(Preprint) AAS 23-368

CISLUNAR CR3BP PERIODIC ORBIT IDENTIFICATION IN VIRTUAL REALITY

Dhathri H. Somavarapu[*], Eirik Mulder[†], and Davide Guzzetti[‡]

This manuscript presents two efficient and human-user-friendly techniques to identify cislunar circular restricted three-body (CR3BP) periodic orbits in virtual reality. While numerous powerful numerical algorithms are available to correct initial guesses for periodic identification problems, the mechanisms to derive good initial guesses are not universal. The quality of the initial guess impacts the accuracy and efficiency of numerical correction algorithms. Our work is an attempt to provide an initial guess construction tool for arbitrary periodic orbit identification problems in an autonomous nonlinear dynamical system. Our work uses cislunar CR3BP dynamics to prove the concept of immersive and interactive initial guess construction. We simulate the cislunar CR3BP on a virtual reality (VR) platform, provide a mechanism to manipulate the position and velocity for the initial guesses of the desired periodic orbits on the VR platform and implement a collocation-based correction algorithm to identify the desired periodic orbits from the initial guesses constructed in VR. The mechanisms we developed proved to be quite powerful at quickly and efficiently identifying desired periodic orbits in terms of the accuracy of the solutions and the simplicity of the discovery process for periodic orbits. To test our proof-of-concept, we attempted to identify non-trivial periodic orbits without any initial seed for position and velocity states and succeeded in those attempts. From these experiments, we conclude that our VR-based initial guess construction mechanisms provide the astrodynamics community with a new way to explore the complex orbital motion in the cislunar CR3BP space.

1 INTRODUCTION

The circular restricted three-body model (CR3BP) does not have a known closed-form analytical solution. Solutions in CR3BP systems are typically identified with numerical nonlinear correction algorithms that are gradient-based. Gradient-based algorithms are highly susceptible to the quality of the initial guess. Bad initial guesses typically yield invalid solutions. Hence there is a desire to find better initial-guess construction mechanisms. To help ease the difficulty of constructing a good initial guess, we propose two virtual-reality (VR) based initial-guess construction methods in this work. These new methods allow a human user designing periodic orbits to experiment with initial guesses rapidly and accurately.

In our previous works,[1,2] we explored the use of guided and arbitrary drawings to facilitate the construction of initial guesses for CR3BP periodic orbits. In such methods, a human user drawing on the VR platform renders the initial guess of a periodic orbit. The user drawing collected on

[*]Ph.D. Candidate, Department of Aerospace Engineering, Auburn University, 211 Davis Hall, Auburn, AL 36849. (equal contribution)

[†]Undergraduate Student, Department of Aerospace Engineering, Auburn University, 211 Davis Hall, Auburn, AL 36849. (equal contribution)

[‡]Assistant Professor, Department of Aerospace Engineering, Auburn University, 211 Davis Hall, Auburn, AL 36849.

the VR platform only supplies an initial guess for the position state component of the orbit. The missing velocity and time-of-flight information is derived by enforcing an orbital energy constraint and applying numerical differentiation (i.e., forward and central finite difference approximations) to the position state components. This method does not provide the user any direct intuition about the underlying CR3BP dynamics and how motion in that model might be correlated to the drawing. Constructing an initial guess in this method is sometimes impossible because the user-drawing may not exist at the prescribed orbital energy level. To overcome such a lack of nearly-instantaneous feedback from the CR3BP dynamical motion, our new method enables the user to construct an initial guess by manipulating the position and velocity of one or more control points and observing the trajectory arcs emanating from the control points in real-time. These arcs are generated by numerically integrating CR3BP dynamics, starting with each control point's initial position and velocity. As such, the initial guess is composed of fully valid CR3BP position, velocity, and time points, thus always producing valid trajectory arcs (i.e., ones that obey the CR3BP dynamics). There is precedent for the kind of approach we take in the current work. Mathur[3] introduced the concept of control point manipulation for trajectory design in GMAT and Copernicus. In Mathur's work, the 3D desktop widget extensions to GMAT and Copernicus are enabled, allowing users to design their own 3D widgets to choose control points in the initial guess for a trajectory. Ramaswamy[4] presented a similar concept of control point manipulation for the interplanetary trajectory design in VR from her collaborative work at JPL. We also found in our testing that our method is very fast at constructing the initial guess and producing a corresponding periodic orbit from the initial guess.

The cislunar CR3BP model for an object's orbital motion under the Earth's and the Moon's gravitational influence is listed in Equations (1).

$$\begin{aligned} \ddot{x} - 2n\dot{y} - n^2 x &= -\frac{(1-\mu)(x+\mu)}{d^3} - \frac{\mu(x-1+\mu)}{r^3} \\ \ddot{y} + 2n\dot{x} - n^2 y &= -\frac{(1-\mu)y}{d^3} - \frac{\mu y}{r^3} \\ \ddot{z} &= -\frac{(1-\mu)z}{d^3} - \frac{\mu z}{r^3} \end{aligned} \tag{1}$$

In Equations (1), μ represents the mass parameter of the system, n represents the mean-motion of the Moon around the Earth in the idealized circular motion (it is always equal to 1), and d and r respectively represent the distances of the third body (spacecraft) with respect to the Earth and Moon. This model allows for identifying predictable structures in the complex three-body motion. One type of these predictable structures is a periodic orbit. Periodic orbits in the CR3BP model are generally non-Keplerian. The shape of periodic orbits in this model is not limited to forms traceable to well-defined conic sections as in the Keplerian motion of a two-body system. With this work, our first objective is to realize an immersive and interactive environment for periodic orbit discovery when a collocation algorithm corrects initial guesses. We first assume the user can only manipulate the initial point along the initial guess in our VR application; we label this approach as single-point manipulation. In a later and alternative iteration of the VR platform, we allow users to influence the position and velocity at numerous control points along the prospective periodic orbit. We label this other approach as multiple-point manipulation. Figure 1 visually represents the concepts of single-point and multiple-point manipulation. Our objectives include ensuring that each application's human interface and user interaction are efficient and user-friendly. To achieve that, the execution of a numerical correction algorithm for identifying periodic orbits from the initial

guesses should be fast. Our goal is for the execution time not to exceed 90 seconds – typically falling under 30 seconds for smaller orbits (less patch points), between 30 to 60 seconds for medium-sized orbits, and capping at 90 seconds for larger orbits (more patch points). The rest of the document describes the framework we developed to achieve these objectives and the initial verification of the framework's capabilities.

