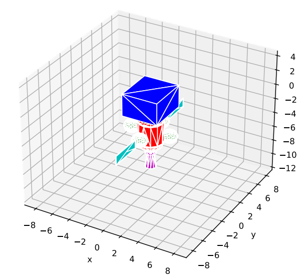
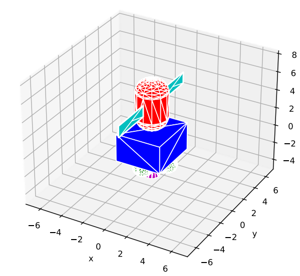
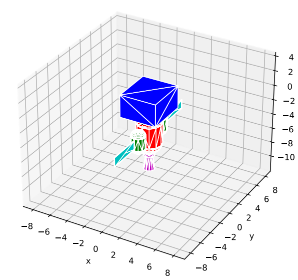
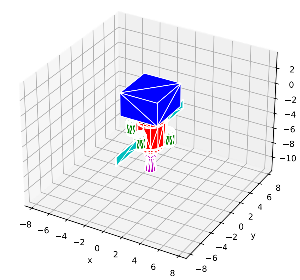
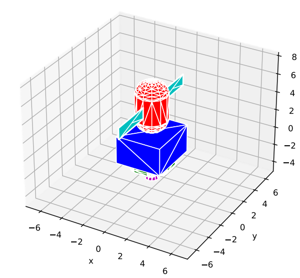
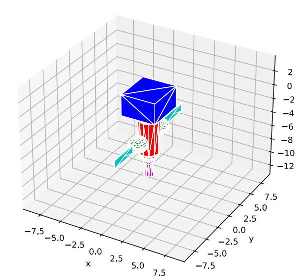
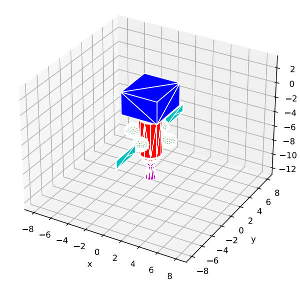
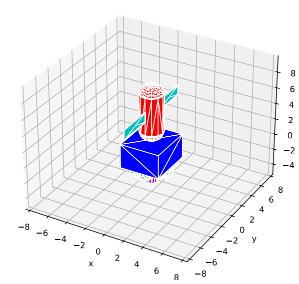
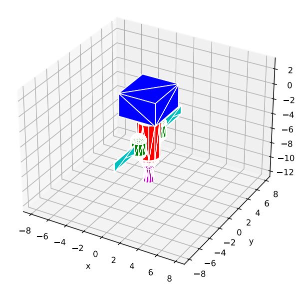
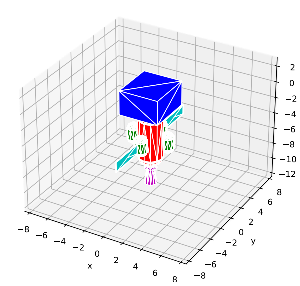
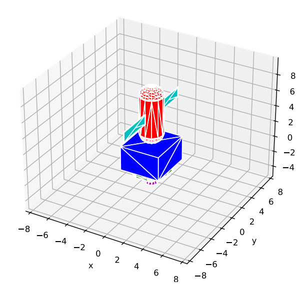
In summary, this paper has developed a method for describing and analyzing the geometric configuration of spacecraft within Dyreqt. Unlike previous studies which used a layer-based description of a spacecraft's geometry, this framework defines the configuration at the level of individual components. This enables the framework to describe any arbitrary geometric configuration and allows it to interface with standard CAD formats. By using this framework, Dyreqt's capabilities have been greatly expanded by incorporating geometric information into subsystem modelling and optimization. It has also allowed Dyreqt results to be used in automatically generated visual models of spacecraft. These capabilities were demonstrated with an example lunar lander design problem. By using the geometric framework, it was possible to determine how the configuration of a lander affects its dimensions at launch and the sizing of its attitude control systems. This allowed trades to be made between different configurations which could not previously be done with Dyreqt. Moving forward, this framework will be applied to a wider range of spacecraft subsystems to create additional new capabilities for Dyreqt.

