ENS 491-492 – Graduation Project

Final Report

Project Title:

Modeling and Implementing Rocket

Control System for Orbiting

Group Members:

Fatma Khalil - 29999

Rawan Basem - 28616

Osman Eren Genç - 27811

Utkan Yılmaz - 28447

Supervisor(s): Melih Türkseven

Date: 12/05/2024

1. EXECUTIVE SUMMARY

A long-standing objective in the field of aerospace engineering is to create a fully autonomous rocket. This project utilizes an extremely powerful simulator called “Kerbal Space Program”, also called KSP, with the aim of trying to create a script that enables autonomous rocket control using thrust vectors instead of relying on an autopilot to perform a specific task; orbiting the home planet “Kerbin”. The project was implemented using a mod called kOS (Kerbal Operating System) that was developed by the simulator’s online community. By using kOS, we were able to access simulation APIs and create a script through the mod’s specialized programming language. Creating the script required a comprehensive understanding of calculus, aerospace engineering, and simulation mechanics. The objective of the script is fulfilled by using a type of controls called “raw controls”. “Raw controls”, unlike “cooked controls”, enables us to manually and precisely control the rocket’s pitch, yaw, and roll instead of letting the rocket CPU decide. Orbiting the planet was achieved by launching the rocket at a specified altitude then calculating a maneuver node at the apoapsis of the orbit before executing it.

2. PROBLEM STATEMENT

2.1. Objectives/Tasks

- Learning the simulator: It was crucial to understand how the simulation works before starting to research on the mathematical formulas and on how to program the script. To do this, time was dedicated to play-testing the simulator using the keyboard and mouse. Doing so is essential for learning the mechanics of the simulator; it also helps to visualize how the script would execute when written.

- Deciding the objective: Since it is impossible to program a script that would enable the rocket to do any task autonomously, deciding the main objective of the script allows us to narrow down the scope and fully dedicate the research. After observing different scenarios in the simulation, it was decided that the main task of the script would be to enable the rocket to orbit “Kerbin”, the home planet in the simulation. It was originally supposed to orbit the “Mun”, one of the moons of “Kerbin”; however, due to time constraints it was changed to “Kerbin” only.
- Scripting: Scripting can be done in multiple ways. The simulator has a large online community that developed mods with either unique APIs or specialized programming languages. It is also possible to use C# and directly accessing the simulator libraries, however, this is slightly challenging and could complicate the scripting process as there is limited support for this method. Deciding on how to program the script is crucial as each method has its own pros and cons. In the end, it was decided the script would be programmed using kOS (“Kerbal Operating System”). This mod was ideal as the programming language is not only simple and easy to understand and learn, but also has immense online support that is useful when encountering problems. To check the validity of the script, it was tested in the simulation and then any observed mistakes were corrected.
- Scientific research: The physics in the simulation is extremely accurate to how the laws of physics works in the real world, and thus, some aerospace engineering knowledge is needed to correctly implement the script. There was a lot of information available on the information wiki and dedicated YouTube channels. Aerospace research papers, such as

the ones released from NASA, were extremely helpful to help formulate the problem and give insight on how to tackle further issues.

2.2. **Realistic Constraints**

As mentioned earlier, the entire project is about creating a script that enables the rocket to do a specified task autonomously in a simulator; in this case, it was orbiting “Kerbin”. Therefore, there were no economic, environmental, or health-safety constraints at all. Any “constraints” faced were not actual constraints, but some standards that were imposed before starting the project. The major one is not using any steering functions. This means that the group was not allowed to use any autopilot functions available in the mod API. The group was required to create its own by controlling the thrust vectors (roll, pitch, and yaw).

3. **METHODOLOGY**

For launching and orbiting missions, it is important to determine and design the orbital trajectory that the rocket must follow. In Kerbal Space Program, Gravity Turn, which is a commonly used ascent profile, relies on the natural gravitational forces acting on the rocket to achieve a gradual turn towards the desired orbit. During a gravity turn, the rocket initially launches vertically and then gradually pitches over to align with its orbital trajectory. Figure 1 plots an example of the speeds of the rocket during launch to orbit mission using gravity turn.

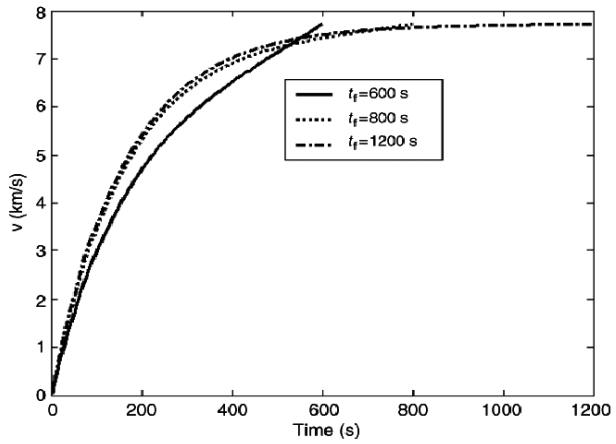

Figure1: Speed vs Time of Gravity Turn

In order to accurately and efficiently achieve the mission of orbiting Kerbin, it is essential to calculate the orbital velocity. The orbital velocity represents the velocity required for an object to maintain a stable orbit around Earth or Kerbin and an example can be seen in Figure 2.