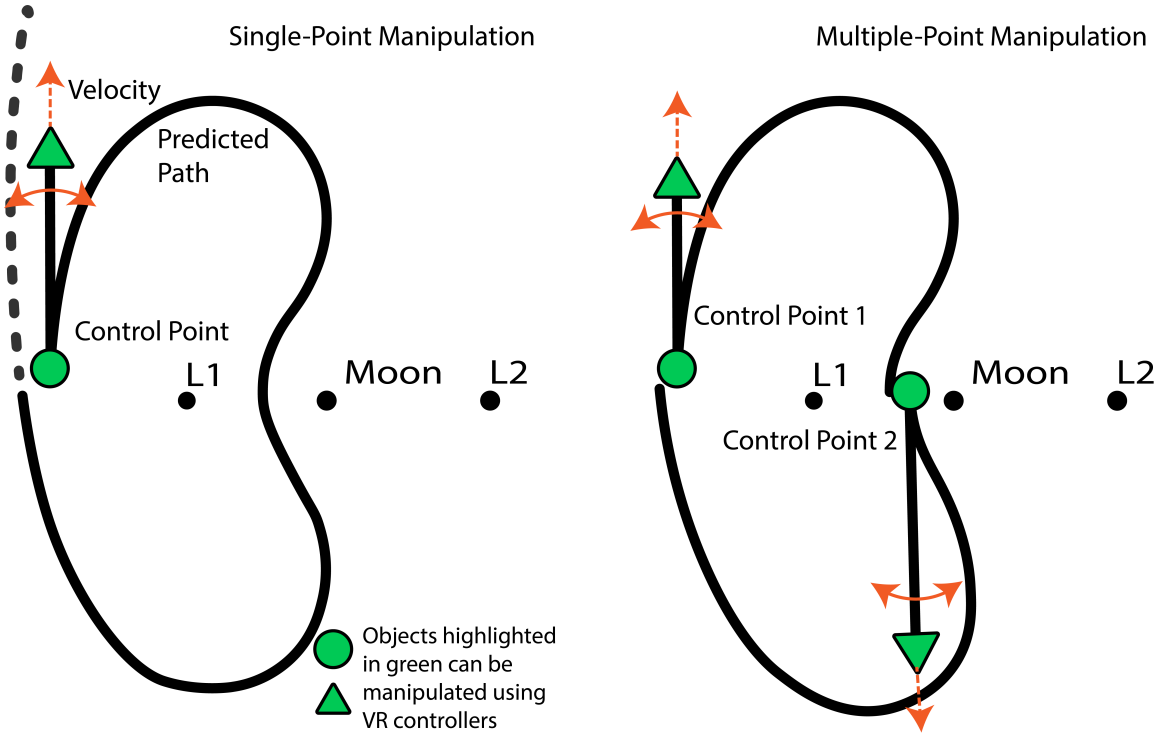


Figure 1: Single-Point and Multiple-Point Manipulation Approaches

2 VR-BASED FRAMEWORKS FOR PERIODIC ORBIT IDENTIFICATION

We designed, developed, and tested two VR-based methods for an easier and faster periodic orbit identification in the cislunar CR3BP model. In these methods, we simulate the cislunar CR3BP system in real 3D on a VR platform, the elements of which include the Earth, the Moon, and the five Lagrange points. We then connect a numerical correction algorithm to this system to identify periodic orbits from the initial guesses constructed on the VR platform application. The initial guesses are constructed by manipulating the position and velocity state components at locations of interest on the VR platform application using VR controllers. The VR applications are implemented as C# language applications on a game development engine called Unity3D. Unity provides a tried and tested system for developing virtual-reality applications, user-interface design, and complex interactive systems with low-performance overhead. Unity provides the rendering framework and allows development using the C# language to write the components required for a 3D scene. To propagate the CR3BP trajectory emanating from the position and velocity chosen by the user in the VR application, we employ a fixed-step Runge-Kutta-4 (RK4) integration of the CR3BP equations-of-motion (1) inside the C# application. Using the numerical correction algorithm, the initial guess constructed from the RK4 integration is then corrected to identify a true periodic orbit. The numerical correction algorithm is implemented in Python using the CasADi[5] package. The CasADi

package supplies the required derivatives in the nonlinear programming problem (NLP) using the automatic differentiation technique. This work develops two VR applications for single-point and multiple-point manipulation approaches to the initial-guess construction: the following sections lay out the user interface, and system architecture for these two VR applications.

2.1 Modality-based User Interface Design

The single-point manipulation VR user interface's design explores the modality concept. There are four modes that the application can be in at any point: Position Manipulation, Velocity Manipulation, Menu Interaction, and Optimization. The primary mode of scene control is implemented using a global menu, which can be opened using the dedicated menu button on the VR controller. The menu appears in front of the user at a distance, letting the user use rays attached to the controllers to click menu buttons and manipulate controls. The user can switch between manipulating the guess's initial position and the initial velocity via a button in the menu. Menu widgets also allow the user to change the height of the environment, select the number of patch points, change the prediction time-of-flight for the optimizer to use, and run the optimizer on the initial guess. While the menu is closed, depending on the current mode, the user can place the initial position of the guess at any location in the Earth-Moon system (Figure 2a) using the trigger on the VR controller, or select any initial velocity respective to the initial position. The user can control the velocity vector's magnitude and direction (Figure 2b), which allows for intuitive manipulation of energy level.