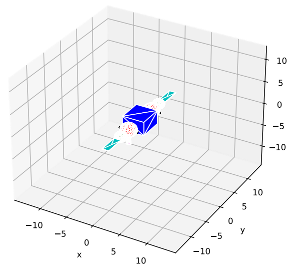
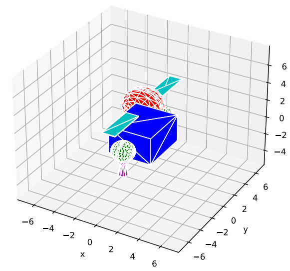
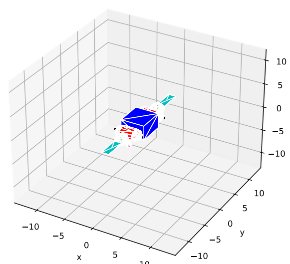
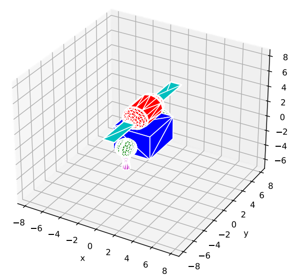
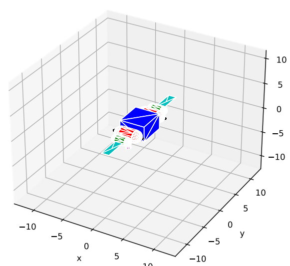
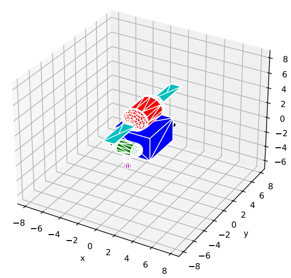
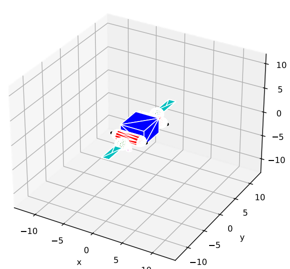
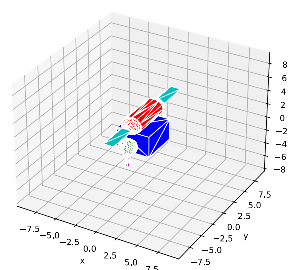
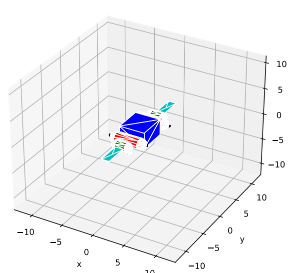
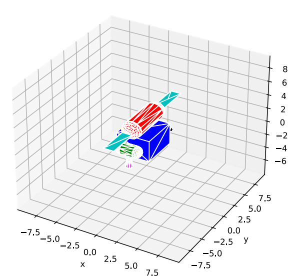
Appendix

A. Table of Lunar Lander Results

Configuration	LD-H2	LD-O2	Gross Mass (kg)	Dry Mass (kg)	Maximum Principal Moment ($kg \cdot m^2$)	RCS Thrust (N)	Radius (m)	Height (m)
Top-Loading with 2 Oxygen Tanks	1	1	12700	1000	159000	300	3.9	8.9
	2	1	13500	1200	177000	394	3.5	10.7
	2	2	13900	1300	173000	431	3	10.8
	3	1	13700	1200	207000	501	3.3	12.3
	3	2	14000	1300	205000	562	3	12.3
Top-Loading with 4 Oxygen Tanks	1	1	13100	1100	114000	230	3.5	8.9
	2	1	14000	1300	143000	346	3.1	10.8
	2	2	14400	1400	145000	386	3	10.8
	3	1	14200	1300	179000	476	3	12.3
	3	2	14600	1400	182000	540	3	12.4
Bottom-Loading with Overhead Hydrogen Tank	1	1	12300	900	151000	235	3.9	8.6
	2	1	13000	1100	158000	245	3.4	8.7
	2	2	13300	1100	141000	222	3.3	7.6
	3	1	13100	1100	161000	250	3.2	8.7
	3	2	13300	1100	144000	227	3.1	7.6
Bottom-Loading with Side-Mounted Tanks	1	1	13200	1100	460000	399	2.7	14.5
	2	1	13700	1200	382000	375	2.6	13
	2	2	14000	1300	365000	376	2.6	12
	3	1	13900	1300	353000	366	3.1	12.4
	3	2	14200	1300	336000	367	3.1	11.4
Mid-Loading	1	1	13000	1100	89000	106	3	8.9
	2	1	13800	1300	114000	120	3	10.8
	2	2	14200	1300	115000	120	3	10.8
	3	1	13800	1300	136000	127	3	12.3
	3	2	14200	1300	137000	128	3	12.4

B. **Table of Lunar Lander Images**

LD - H2	LD - O2	Top-Loading with 2 Oxygen Tanks	Top-Loading with 4 Oxygen Tanks	Mid-Loading
1	1			
2	1			
2	2			
3	1			
3	2			

LD-H2	LD-O2	Bottom-Loading with Side-Mounted Tanks	Bottom-Loading with Overhead Hydrogen Tank
1	1		
2	1		
2	2		
3	1		
3	2		

References

[1] Trent, D. J., "Integrated Architecture Analysis and Technology Evaluation for Systems of Systems Modeled at the Subsystem Level," dissertation, 2017.

[2] Robertson, B., Ramos, E. M., Diaz, M. J., and Mavris, D., "A Conceptual Design Study for an Unmanned, Reusable Cargo Lunar Lander," International Astronautical Congress, vol. 70, 2019.

