Lunar Descent Vehicle Simulation

Chase Kincaid*, Victoria Rutledge*, Christian Mounce*, Peyton Yates*
Mississippi State University, Mississippi State, MS 39762

Within society and the world of Aerospace Engineering, simulations are an integral part of mission planning, testing and evaluation. It is critical to spaceflight missions such as Apollo 11. Our group wants to perform a validation test with the Apollo 11 Mission. The first step in this process is to create a CAD model of the Apollo Eagle Module. With this, our group can outline the mission parameters in Simulink, create a 3-D CAD model in SOLIDWORKS, and apply Unity-3D to implement a simulated environment with real-world physics engine.

Nomenclature

MATLAB = Matrix Laboratory
CAD = Computer Aided Design
C# = C-Sharp
DCM = Direction Cosine Matrix
DOF = Degree of Freedom
EOM = Equations of Motion
MOI = Moment of Inertia
NASA = National Aeronautics and Space Administration
UDP = User Datagram Protocol
Unity3D = Unity Real-Time Development Platform

I. Introduction

SIMULATIONS are an important aspect of the Aerospace industry by allowing for future spaceflight missions to be researched, planned, tested, and revised within budget and time constraints. Our goal is to create and visualize a lunar lander using Simulink and Unity 3D with certain parameters to observe and control a space vehicle.

For our project, our team will use SOLIDWORKS to design an Apollo Eagle Module. Our group chose to work with the Apollo Eagle Module because there is data provided by NASA so that we can do a validation test with our simulation. The CAD model can demonstrate the model that can be used in the Unity3D visual simulation.

II. Objectives

Our first objective for this project is to outline the mission design of the Apollo Eagle with SOLIDWORKS, which the Apollo Eagle Module Specifications needed to be found. Then, the CAD model of the Apollo Eagle needed to be completed. Once the SOLIDWORKS is completed, then the Simulink Model of the Vehicle Dynamics can be worked on. The Simulink Model needs to include the forces, MOI, and the Lunar Model. Because our group wanted to add a joystick input to the Lunar Module Simulation, a joystick input needs to be added to the Simulink code. While the Simulink code is being worked on, the Unity3D build can be worked on as well. In Unity3D, the Moon terrain build, visuals, and the camera angles need to be completed.

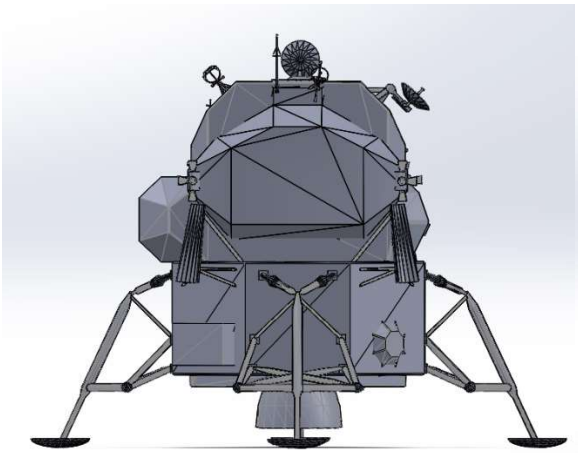
III. Discussion

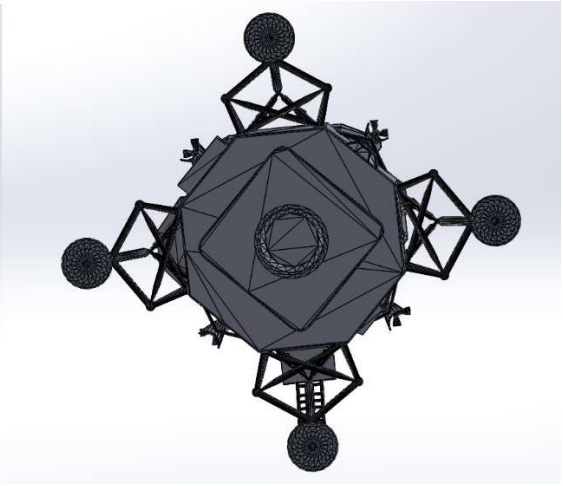
Below describes the discussion of the project in separate sections.

* Undergraduate Student, Mississippi State University
* Undergraduate Student, Mississippi State University
* Undergraduate Student, Mississippi State University
* Undergraduate Student, Mississippi State University

A. CAD Model

The CAD Module of the Apollo Eagle Module is shown below in Figure 1-3.