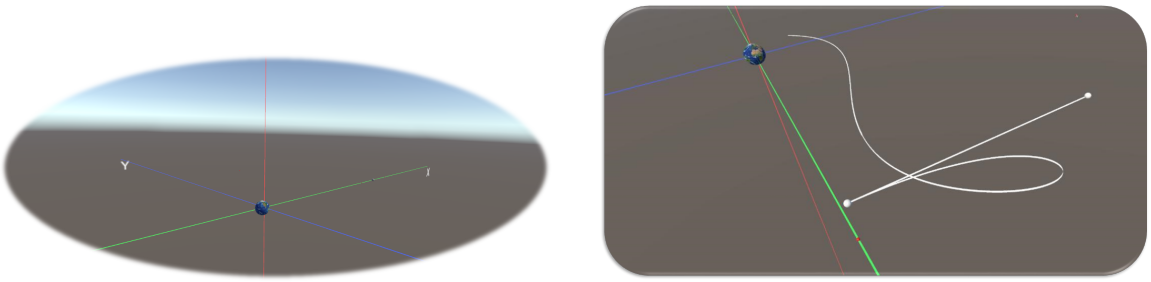
Table 1 shows the manipulations available to the user in the different implemented modes. First, the user selects a position to start the initial guess, and then they open the menu to change to velocity manipulation. A common pattern is to switch between making small changes to both position and velocity and changing the time of flight to achieve the desired orbit guess. Once the user is satisfied, they can open the menu, select a number of patch points (where more patch points often result in a higher resolution solution, but longer optimization time), and then presses the button to invoke the optimizer. Several seconds later, once the background Python optimizer finishes, the solution orbit will be displayed in magenta, and the parameters for the solution will be displayed in a menu panel.

Mode	Position Manipulation	Velocity Manipulation	Menu Active	Optimization
Set Initial Position	YES	NO	NO	NO
Set Initial Velocity	NO	YES	NO	NO
Adjust Patch Points (menu button)	NO	NO	YES	NO
Adjust TOF (menu button)	NO	NO	YES	NO
Run Optimizer (menu button)	NO	NO	YES	NO
Adjust Scene Settings (menu)	NO	NO	YES	NO

Table 1: The manipulations available in each scene mode

In every frame, the user is changing the initial position and velocity, and a prediction of the trajectory from the initial guess is generated using a Runge-Kutta-4 integrator on the CR3BP dynamics. The inputs to the integrator are the current position and velocity (state 0), and the time of flight selected in the menu, which controls the duration of integration. The propagated trajectory gives

an intuitive visual for how the final solution will look, and the real-time updating of the prediction allows the user to make small adjustments and rapidly refine the initial guess. Once the user has selected a desired guess for initial position and velocity, they can elect to start the optimizer. When this happens, the unity program saves the propagated initial guess to the file system, then initializes the optimizer in the background with the file as an input. The optimizer is developed in Python and runs simultaneously with the VR app when spawned. Once the optimizer finishes, it saves its output to files, and exits, allowing the virtual-reality application to load and display the solution to the user. Once the optimizer finishes and the solution is displayed (usually in a matter of seconds), the user can archive the solution for future use, or adjust the initial conditions and re-run the optimizer. This allows for a highly flexible iteration process, where the user can repeatedly adjust until a desired solution is found. Figure 3 showcases a visual flowchart for all the steps previously described for the single-point manipulation method in this section.



(a) A Depiction of the Earth-Moon CR3BP System in Single-Point Manipulation VR Application

(b) A Depiction of the Manipulation of the Initial Velocity Vector

Figure 2: Single-Point Manipulation Method VR Application

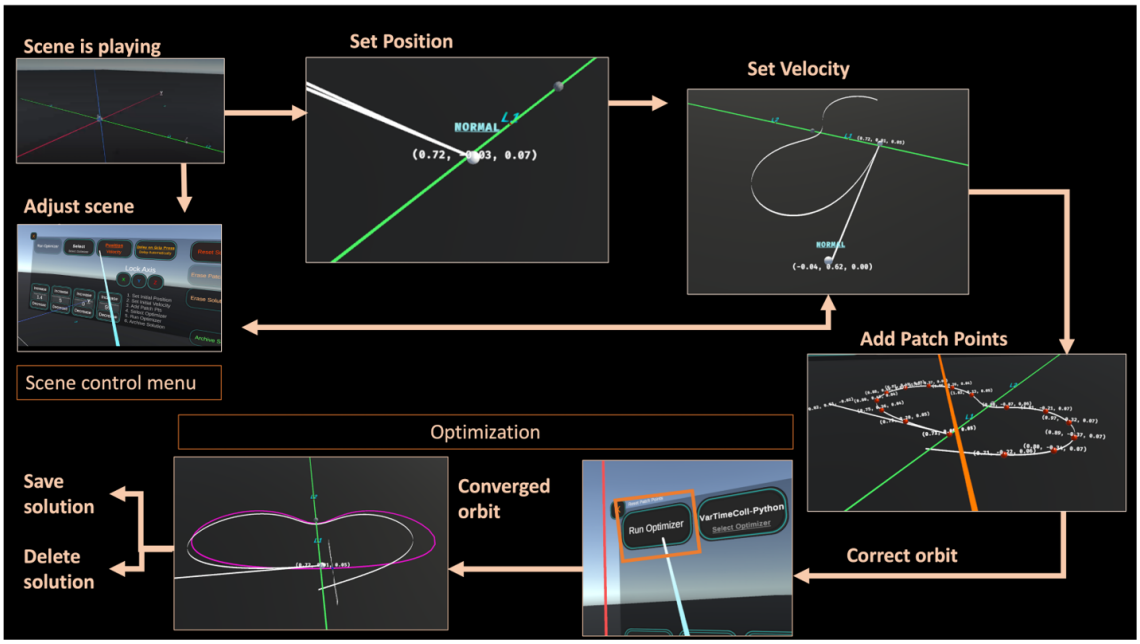


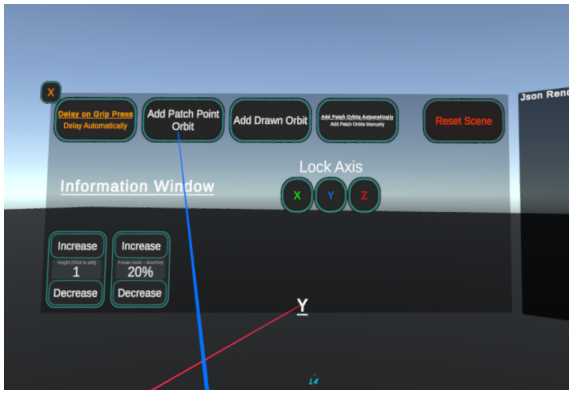
Figure 3: A Visual Flow-Chart of the Single-Point Manipulation Method VR Application