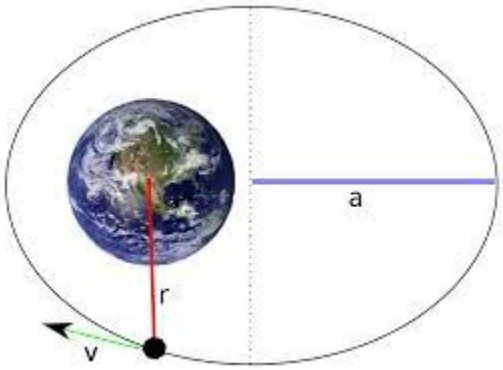

Figure 2: Spacecraft in Earth's orbit

It is determined by the balance between gravitational attraction and centrifugal force assuming no atmosphere using the vis-viva equation 1 shown below.

$$v^2 = \mu\left(\frac{2}{r} - \frac{1}{a}\right) \qquad \text{(eqn. 1)}$$

where:

μ: is the standard gravitational parameter for kerbin

r: is the altitude of the spaceship + the Kerbin's altitude

a: the semi-major axis of the orbit.

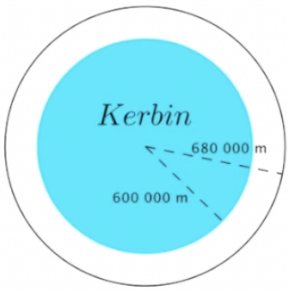

Figure 3: low Kerbin orbit

Assuming the spacecraft is orbiting at LKO, low Kerbin orbit, that has an altitude of 80 km and in a circular orbit as shown in figure 3. since r=a for a circular orbit, eqn.1 becomes:

$$v = \sqrt{\frac{\mu}{r}} \qquad \text{(eqn. 1.1)}$$

The radius of Kerbin is 600,000m as stated in the KSP wiki website, then r = 680,000 m. and the velocity found using the vis-viva equation basically tells us how fast the spacecraft will be going in a 80km orbit with zero inclination above Kerbin.

The next step after determining the orbital velocity of the space vehicle using the desired input altitude is calculating the optimal launch heading angle for the spacecraft. To address this challenge, a series of six solution steps have been outlined. First, the Inertial Azimuth, representing the compass direction at the launch site, should be calculated. From figure 4, it can be seen that the azimuth can be found using trigonometry relations for a desired inclination (n), direction of the orbit relative to a reference point, typically north, and latitude (λ), the orientation of the observer.

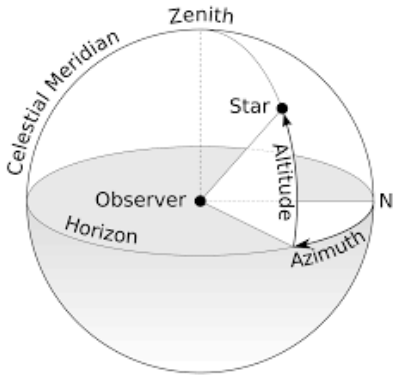

Figure 4: Azimuth

Next, the rotational velocity of the planet's surface at the latitude of the launch site is computed using equation 2 below.

$$V_{rot} = \frac{2\pi\, cos(\lambda)}{P} = 175 \text{ m/s} \qquad \text{(eqn. 2)}$$

where P is the rotational period of the planet.

Subsequently, the orbital velocity required for the desired orbit explained earlier is determined and decomposed into its component vectors.

$$V_x = V_{orbital}\, sin(Azimuth_i)$$

$$V_y = V_{orbital}\, cos(Azimuth_i)$$

Then, the East-West component V_x is added to the rotational velocity V_{rot} calculated earlier. Finally, the decomposed and adjusted vectors are recombined to ascertain the final launch heading angle from Vx and Vy using equation 3. By following these steps, one can effectively calculate the optimal direction in which the spacecraft should be launched to achieve the desired orbit efficiently.

$$\theta = arctan(\frac{V_x}{V_y}) \qquad \text{(eqn. 3)}$$

The next important calculation is the Launch windows for spacecraft which are specific periods during which it is most efficient to launch a spacecraft into orbit around Kerbin. These windows are determined by various factors, including the relative positions and inclination of Kerbin' plane and the target orbital plane, as well as the spacecraft's intended trajectory and desired orbital parameters. Optimal launch windows typically occur when the target orbit aligns with the spacecraft's launch site on the surface of Kerbin. To decide the optimal launching window, ascending nodes shown in figure 5 must be found, points where the spacecraft's orbital plane intersects the equatorial plane. Launching close to the ascending node ensures alignment between the spacecraft's trajectory and the target orbit, minimizing the need for subsequent maneuvers. Optimal windows occur when the launch site's latitude and the target orbit's inclination align, reducing the energy required for inclination changes. Choosing the time for the launch window is also critical to achieve the best launch and orbit trajectory. Time is usually calculated to be earlier than when the desired orbital plane is aligning with the launch site's plane. That is because Kerbin is rotating and if launch time is exactly when the align=ments occurs, the rocket would miss the launch window by the time it exits the atmosphere.