Figures 1 & 2. Side View and Bottom View of the Lunar Lander, respectively

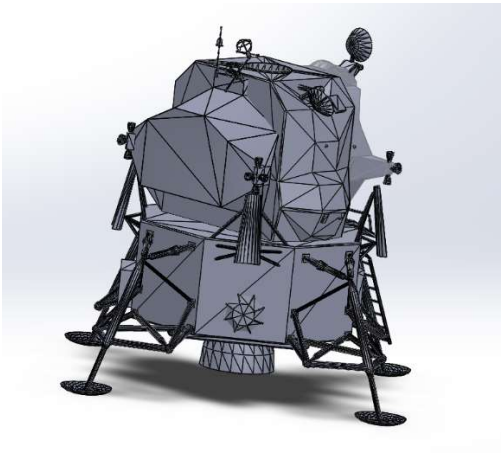


Figure 3. Front View of Lunar Lander

The figures show the front, bottom, and the side view of the replica of the Apollo 11 lunar lander. This CAD model is used in the Unity3D visual simulation. For the thrusters, the lander utilizes twelve individual thrusters.

B. Simulink Vehicle Dynamics

The goal of the Simulink code is to input the yaw, pitch, roll angles and forces on the Lunar Lander so that the correct vehicle dynamics are applied to the body. The overall outline of what the Simulink code is achieving is shown in Figure 4.

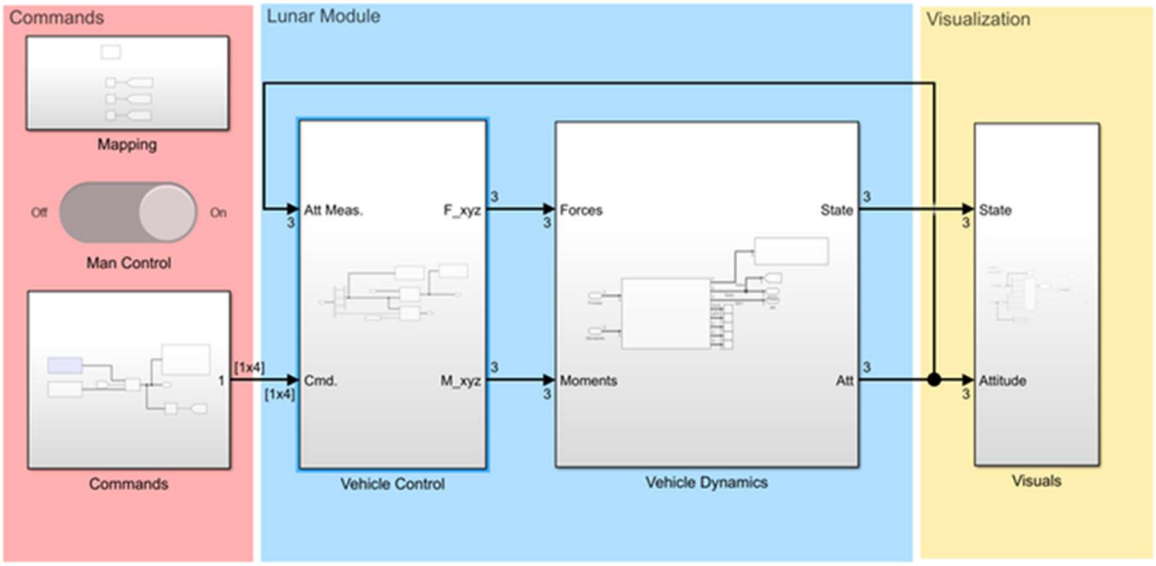


Figure 4. Stages of the Simulink Code

As shown, there are three main components of the Simulink code. First is the initial commands that is going to be subjected on the lunar module. Next, these commands being applied to the vehicle need to be transformed to identify the rates and position of the vehicle. Lastly, the visualization block is how the lunar lander is able to be seen on how the inputs effect the dynamics.

The 6DOF Euler angles Simulink Toolbox provides a set of equations that can be used to simulate the motion of a rigid body in 3D space using Euler angles. These equations are based on the Euler angles representation of the body's orientation and consider the body's angular velocities and accelerations. Within the aerospace block set, the pilot joystick input was utilized to implement roll, pitch and yaw in the form of Euler angles as shown below in Figure 5. The initial throttle is scaled from 1/100 to regulate our system's theoretical fuel consumption and produced force. These forces and angles are then input as commands into our vehicle's dynamics.

The 6DOF block set in the Aerospace Toolbox provides outputs for each six degree of freedom which represent the three rotations, roll, pitch, and yaw. For the inputs of the Euler angles, the initial values for each are shown in Equation 1.

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} 0 \\ 90 \\ 0 \end{bmatrix} degrees \tag{1}$$

These inputs are shown in the manual control inputs shown in Figure 5.