2.2 Object-based User Interface Design

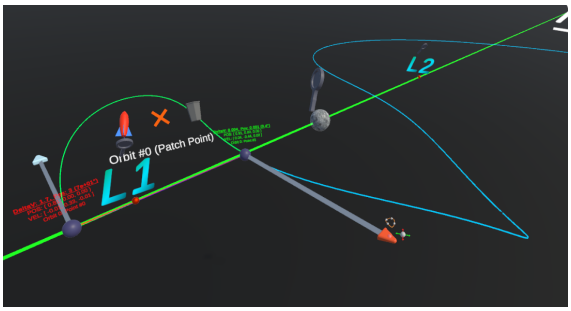
The multiple-point manipulation VR application explored a design centered around creating and manipulating different objects in the CR3BP system. These objects could include an initial-guess orbit made up of control points, and the corresponding CR3BP path. Each control point consists of a position in the CR3BP system and a velocity. Each control point implements a Runge-Kutta-4 real-time propagator, which predicts the motion of a spacecraft in the system with the selected position and velocity. The user must add a new orbit from the global scene menu to begin the orbit-generation process. This step is depicted in Figure 4a. Then, the user can add a sequence of patch points, each with a position and velocity. This step is depicted in Figure 4b. Once the user selects all the control points, shown in Figure 5a, they can click the optimizer button with the controller (attached to the set of control points). After several seconds, when the optimizer finishes, a solution orbit will be displayed in the scene. This is shown in Figure 5b. The primary objective of the application is to string together a sequence of control points and predicted trajectories to create a precise desired orbit. When control points are combined into an orbit, each control point propagates a predicted path until it reaches the next control point in the sequence. A typical workflow consists of making an initial guess at position and velocity, similar to single-point manipulation, and then inserting multiple control points along the trajectory to make small adjustments to the path. This allows for generating more complex orbits by reducing the precision needed at each point. Every time the user places a control point along the trajectory, it cuts off the previous trajectory at the point and inherits the final velocity of the previously predicted path. This means the user can make small tweaks to the velocity, starting from the previous prediction. Once the user has connected a sequence of position and velocity guesses into a guessed orbit, they can initiate the optimizer, which will attempt to correct the sequence of points into a dynamically-correct CR3BP periodic orbit. In the multiple-point manipulation application, the user can make multiple control-point orbits simultaneously, and each control point belongs to a single orbit. Orbits can be adjusted and optimized individually, which lets the user generate multiple orbits and view them in the same environment. Recall Figure 1, which shows the contrast between the single-point and multiple-point manipulation solution methods. For single-point manipulation, the user manipulates a single initial position and velocity to optimize the prediction for the desired orbit, then runs the optimizer. For multiple-point manipulation, the user places a position/velocity point and then can place additional control points along the line to correct deviations. Each additional control point terminates the prediction of the point before it and starts a new prediction starting at that position and velocity. The single-point manipulation concept is shown on the diagram's left side, and the multiple-point manipulation concept is shown on the right side, with two control points placed. In the multiple-point manipulation approach to initial-guess construction, the human user is prompted to manipulate the position and velocity for multiple control points along a path to formulate an initial guess. Figure 4b shows a snapshot of the velocity manipulation.

Captures of the scene in Figures 4 and 5 show the steps a user takes to generate an initial guess in the multiple-point manipulation VR application. Figure 4a shows the user adding a new orbit object to the scene, the first step in solution generation. Figure 4b demonstrates the manipulation of a control point's initial position and velocity. The previous control point in the orbit is also displayed, and the propagation can be seen to follow CR3BP dynamics before terminating at the position of the new control point. Figure 5a shows the completed set of control points before the user runs the optimizer, and Figure 5b shows the final result once the optimizer has been run. This final result is generated by the Python-based optimizer, using the initial guess from the collection of control

points. Figure 6 is showcasing an overall visual flow-chart for the multiple-point manipulation method, encompassing the steps previously listed for this method visually.

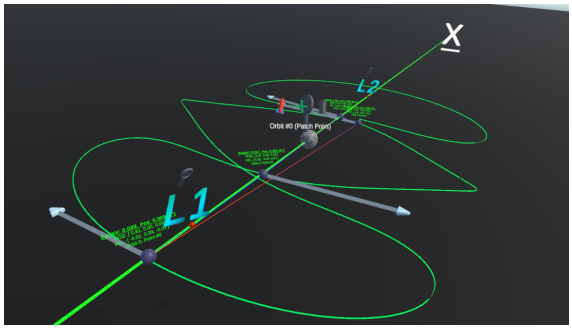


(a) Adding a New Orbit via the Scene Menu

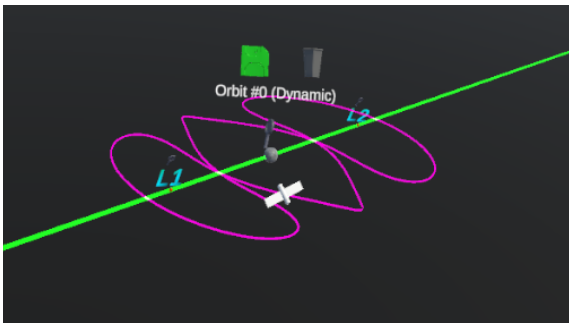


(b) Manipulation of the Initial Position and Velocity for a Control Point

Figure 4: Creating an Orbit



(a) Completed Set of Control Points



(b) Optimized Solution

Figure 5: Generating a Solution

3 THE SYSTEM ARCHITECTURE

The general system architecture for two VR applications is captured in this section, as both the applications have a similar high-level design, with only the user interaction elements being different given the differences in the single-point and multiple-point manipulation initial-guess construction techniques.