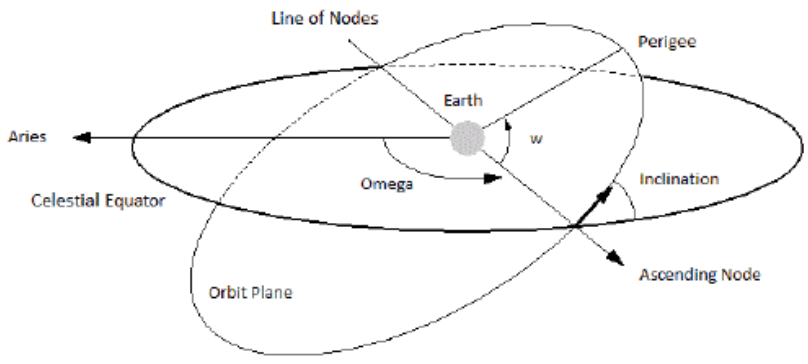

Figure 5: orbital plane inclination with respect to earth center plane

In Kerbal Space Program (KSP), delta-v (Δv) refers to the total change in velocity required for a spacecraft to perform a specific maneuver, such as reaching orbit, changing orbits, or performing interplanetary transfers. Delta-v is a fundamental concept in spaceflight planning and represents the amount of propulsive energy needed to achieve a particular trajectory or orbital change. It encompasses all velocity changes, including both acceleration and deceleration, and is typically measured in meters per second (m/s). Understanding and efficiently managing delta-v is essential for planning maneuvers.

In order to calculate the Delta-V required for a mission, first , the initial orbital velocity required for the desired Kerbin orbit using the vis-viva equation explained earlier, which takes into account the gravitational parameter of Kerbin and the altitude of the desired orbit. for initial calculations and code testing, the orbit desired altitude is set as 80 km making r = 680,000 meters. then finding orbital velocity at Kerbin space using equation 1.1 is:

$$v = \sqrt{\frac{\mu}{r}} = 2426\, m/s$$

Next is calculating the Delta-v of ascent $\overline{\Delta v}_a$ which is the change in velocity required during the ascent phase of a rocket's journey from the surface of Kerbin, to a desired orbital altitude or trajectory. It is found using the orbital velocity (v) and the Vis-Viva equation as illustrated in equation 4.

$$\overline{\Delta v}_a = \Delta v_1 + \Delta v_2 + v \qquad \text{(eqn. 4)}$$

where,

$\Delta v_1 = \sqrt{\frac{\mu}{r_1}}(\sqrt{\frac{r_2}{a}} - 1)$, Delta-V for Kerbin orbit

$$\Delta v_2 = \sqrt{\frac{\mu}{r_2}}(1 - \sqrt{\frac{r_1}{a}})$$, Delta-V for Kerbin+altitude orbit

μ is the standard gravitational parameter for kerbin

a: semi-major axis = r1+r2/2

r1: is the radius of Kerbin

r2: is the radius of orbit

Knowing that Kerbin or any planet is rotating about its axis, the planet's rotational speed must be taken into account. The planet's rotational speed affects the rocket's launch because it adds to or subtracts from the rocket's initial velocity, depending on whether the launch is in the same direction as or opposite to the planet's rotation. When launching in the same direction as the planet's rotation, the rocket benefits from an additional velocity boost, requiring less delta-v to reach orbit. Conversely, launching against the planet's rotation necessitates overcoming this rotational velocity, increasing the total delta-v required for the ascent. Thus, accounting for the planet's rotational speed is crucial for optimizing launch trajectories. the kerbin's rotating speed $\overline{v}_r$ is 175 m/s according to KSP wiki. Lastly, For the purpose of setting an accurate budget for the required velocity to reach orbit, it is crucial to account for atmospheric drag during ascent. due to the complexity of calculating the atmospheric drag coefficient. The Delta-V map in figure 6, is used to find the atmospheric drag. For example, if our mission is low orbit then Delta-V is 3400 m/s from the map. The Delta-V for atmospheric drag would be the sum of all the Delta-V values found subtracted from the Delta-V in the map.

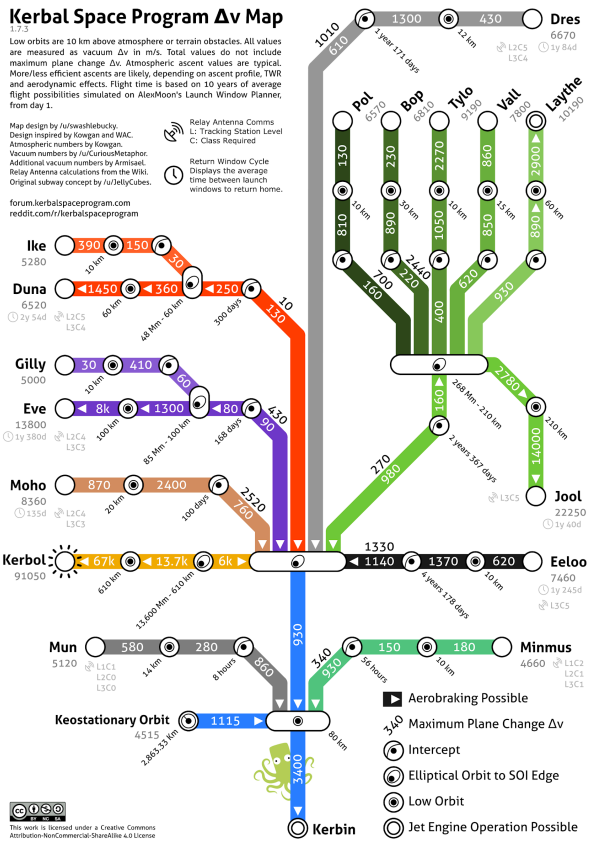

Figure 6: Kerbal Space Program Delta-V map

In conclusion, the Δv required for reaching orbit is found from equation 5.