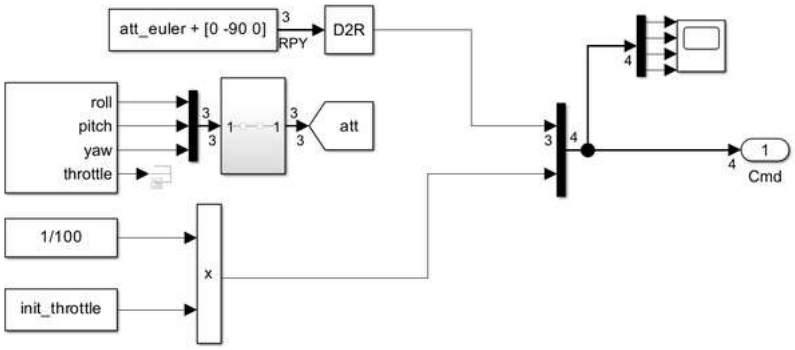


Figure 5. Manual Control Inputs

The block set input applied force and applied moments and can output the velocity, position, Euler rotation angles, and the DCM. These degrees of freedom are expressed in terms of Euler angles, which define the orientation of the body with respect to a fixed reference frame. It is important to note that this block set has some limitations. The limitations are that the applied forces act at the center of gravity and that the mass and inertia are constant.

The general mathematical process can reduced down to the following equations. First, the moments and the inertial matrices are inputted into the 6DOF Block shown in Equations 2 and 3, where L, M, and N are applied moments.

.

$$M = \begin{bmatrix} L \\ M \\ N \end{bmatrix} \tag{2}$$

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \tag{3}$$

With the given inputs and the Euler angles, the quaterion can be solved for from the Euler angles which is shown in Equation 4.

$$q = \frac{1}{2}\begin{bmatrix} C(\psi)C(\theta)C(\phi) + S(\psi)S(\theta)S(\phi) \\ C(\psi)C(\theta)C(\phi) - S(\psi)S(\theta)S(\phi) \\ S(\psi)S(\theta)S(\phi) + C(\psi)C(\theta)C(\phi) \\ S(\psi)S(\theta)S(\phi) + C(\psi)C(\theta)C(\phi) \end{bmatrix} \tag{4}$$

Lastly, the quaternion is used to calculate the angular rates p, q, and r used in later parts in the Simulink code.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 2 * (q1q3 - q0q2) \\ 2 * (q0q1 + q2q3) \\ q0q0 - q1q1 - q2q2 + q3q3 \end{bmatrix} \tag{5}$$

The subsystem that uses the previous equations are used in the 6DOF Body to Quaternion Block is shown below in Figure 6.

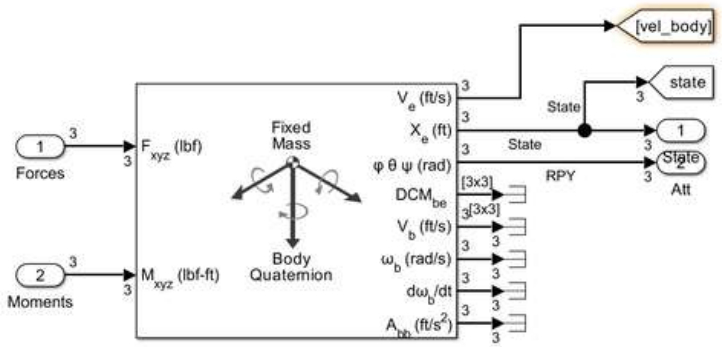


Figure 6. Aerospace EOM 6DOF Block set

The equations in the toolbox take inputs such as the body's mass, moments of inertia, external forces and torques, and initial conditions, and output the body's position, velocity, and orientation at any given time during the simulation. This is how the Simulink code tracks the rotation and the movement of the lunar lander as the inputs are changing.

The next main part is the attitude control block. This block takes inputs of the position and performs calculations to find the number of jets, rotational position, initial fire, coast 1, fire 2, then coast 2. The outputs for the jet being fired are commanded to fire based on the variation of distance between the jet and total thrust. This attitude process is shown in Figure 7. Total thrust is commanded between 0 and 1, and is then multiplied by the total amount of thrust and the objects center of gravity or moment arm which outputs moments in the body axis.