3.1 Hardware

The hardware for the application consists of a Virtual-Reality headset, two base-station tracking devices, and a pair of wireless controllers, along with a Windows computer to run the code itself. The user wears the headset and uses the controllers, each with a trigger, menu button, trackpad, and grip, to interact with the scene displayed in the headset. The VR application runs on a Windows computer, which is connected to the headset, and outputs the rendered scene to the headset's display. Controller input is received from the headset and is parsed and handled by the running application.

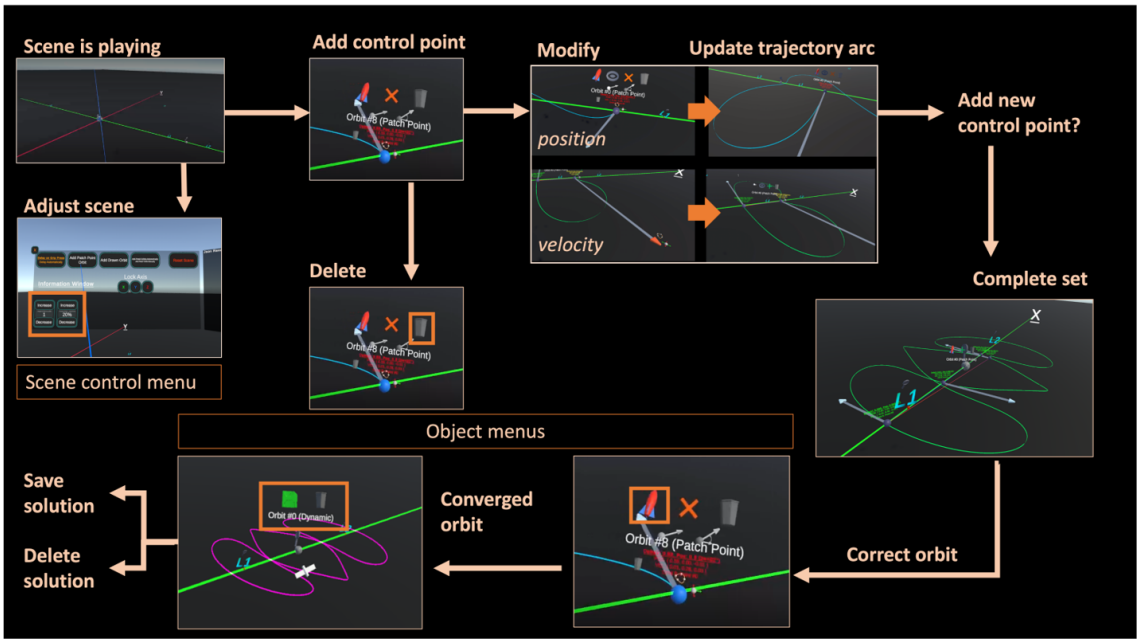


Figure 6: A Visual Flow-Chart for the Multiple-Point Manipulation VR Application

For testing and development, the experimental set-up illustrated in Figure 7) was used, with both a wireless and a wired connection to a Windows computer.

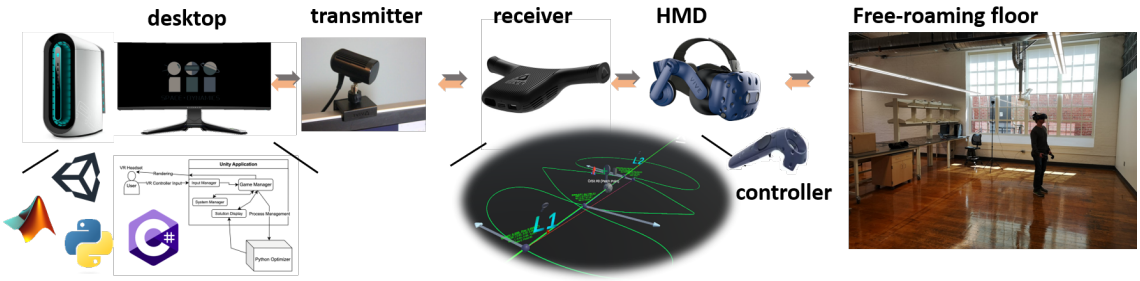


Figure 7: Experimental Set-Up

3.2 Software

The software has two primary components, the VR frontend, built in Unity using the C# programming language, and the backend, built in Python. The frontend is the only component the end user needs to interact with directly, and while it is running, it will launch a backend optimizer as processes on the Windows computer and monitor them until they finish.