$$\overline{\Delta v} = \overline{\Delta v}_a - \overline{v}_r + \overline{\Delta v}_d \qquad \text{(eqn. 5)}$$

where:

$\overline{\Delta v}_a$ is the Δv for the cost of ascent.

$\overline{v}_r$ is the kerbin's rotating speed which is 175 m/s.

$\overline{\Delta v}_d$ is the velocity cost to overcome aerodynamic drag force.

Until now, all calculations are explained with respect to magnitudes only, but velocities are vectors that are measured by direction also. The program is developed to also input the desired inclination of the orbits. Uses trigonometry and cosine laws to find the directions

required of each delta-V , and use it to accurately control and drive the rocket to the desired orbit from the launchpad on Kerbin.

Moving on to the code algorithm, the program passes the desired orbit altitude and inclination as a parameter. The code flow is very similar to the theoretical description mentioned earlier on how to reach orbit. In addition to mathematical equations and data extraction, the program,

4. RESULTS & DISCUSSION

4.1 Results using autopilot

During program development using KOS and python the code used autopilot for launching and orbiting. This is essential for data extraction and analysis to obtain the optimal trajectory for the mission. Testing the code initially, it was challenging to get the rocket to exit the Kerbin's atmosphere because the engine would explode. Mods to determine Delta-V, and controlling functions were added to the program in an attempt to solve the issue. An essential mod was used to control the Yaw, Pitch and roll angles of the rocket that control the direction of the heading. After editing the code and many tests, the code was able to launch and insert the rocket into orbit successfully using autopilot and KOS mods and functions provided by the KOS documentation. All codes were tested for a circular low Kerbin orbit with an altitude of 80 km above Kerbin and with zero inclination initially for error tracking. The results of the successful mission using one of the scripts are shown in Figures 7-9.