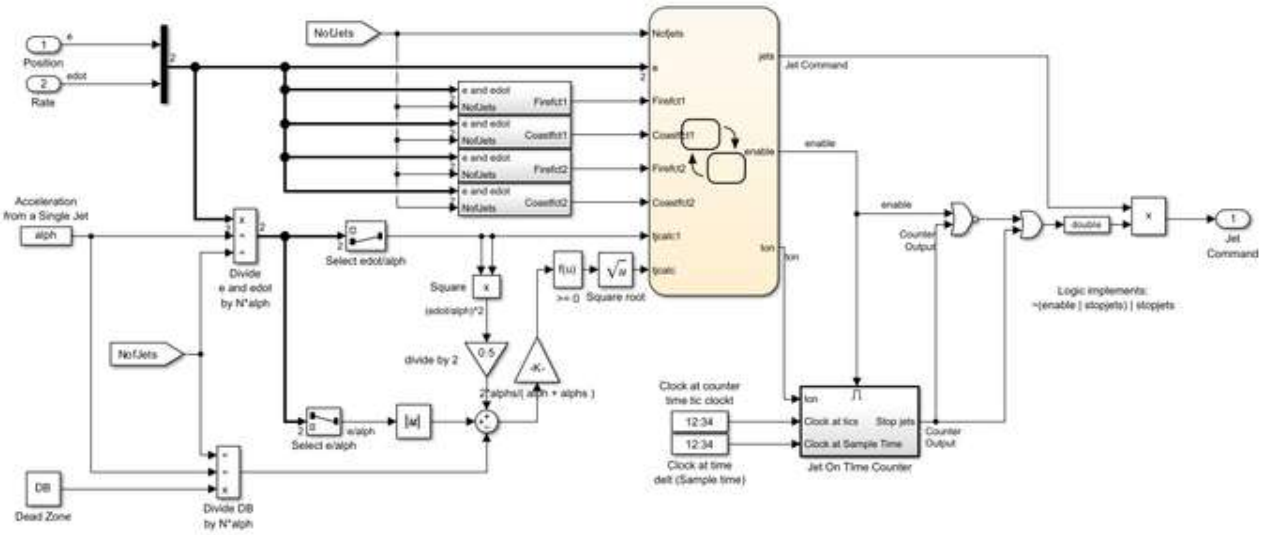


Figure 7. Attitude Control Law Block

Once all of the equations and values are calculated throught the Simulink Code the side, vertical, forward, pitch, yaw, and roll can be sent to the UDP via translations so that Unity3D can read the correct values. This process of transmitting the data is shown in Figure 8.

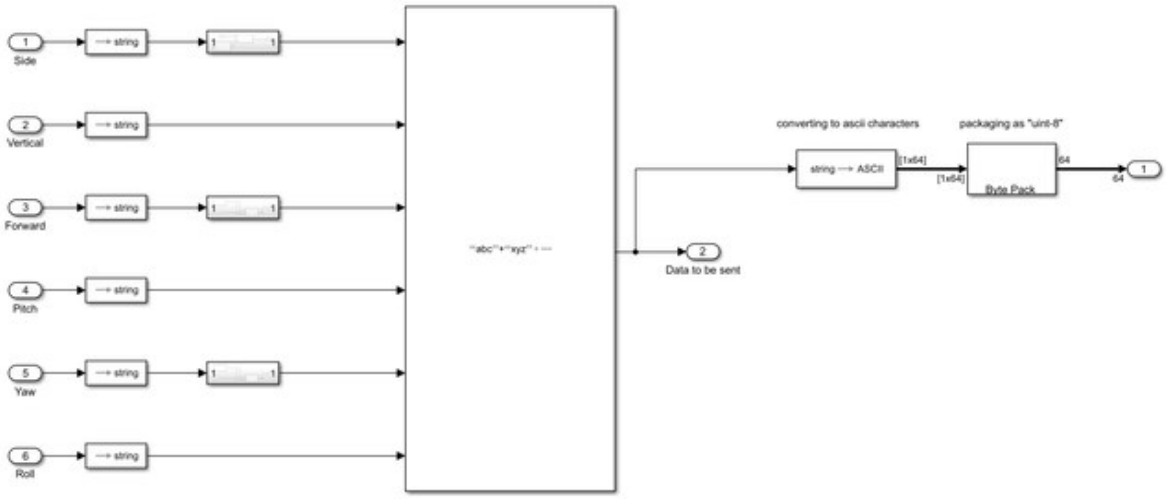


Figure 8. Trasmit of the Simulink Vehicle Dynamics Data to Unity3D

C. Unity3D Scene Build

The Unity3D build is shown below in Figure 9. This figure shows the components of the simulation and what you would see when running the simulation.