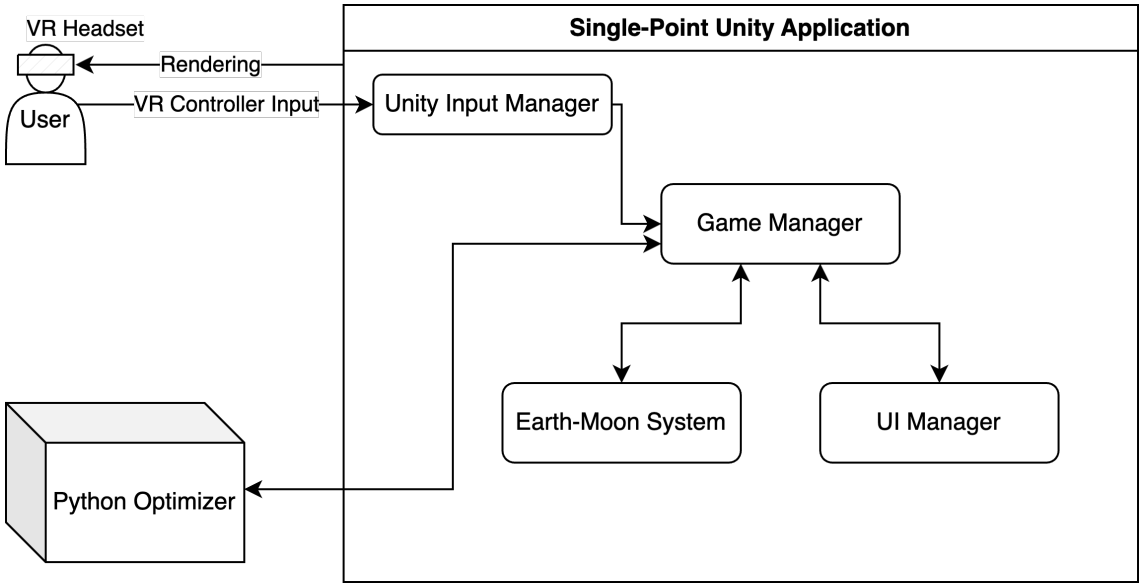


Figure 8: Diagram of the Single Manipulation Point System Architecture

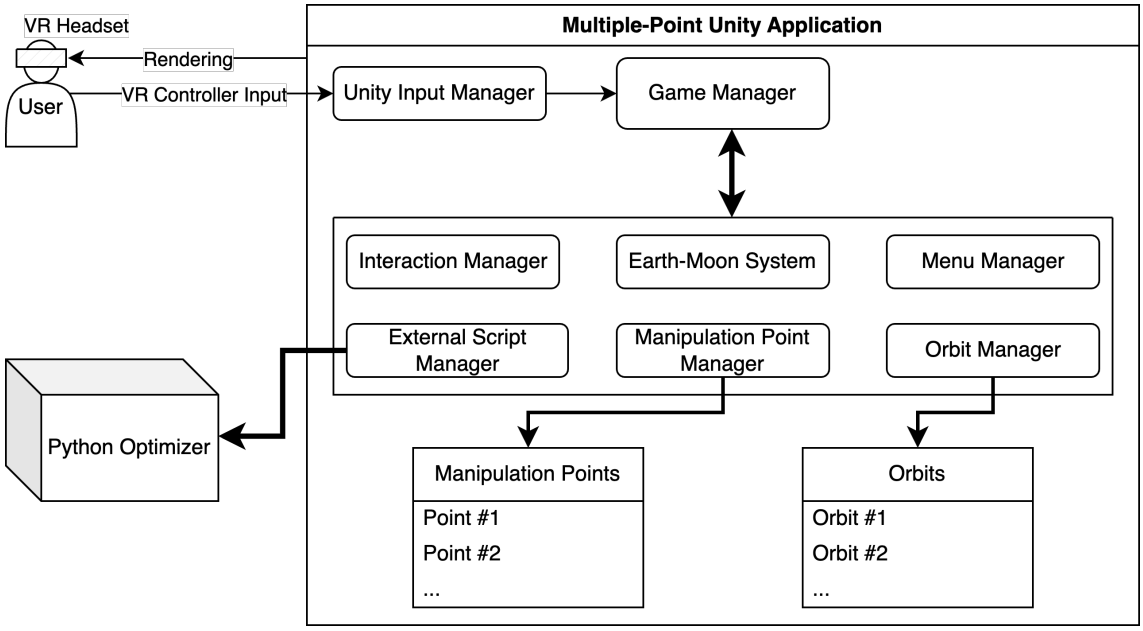


Figure 9: Diagram of the Multiple Manipulation Point System Architecture

Figure 8 shows the system architecture for the single manipulation point VR application. Figure 9 shows the system architecture for the multiple manipulation point VR application. Communication between the front and backend is achieved using files and inter-process calls. When the frontend calls the optimizer, it saves the current initial-guess orbit to a file and passes the filename to the optimizer script on process start. The optimizer then attempts to find a periodic orbit and saves the solution orbit back out to an output file, which is used by the frontend to display in the scene, and allow for archiving.

4 THE NUMERICAL CORRECTION ALGORITHM

A collocation numerical correction algorithm is employed to translate the VR initial guess to a valid CR3BP periodic orbit. The collocation method is an implicit integration method. In the work presented here, we leverage the collocation algorithm described by Somavarapu and Guzzetti.[1] The input and constraint vectors for this corrector algorithm are given in Equations (2).

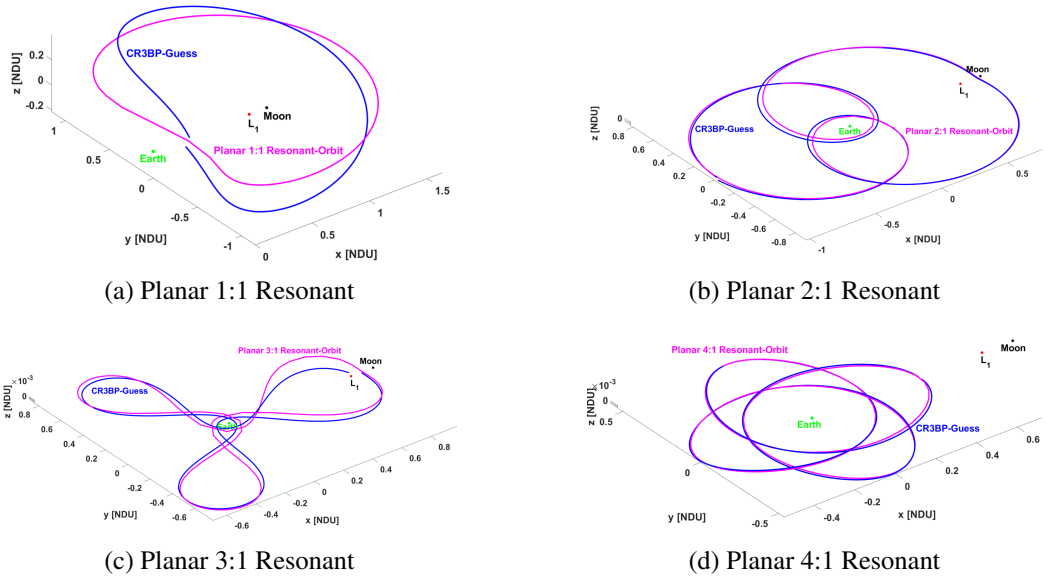
$$\vec{X} = \begin{bmatrix} x_{1,1} \\ . \\ . \\ x_{1,\frac{N+1}{2}} \\ . \\ . \\ x_{n,1} \\ . \\ . \\ x_{n,\frac{N+1}{2}} \\ . \\ . \\ \dot{z}_{1,1} \\ . \\ . \\ \dot{z}_{1,\frac{N+1}{2}} \\ . \\ . \\ \dot{z}_{n,1} \\ . \\ . \\ \dot{z}_{n,\frac{N+1}{2}} \\ T_1 \\ . \\ . \\ T_n \end{bmatrix} \quad \text{and} \quad \vec{F} = \begin{bmatrix} \vec{s}_{1,0} - \vec{s}_{n,f} \\ \vec{s}_{2,0} - \vec{s}_{1,f} \\ . \\ . \\ . \\ \vec{s}_{n,0} - \vec{s}_{n-1,f} \\ [C_1 D - \begin{bmatrix} \dot{\vec{s}}_{1,2} & \dot{\vec{s}}_{1,4} & \dot{\vec{s}}_{1,6} \end{bmatrix}] \times W_1 \\ [C_2 D - \begin{bmatrix} \dot{\vec{s}}_{2,2} & \dot{\vec{s}}_{2,4} & \dot{\vec{s}}_{2,6} \end{bmatrix}] \times W_2 \\ . \\ . \\ . \\ [C_n D - \begin{bmatrix} \dot{\vec{s}}_{n,2} & \dot{\vec{s}}_{n,4} & \dot{\vec{s}}_{n,6} \end{bmatrix}] \times W_n \end{bmatrix} \tag{2}$$