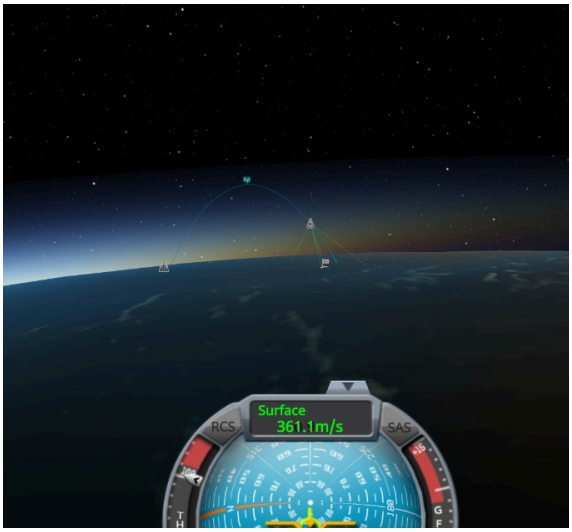

Figure 7: Spaceship leaving Kerbin's atmosphere

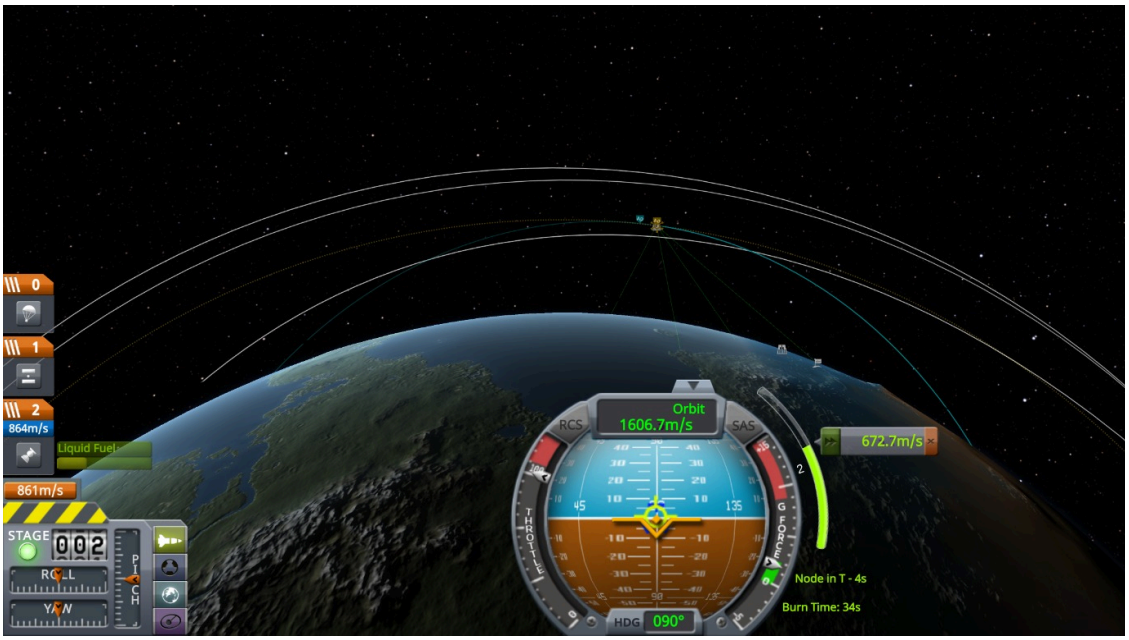

Figure 8: Rocket after reaching apoapsis

Figure 9: Rocket successfully orbiting Kerbin

4.1 Results using calculations

Changing the code from autopilot and mod function to equation-based is very challenging due to the unfamiliarity of the KOS environment. It was challenging to achieve and required many tests and research to fully understand the syntax and the format of which formulas can be written in KOS. A code fully dependent on formulas is nearly impossible to implement for this mission. KOS may have its own limitations in terms of processing power and execution speed, which can further complicate the implementation of complex equation-based solutions. Since the code is required to find each value at each step second, which is 1 msec, calculating every value is necessary, and some mods are used. For testing the codes, different altitudes were set as inputs

and results from different runs were observed. Results from testing the code are presented below. Figures 10-12, display the results for a desired low Kerbin orbit, as seen