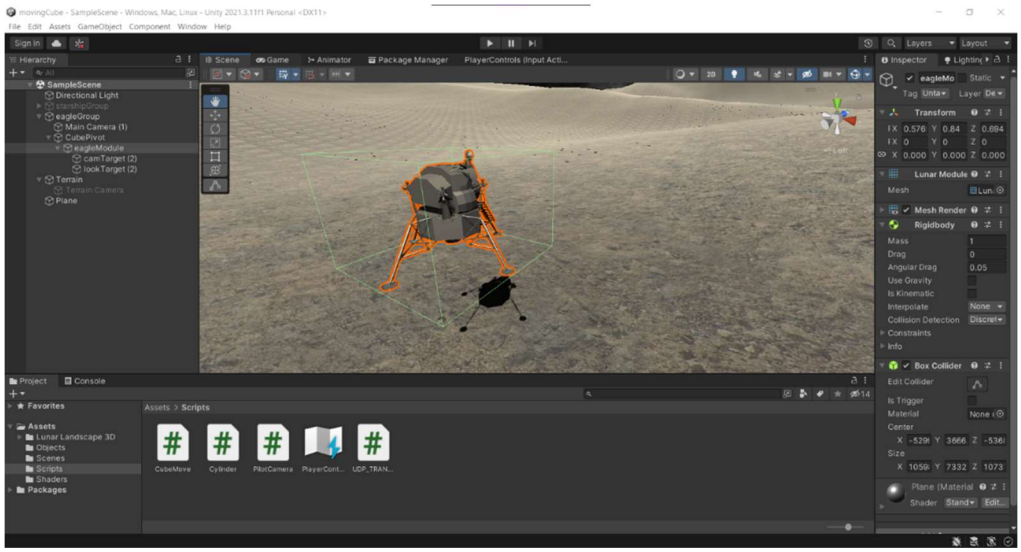


Figure 9. Image of the Unity3D Simulation Build

In the Unity3D script, the C# code is shown in Figures 10 and 11. Figure 10 shows the UDP Code which imports the telemetry data from Simulink. The telemetry data being imported is the position allocated in roll, pitch, and yaw. While Figure 11 shows how the inputs are affecting the body of the lunar lander.

```
25
26      // Send data to port 1234
27      // format: string with 3 numbers separated by ','
28      // encoding: 'utf-8'
29
30
31
32      void Start()
33      {
34          udpServer = new UdpClient(1234);
35          t = new Thread(() => {
36              while (true)
37              {
38                  this.receiveData();
39              }
40          });
41          t.Start();
42          t.IsBackground = true;
43          remoteEP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 41234);
44      }
45
```

Figure 10. Shows the UDP C# Code that pulls in the telemetry data from Simulink Code

```
47
48      void Update()
49      {
50          transform.localPosition = new Vector3(transformPosition[0], transformPosition[1], transformPosition[2]);
51          transform.localRotation = Quaternion.Euler(transformRotation[0], transformRotation[1], transformRotation[2]);
52      }
53
54      private void OnApplicationQuit()
55      {
56          udpServer.Close();
57          t.Abort();
58      }
59
60      private void receiveData()
61      {
62
63          byte[] data = udpServer.Receive(ref remoteEP);
64          if (data.Length > 0)
65          {
66              var str = System.Text.Encoding.Default.GetString(data);
67              Debug.Log("Received Data: " + str);
68              string[] messageParts = str.Split(',');
69
70              // Position
71              transformPosition[0] = float.Parse(messageParts[0], CultureInfo.InvariantCulture);
72              transformPosition[1] = float.Parse(messageParts[1], CultureInfo.InvariantCulture);
73              transformPosition[2] = float.Parse(messageParts[2], CultureInfo.InvariantCulture);
74
75              // Orientation
76              transformRotation[0] = float.Parse(messageParts[3], CultureInfo.InvariantCulture);
77              transformRotation[1] = float.Parse(messageParts[4], CultureInfo.InvariantCulture);
78              transformRotation[2] = float.Parse(messageParts[5], CultureInfo.InvariantCulture);
79          }
80      }
81  }
```

Figure 11. Shows the rapid development version of the telemetry data and applying it to an object in Unity3D

VI. Results

The simulation was overall successful with the simplified constraints our group put in the Simulink code. For the results, the simulation was ran for three separate trials to examine how the input changes the flight path of the lunar lander. These different inputs were examined in three different trial runs.

A. Trial One

For Trial One, our group inputted a pitch of 20 degrees. Below shows the flight path graph in Figure 12.