In these equations, n is the number of segments of the periodic orbit, N is the degree of the approximating Legendre-Gauss orthogonal polynomial for each periodic orbit segment. The two subscripts i, j for the state vector $\vec{s}$ indicate the segment and polynomial node identifiers respectively. Matrices C_i, D and W_i are described by Somavarapu and Guzzetti.[1] The components that make up the state vector $\vec{s}_i$ are $\vec{s}_i = [x_i,\, y_i,\, z_i,\, \dot{x}_i,\, \dot{y}_i,\, \dot{z}_i]^T$. The constraints, $\vec{F}$, enforce continuity at segment boundaries and zero derivative error at the defect nodes of the approximating polynomial.

5 RESULTS

Several types of orbits are identified using both single-point and multiple-point manipulation VR applications, leveraging human intuition and geometry of the orbits in the CR3BP system. Some of these orbits are presented in this section. Although we were able to easily recover Lagrange point orbits, we are showcasing other types of complex CR3BP periodic orbits in this case, to

highlight the robustness of the methods. Four types (1:1, 2:1, 3:1, and 4:1) of resonant orbits are identified using the single-point manipulation VR application. Figure 10 is showing the planar resonant orbits identified in this VR application. Figure 11 is showing the spatial (3D) resonant orbits identified in this VR application. We are able to identify complex orbits such as a Horseshoe orbit in this VR application. Figure 12 is showing a Horseshoe orbit identified. Using the multiple-point manipulation approach to initial-guess construction, we can identify system spanning complex orbits such as the $L_1 - L_2$ cycler orbits. Figure 13a is showing an $L_1 - L_2$ Lyapunov cycler orbit identified. Figure 13b is showing an $L_1 - L_2$ halo cycler orbit identified. Figure 14 is showing a Horseshoe orbit identified.



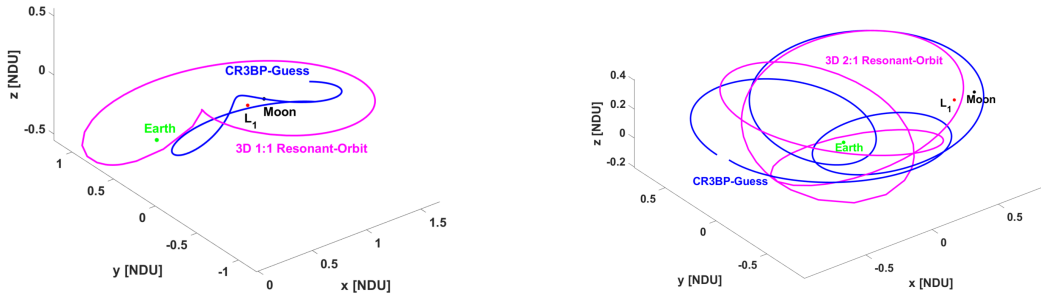
(a) Planar 1:1 Resonant

(b) Planar 2:1 Resonant

(c) Planar 3:1 Resonant

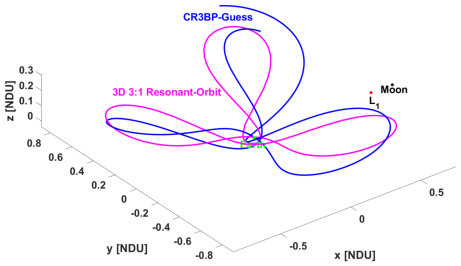
(d) Planar 4:1 Resonant

Figure 10: Sample Solutions for Planar Resonant Orbits (Single-Point Manipulation)

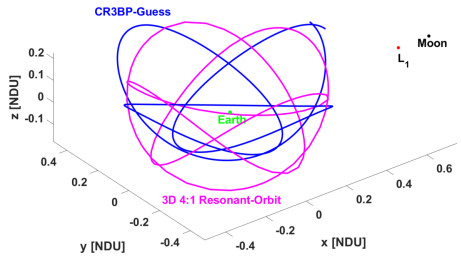


(a) Spatial 1:1 Resonant

(b) Spatial 2:1 Resonant



(c) Spatial 3:1 Resonant



(d) Spatial 4:1 Resonant

Figure 11: Sample Solutions for Spatial (3D) Resonant Orbits (Single-Point Manipulation)