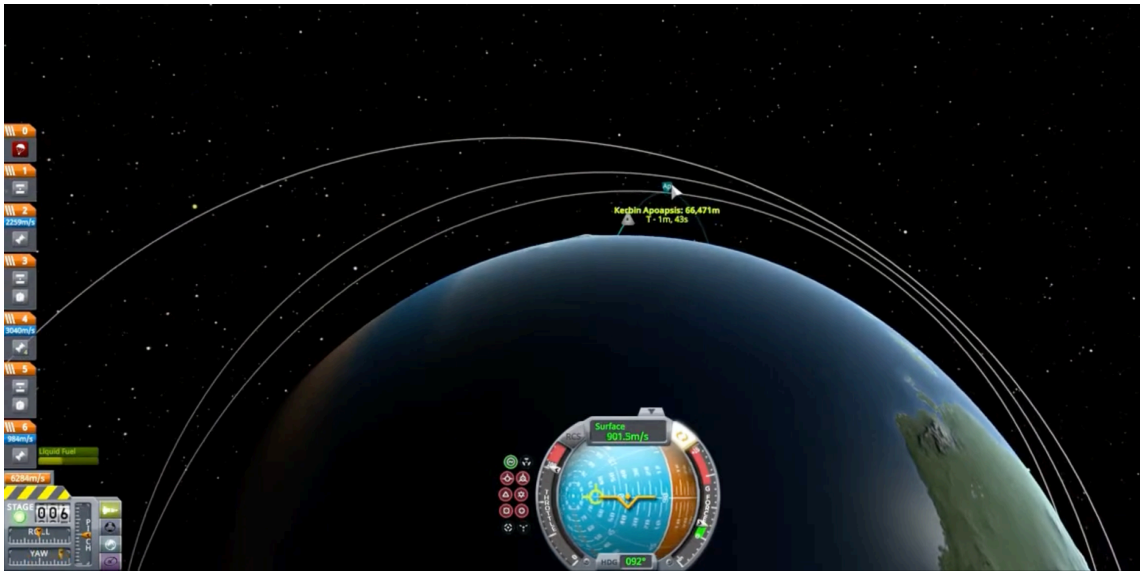

Figure 10: Spaceship after atmosphere exit

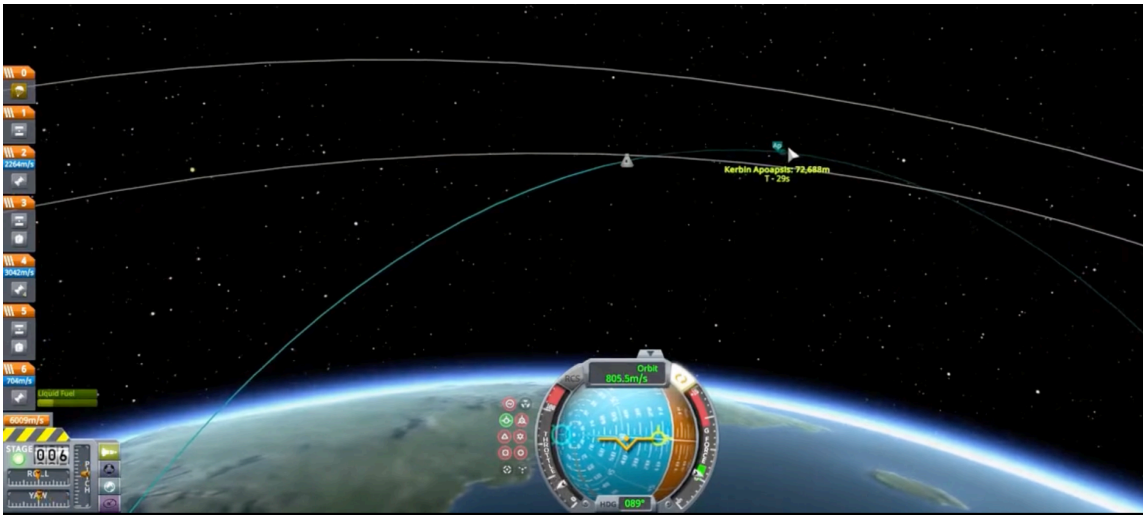

Figure 11: Spaceship emerging Apoapsis

Figure 12: Rocket in orbit

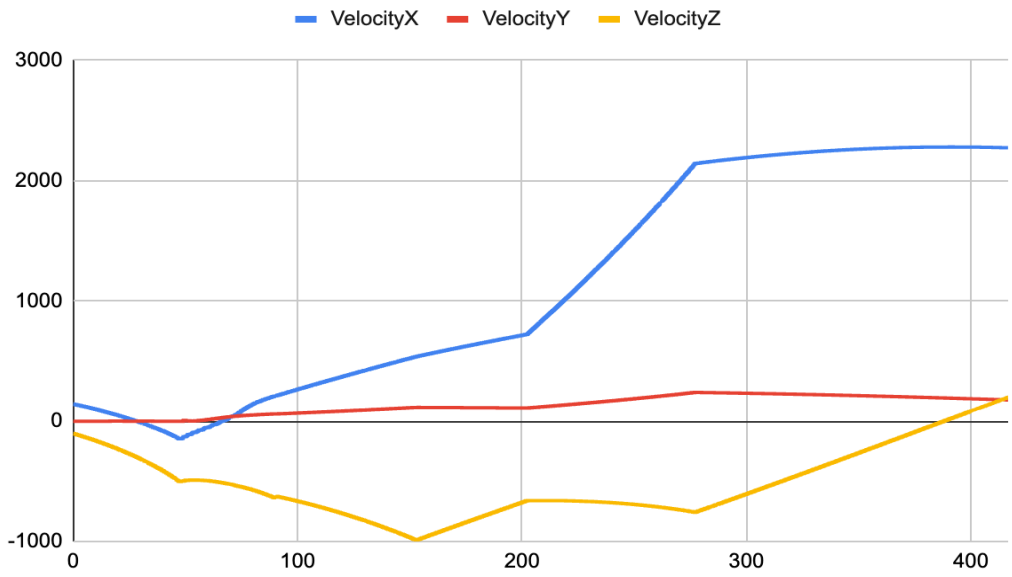

Figure 13: rocket velocities [m/s] in x,y, and z directions vs time [sec].

Figure 13 Displays the plots graphed using the values of thinned from the simulator after a successful, orbiting mission. The values plotted are the velocities in X, Y and Z directions versus time. As seen from the figure, the velocity is less than the delta velocity calculated meaning the rocket successfully reached orbit of Kerbin without consuming more energy than the expected. And it was able to launch and reach orbit with a velocity less than the maximum velocity predicted for the rocket to use to complete the same mission. As the map indicates in figure 6, such a mission usually requires the rocket to have a velocity of 3400 m/s.

5. IMPACT

The project's success in creating autonomous landing control systems in the context of the Kerbal Space Program (KSP) has significant scientific implications since it advances the understanding of dynamic modeling in aerospace engineering and complicated control algorithms. This improves the field of control systems and paves the way for more accurate and effective landing techniques for spacecraft, which could completely change the way that space exploration is conducted. In terms of technology, real-time interaction between software simulations and physical systems has advanced significantly with the integration of simulated controllers. Applications for this technology are not limited to space exploration; they also include robotics, autonomous vehicles, and industrial automation.

Additionally, the project promotes socio-economic effects by giving researchers and students valuable practical experience, developing their abilities, and encouraging more creativity in multidisciplinary domains. This assists entrepreneurship and possible commercialization prospects in connected businesses, which in turn propels economic growth and technological advancement, in addition to producing qualified employees.

6. ETHICAL ISSUES

The usage of jet fuel in the rockets may be the only point that can be criticized as harmful to the environment where the aviation industry already contributes to climate change with the burning of fossil fuels releasing CO2 in the atmosphere. There are no ethical issues related to this project due to it being a controller based simulation project.

7. PROJECT MANAGEMENT

Initially, it was decided that the main objective of the script was to enable the rocket to autonomously orbit the “Mun”. When trying this manually in the simulator, it proved to be a bit challenging due to multiple reasons. In order to orbit the “Mun”, the rocket must first orbit “Kerbin”. Orbiting “Kerbin” is not the challenge, however. After orbiting “Kerbin”, a maneuver node must be created so that the rocket can go on an escape trajectory to leave “Kerbin’s” sphere of influence (SOI). Leaving “Kerbin’s” sphere of influence is required, otherwise its gravity will keep pulling the rocket towards it. After leaving the sphere of influence, another maneuver node must be created to enter the “Mun’s” sphere of influence. Finally, after entering the “Mun’s” sphere of influence, the creation of the last maneuver node enables the rocket to form an orbit around it. To add to that, all of the steps must be made in a very small time frame with no way to correct a mistake. As described, orbiting the “Mun” is a very meticulous and unforgiving procedure, so the main task has been changed to simply orbiting “Kerbin”.

Another reason the main task was changed was due to time constraints. Unfortunately, it took quite some time to create a script that simply orbits “Kerbin” without enabling a user

parameter and without using steering functions available in kOS. The lack of time was caused as a result of the difficult subject matter, advanced mathematics.