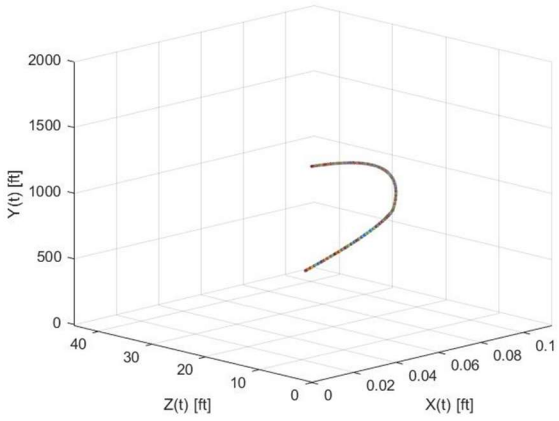


Figure 12. Constant 29 degree Pitch Angle

It is important to note that the joystick is sensitive that the data being translated is not exactly zeroed out. However, the pitch angle was still roughly 29 degrees. Our group also recorded the changes in the yaw, pitch, and roll of the lunar lander object in Figure 13 via the Scope graphs in Simulink.

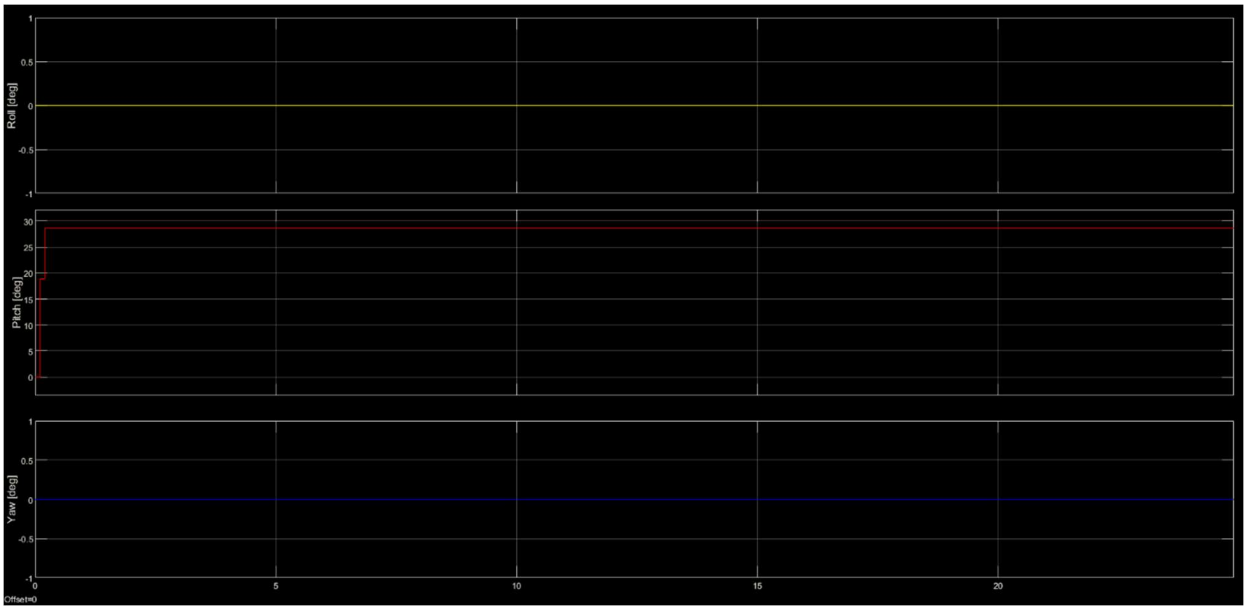


Figure 13. Scope of Yaw, Pitch, and Roll Angles For Trial One

B. Trial Two

To examine a different range in motion on the joystick, an input of a constant pitch and roll angles at approximately 29 degrees shown in Figure 14.

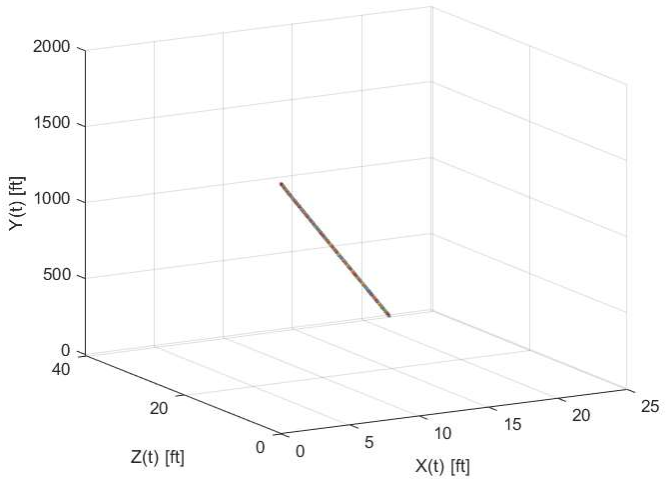


Figure 14. Constant 29 degree Pitch and Roll Flight Path

The scope graph of these flight path angles are shown in Figure 15.