[3] Trent, D. J., Edwards, S. J., and Chappell, M. B., "Design of a Mars Common Chemical Transportation System," AIAA SCITECH 2024 Forum, Jan. 2024.

[4] Peak, V., Balchanos, M. G., and Mavris, D. N., "A Model-Based Trade-Off Analysis for Sizing Integrated Lunar Surface Systems," AIAA SCITECH 2024 Forum, Jan. 2024.

[5] Leve, F. A., Peck, M. A., and Hamilton, B. J., Spacecraft Momentum Control Systems, Heidelberg, Germany: Springer, 2015.

[6] Wertz, J. R., Everett, D. F., and Puschell, J. J., Space Mission Engineering: The New SMAD, Torrance, CA: Microcosm Press, 2018.

[7] Iida, H., and Keiken, N., "A New Approach to Magnetic Angular Momentum Management for Large Scientific Satellites," NEC Research & Development, vol. 37, 1996, pp. 60–77.

[8] Tamrazian, S., "Development of a Coupled Orbit-Attitude Propagator for Spacecraft of Arbitrary Geometry," dissertation, 2019.

[9] Heineman, W., Design Mass Properties II: Mass Estimating and Forecasting for Aerospace Vehicles Based on Historical Data, Nov. 1994.

[10] Jefferies, S., Collins, T., Cianciolo, A. D., and Polsgrove, T., "Impacts of launch vehicle fairing size on human exploration architectures," 2017 IEEE Aerospace Conference, Mar. 2017.

[11] Requicha, A. G., "Representations for Rigid Solids: Theory, Methods, and Systems," ACM Computing Surveys, vol. 12, Dec. 1980, pp. 437–464.

[12] Tulsiani, S., Su, H., Guibas, L. J., Efros, A. A., and Malik, J., "Learning Shape Abstractions by Assembling Volumetric Primitives," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jul. 2017.

[13] Chua, C. K., Leong, K. F., and Lim, C. S., Rapid Prototyping, River Edge, New Jersey: World Scientific Publ. Co., 2004.

[14] Tae Ha Park and Simone D'Amico. "Rapid Abstraction of Spacecraft 3D Structure from Single 2D Image," AIAA 2024-2768. AIAA SCITECH 2024 Forum. January 2024.

[15] Lourakis, M., and Terzakis, G., "Efficient absolute orientation revisited," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2018.

[16] Markley, F. L., "Statistical attitude determination," Encyclopedia of Aerospace Engineering, Dec. 2010.

[17] Cline, D., Variatiosninal Principles in Classical Mechanics, Davis, CA: LibreTexts, 2021.

[18] Gao, S., and Pan, H., "Solving Smallest Enclosing Circle Problem by Wading Across Stream Algorithm," Journal of Physics: Conference Series, vol. 2173, 2022, p. 012081.

[19] "Lunar Reconnaissance Orbiter (LRO): Leading NASA's Way Back to the Moon," NASA Press Kit, Jun. 2009.

[20] Lunar Reconnaissance Orbiter Project: Technical Resource Allocations Specification, NASA GSFC, May 31, 2006.

[21] Tooley, C., "Lunar Reconnaissance Orbiter Spacecraft & Objectives," 2006 AIAA-Houston Annual Technical Symposium, May 2006.

[22] Shah, N., "A modified Lunar reconnaissance orbiter (LRO) high gain antenna (HGA) controller based on Flight Performance," AIAA Guidance, Navigation, and Control Conference, Jun. 2010.

[23] Kaufman, J. M., "Automated maneuver design and checkout for the Lunar Reconnaissance Orbiter," thesis, 2014.

[24] Trent, D., and Edwards, S. J., "Analysis of Alternative Architectures for a 2024 Lunar Sortie," ASCEND 2020, Nov. 2020.

[25] "Space Exploration Vehicle Concept," NASA Headquarters.

[26] Creel, K., Frampton, J., Honaker, D., McClure, K., Zeinali, M., Bhardwaj, M., Bulsara, V., Kokan, D., Shariff, S., Svarverud, E., and Wirz, R., "Pressurized Lunar Rover (PLR)," NASA/USRA Advanced Design Program 8th Summer Conference, 1994.

[27] "RL10 Propulsion System," Aerojet Rocketdyne, Available: https://www.rocket.com/sites/default/files/documents/RL10_data_sheet.pdf.

[28] Connolly, J. F., After LM: NASA Lunar Lander Concepts Beyond Apollo, Washington, DC: National Aeronautics and Space Administration, 2020.

[29] "Apollo Operations Handbook: Lunar Module, Volume 1 Subsystems Data," Grumman, June 15, 1969.

[30] Shelton, H., "Apollo Experience Report - Guidance and Control Systems: Lunar Module Stabilization and Control System," NASA JSC, Nov. 1975.

[31] Donahue, B., and Fowler, C., "Lunar Lander Configuration Study and Parametric Performance Analysis," 29th Joint Propulsion Conference and Exhibit, 1993.