In terms of project management, strong communication cannot be stressed enough. Most of the shortcomings and plan changes happened due to a lack of communication/clear work assignment between team members. Another vital element is proper time management. Throughout the semester, there has been a tremendous amount of deadlines imposed from multiple courses. The large amount of course work made it difficult to properly and efficiently dedicate an immense amount of time solely for the project. Nonetheless, the group made sure to dedicate enough time in order to achieve the minimum requirements of the project. Had there been more time, a more complex and sophisticated script would have been made.

8. CONCLUSION AND FUTURE WORK

The project focused on developing an autonomous flight script within the Kerbal Space Program (KSP) simulator, leveraging the kOS (Kerbal Operating System) mod. This script was tailored to establish a circular orbit around "Kerbin," requiring intricate mathematical calculations and algorithmic design. In addition to accomplishing this primary objective, the project offered valuable insights into simulation mechanics, aerospace engineering principles, and algorithm development, which can be applied to future endeavors in similar domains. Despite facing challenges like the script's complexity and time constraints, the project serves as a successful demonstration of creating a script for automated rocket control in the KSP simulator. Looking ahead, potential avenues for advancement include refining the script's functionality, broadening its capabilities to handle more complex tasks such as planetary landings, and investigating alternative mods or tools to enhance its adaptability across diverse platforms and

environments. Furthermore, integrating the software with other simulation environments beyond KSP holds promise for extended testing and validation, potentially contributing to various fields like planetary exploration and satellite deployment simulations.

In future work, an additional goal is to extend the functionality of the script to enable the rocket to transition from a low orbit around Kerbin to a different orbit, such as a geostationary orbit. This expansion would necessitate further refinement of the script's algorithms and calculations to accommodate the complexities involved in altering orbital parameters. Successfully achieving this objective would significantly enhance the script's versatility and applicability, allowing for more sophisticated maneuvers and missions within the KSP simulator. Additionally, it would provide valuable insights into orbital mechanics and spacecraft navigation, further advancing the project's contributions to aerospace engineering and simulation technology.

9. **APPENDIX**

One of the codes for scripting:

```
RUNONCEPATH("0:/lib/maneuverFunctions.ks").


PARAMETER userAlt is 80000.

SET direction to 90.

SET roll to 360 - 90.

SET pitch to (90 *12000)/(ALTITUDE + 12000).

SET progradeBool to FALSE.


LOCK THROTTLE to 1.0.

//LOCK STEERING to UP.


STAGE.


SET MYSTEER TO HEADING(90, 90).

LOCK STEERING TO MYSTEER.


WHEN SHIP:MAXTHRUST = 0 THEN

{

    PRINT "Staging".

    STAGE.

    PRESERVE.

}


until APOAPSIS >= userAlt

{
```

```
    WAIT UNTIL SHIP:ALTITUDE >= 10200.


    SET MYSTEER TO HEADING(direction, (90 - userAlt/1000)).
    WAIT 2.
    LOCK THROTTLE to 1.0.
    PRINT "heading to target".


    UNTIL(APOAPSIS > userAlt)
    {
        IF (ALTITUDE > 20000) AND NOT progradeBool
        {
            SET MYSTEER TO PROGRADE.
            SET progradeBool to TRUE.
        }


    }
}
LOCK THROTTLE to 0.0.
circAt().
PRINT "Adding node".
exeMan().
PRINT "Executing node".
```