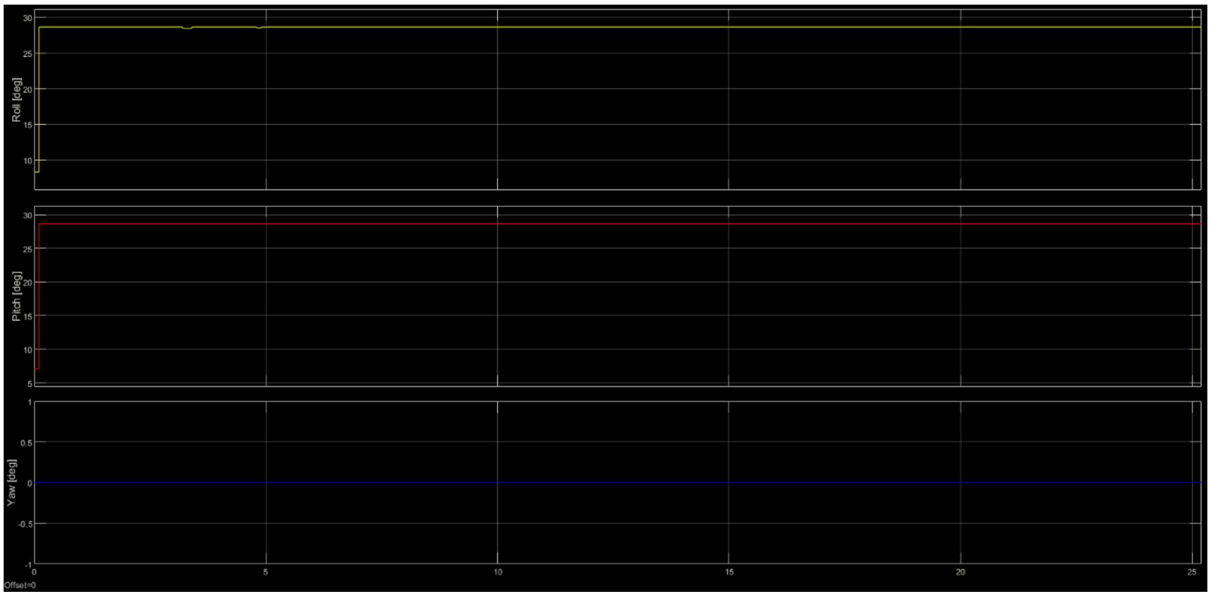


Figure 15. Scope of Yaw, Pitch, and Roll Angles For Trial Two

C. Trial Three

Lastly, an input of various random inputs were applied to the joystick and this flight path is shown in Figure 16.

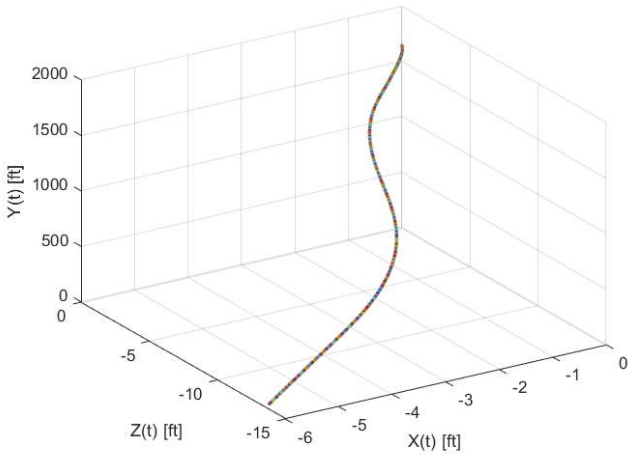


Figure 16. Random Inputs on the Joystick Flight Path

Next, the scope graph showing the flight path angles is shown in Figure 17.

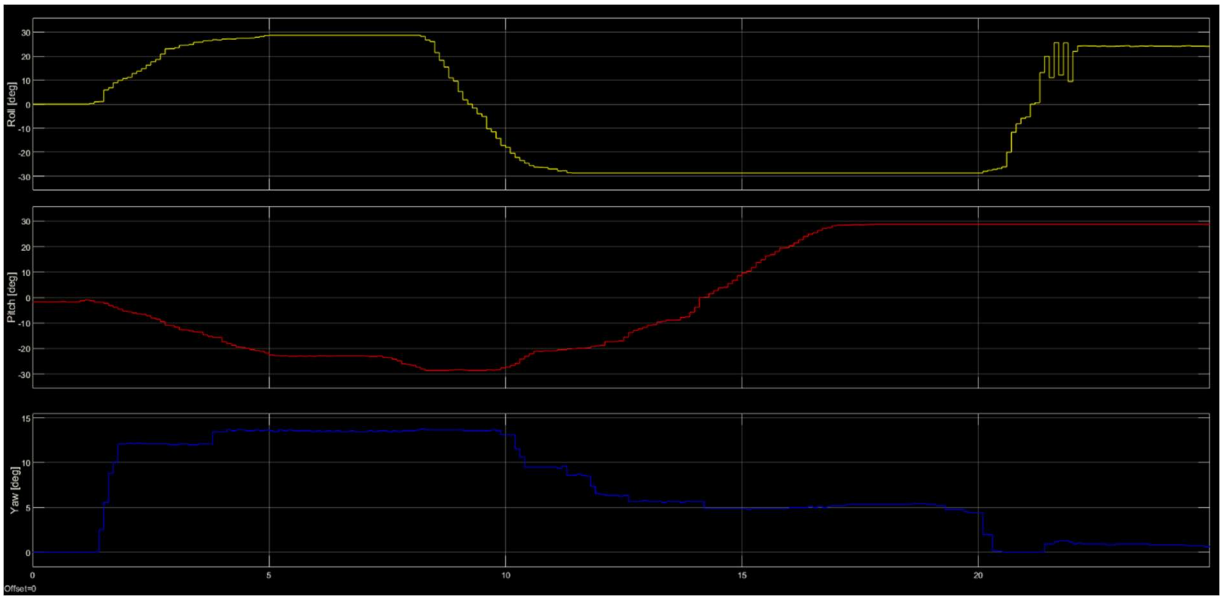


Figure 17. Scope of Yaw, Pitch, and Roll Angles For Trial Three

V. Conclusion

In conclusion, the Simulink graphs result in a 3-Dimensional view of the Lunar Modules flight path. Various input commands were implemented on the joystick. These commands affected the vehicle dynamics of our lunar lander. This Simulink code can serve as a computational and experimental guide for future endeavors such as automated code or any other lunar simulations. A comparative analysis of documented missions can be achieved once optimization of the controls laws and automated flight are accomplished. The experimental results and simulation validate the theoretical framework of attitude dynamics, moments of inertia, flight control systems. Overall this project served as a worthwhile project to connect theoretical knowledge attained in the classroom to practical usage in a real world lunar descent vehicle simulation.

Acknowledgments

This group acknowledges our faculty advisors Mr. Calvin Walker and Dr. Yang Cheng for giving us input guidance along the way.

References

[1]Akharas, I., Hennessey, M. P., and Tornoe, E. J, "Simulation and Visualization of Dynamic Systems in Virtual Reality Using SolidWorks, MATLAB/Simulink, and Unity," Proceedings of the ASME 2020 International Mechanical Engineering Congress and Exposition, 16-19 Nov. 2020.

[2]Bennett, F. V., "Apollo Experience Report: Mission Planning for Lunar Module Descent and Ascent," NASA Available: https://ntrs.nasa.gov/api/citations/19720018205/downloads/19720018205.pdf

[3]Dornheim, M. A., "Planetary Flight Surge Faces Budget Realities," *Aviation Week and Space Technology*, Vol. 145, No. 24, 9 Dec. 1996, pp. 44-46.

[3]F. Zhang, G.-R. Duan, integrated translational and rotational control for the terminal landing phase of a lunar module, Aerosp. Sci. Technol. 27 (1) (2013) 112–126, http://dx.doi.org/10.1016/j.ast.2012.07.003.

[4]Goh, W. "Preliminary Design of Reusable Lunar Lander Landing System," Master's, Luleå University of Technology, 2017.

[5]Hallowell, H. E., Culp, R. D., and Sostaric, R. R., Guidance and control 2007: Proceedings of the 30th Annual aas rocky mountain guidance and control conference held February 3-7, 2007, Breckenridge, Colorado, San Diego, CA: Published for the American Astronautical Society by Univelt, 2007.

[6]MATLAB, Matrix Laboratory, R2019b Ver., MathWorks, 28 Sept. 2021.

[7]Morrow, J. M. "Design and Analysis of Lunar Lander Control," Department of the Aeronautics and Astronautics at Massachuetts Intsitute of Technology, June 2012.

[7]Sagliano, M. "Apollo 11 Reloaded: Optimization-based Trajectory Reconstruction," American Institute of Aeronautics and Astronautics, Jan. 2021.

[8]Sostaric, R., and Rea, J., "Powered descent guidance methods for the Moon and Mars," AIAA Guidance, Navigation, and Control Conference and Exhibit, 2005. https://doi.org/10.2514/6.2005-6287

[9]Woycechowsky, B. Lunar Module Moon-Referenced Equations of Motion," Center for Technology & Innovation, Inc., 2021.
[10]Unity, Unity3D, 2022.20b4 Ver., Unity Technologies, 28 July 2022.