Code for creating maneuver node and executing it:

```
GLOBAL FUNCTION circAt {
  PARAMETER loc IS "apo".
  IF loc = "apo" {
    SET futureVelocity to SQRT(VELOCITY:ORBIT:MAG^2-2*BODY:MU*(1/(BODY:RADIUS+ALTITUDE) - 1/(BODY:RADIUS+ORBIT:APOAPSIS))).
    SET circVelocity to SQRT(BODY:MU/(ORBIT:APOAPSIS+BODY:RADIUS)).
    SET newNode to NODE(TIME:SECONDS+ETA:APOAPSIS, 0, 0, circVelocity-futureVelocity).
  } ELSE {
    SET futureVelocity to SQRT(VELOCITY:ORBIT:MAG^2-2*BODY:MU*(1/(BODY:RADIUS+ALTITUDE) - 1/(BODY:RADIUS+ORBIT:PERIAPSIS))).
    SET circVelocity to SQRT(BODY:MU/(ORBIT:PERIAPSIS+BODY:RADIUS)).
    SET newNode to NODE(TIME:SECONDS+ETA:PERIAPSIS, 0, 0, circVelocity-futureVelocity).
  }
  ADD newNode.
}
```

```
GLOBAL FUNCTION exeMan {
    IF HASNODE {
        CLEARSCREEN.
        SAS OFF.

            // holds data regarding maneuver node
        SET mNode to NEXTNODE.
            // makes sure there are active engines
        IF (currentISP() < 0) {
            PRINT " ".
            PRINT "NO ACTIVE ENGINES.".
            WAIT UNTIL (currentISP() > 0).
        }
            // start time of burn
        SET startTime to calculateStartTime().
            // start direction of burn
        SET startVector to mNode:BURNVECTOR.
        lockSteering().
        startBurn().
        endBurn().
            // if no node in path, program ends
    } ELSE {
        PRINT " ".
        PRINT "No node in flight path.".
        PRINT " ".
    }
}
```

```
FUNCTION currentISP {
    LIST ENGINES in engineList.
    SET sumOne to 0.
    SET sumTwo to 0.
    FOR eng in engineList {
        IF eng:IGNITION {
            SET sumOne to sumOne + eng:AVAILABLETHRUST.
            SET sumTwo to sumTwo + eng:AVAILABLETHRUST/eng:ISP.
        }
    }
    IF (sumTwo > 0) {
        RETURN sumOne / sumTwo.
        // returns -1 if no active engines
    } ELSE {
        RETURN -1.
    }
}
```

```
FUNCTION calculateStartTime {
    RETURN TIME:SECONDS + mNode:ETA - burnTime() / 2.
}
```

```
FUNCTION startBurn {
    SET maxAcc to SHIP:MAXTHRUST / SHIP:MASS.
    SET throttleSet to 0.
    LOCK THROTTLE to throttleSet.

        // 3 second countdown
    WAIT UNTIL TIME:SECONDS > (startTime-3).
    PRINT " ".
    PRINT "Starting burn in ...".
    PRINT "3".
    WAIT 1.
    PRINT "2".
    WAIT 1.
    PRINT "1".
    WAIT 1.
    PRINT "Locking throttle to full.".

    SET throttleSet to MIN(mNode:DELTAV:MAG / maxAcc, 1).
}
```

```
FUNCTION endBurn {
    WAIT UNTIL maneuverComplete().
    PRINT " ".
    PRINT "Burn Complete.".
    PRINT " ".
    LOCK THROTTLE to 0.
    UNLOCK STEERING.
    SAS ON.
    WAIT 5.
}
```

Mod to extract data:

```
using System.Collections.Generic;
using UnityEngine;
using KSP;
using System;
using System.IO;
using System.Diagnostics;

namespace KSP_XYZ
{
    [KSPAddon(KSPAddon.Startup.Flight, false)]
    0 references
    public class XYZ_Coor : MonoBehaviour
    {

        0 references
        private void Update()
        {
            Vessel currentVessel = FlightGlobals.ActiveVessel;

            if (currentVessel != null)
            {
                // Get the vessel's velocity vector
                Vector3d velocityVector = currentVessel.obt_velocity;

                // Extract individual velocity components
                double velocityX = velocityVector.x;
                double velocityY = velocityVector.y;
                double velocityZ = velocityVector.z;

                // Get the time in seconds
                double currentMissionTime = currentVessel.missionTime;

                // Specify the path for the CSV file
                //string filePath = "D:\\Games\\Kerbal.Space.Program.v1.12.5.3190.Incl.ALL.DLC\\Kerbal.Space.Program.v1.12.5.3190.Incl.ALL.DLC\\GameData\\DataExport\\graphs\\filght_coordinates";
                //string filePath = Path.Combine("D:\\Games\\Kerbal.Space.Program.v1.12.5.3190.Incl.ALL.DLC\\Kerbal.Space.Program.v1.12.5.3190.Incl.ALL.DLC\\GameData\\DataExport\\graphs\\", fileName);

                string currentDateTime = DateTime.Now.ToString("yyyyMMdd");

                string fileName = $"xyz_coordinates_{currentDateTime}.csv";
                string relativePath = Path.Combine("GameData", "DataExport", "graphs", fileName);
                string filePath = Path.Combine(KSPUtil.ApplicationRootPath, relativePath);

                // Check if the file exists, if not, create it and write the header
                if (!File.Exists(filePath))
                {
                    using (StreamWriter writer = new StreamWriter(filePath, true))
                    {
                        // Write the header
                        writer.WriteLine("VelocityX,VelocityY,VelocityZ,MissionTimeMillis");
                    }
                }

                // Append the values to the CSV file
                using (StreamWriter writer = new StreamWriter(filePath, true))
                {
                    // Write the data
                    writer.WriteLine($"{velocityX},{velocityY},{velocityZ},{currentMissionTime}");
                }

                // Print a message or log that the data has been written
                //Debug.Log("Data written to CSV file.");
            }
        }

    }
}
```

10. **REFERENCES**

Aben, Mike. “KOS-Scripts.” *GitHub*, github.com/MikeAben64/kOS-Scripts/blob/main/KSRSS/lib/launchFunctions.ks.

Adetoro, Moshood, et al. “Application of Numerical Simulation of Nonlinear Models to Three Stages Micro Satellite Launch Vehicles (MSLVs) Trajectory.” *Researchgate*, www.researchgate.net/publication/310241038_Application_of_Numerical_Simulation_of_Nonlinear_models_to_Three_Stages_Micro_Satellite_Launch_Vehicles_MSLVs_Trajectory. Accessed

“Kerbin.” *Kerbin - Kerbal Space Program Wiki*, wiki.kerbalspaceprogram.com/wiki/Kerbin.

Orbital Mechanics, faculty.fiu.edu/~vanhamme/ast3213/orbits.pdf.

“ORBITAL MECHANICS.” *Basics of Space Flight: Orbital Mechanics*, www.braeunig.us/space/orbmech.htm#position.