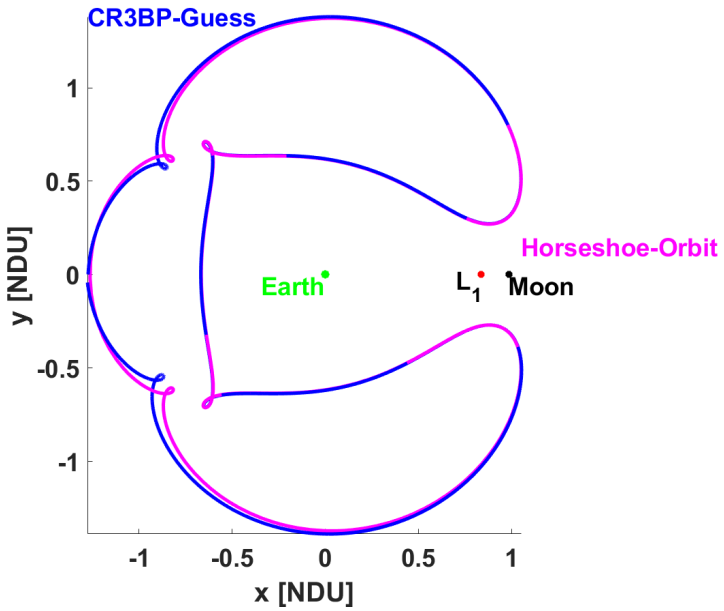
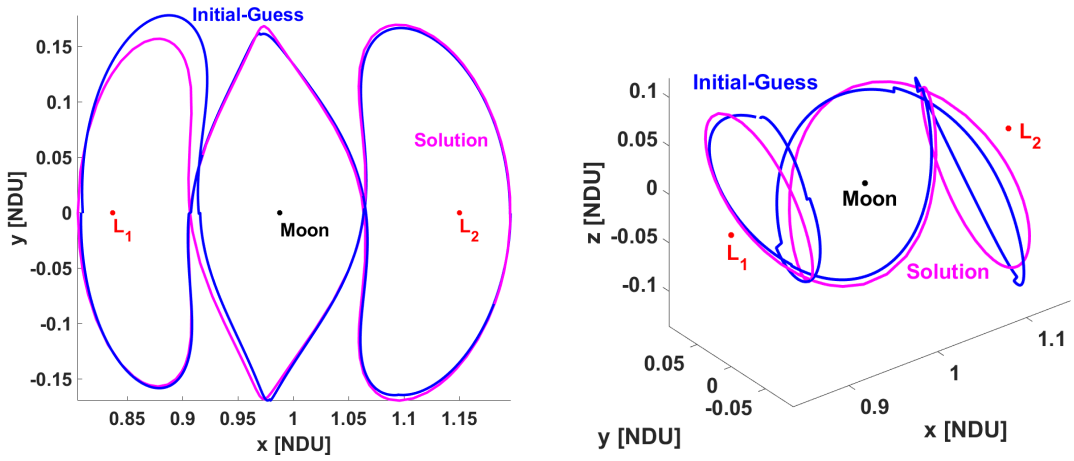


Figure 12: A Horseshoe Orbit (Single-Point Manipulation)



(a) An $L_1 - L_2$ Lyapunov Cycler Orbit

(b) An $L_1 - L_2$ Halo Cycler Orbit

Figure 13: Cycler Orbits Identified with Multiple-Point Manipulation

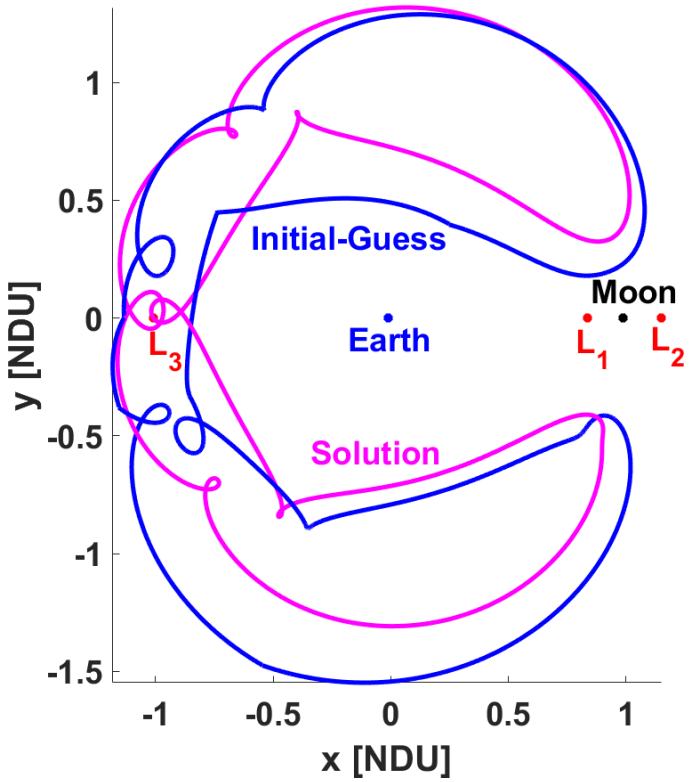


Figure 14: A Horseshoe Orbit (Multiple-Point Manipulation)

6 CONCLUSION

Each of the orbits identified in the results section 5 is constructed and identified in under five minutes of real-time on the VR platform without any previous knowledge of the initial position and velocity vectors. Based on our experience with the tests conducted on this platform, we believe our method is a quick and versatile solution. These results serve in advancing the formulation of immersive and interactive frameworks for universal initial-guess construction in the complex orbital motion model that is the cislunar CR3BP model.

7 ACKNOWLEDGEMENTS

Eirik Mulder is grateful for the financial support received through Auburn University Undergraduate 2023-24 Research Fellowship.

REFERENCES

[1] D. H. Somavarapu and D. Guzzetti, "Toward Immersive Spacecraft Trajectory Design: Mapping User Drawings to Natural Periodic Orbits," *Acta Astronautica*, Vol. 184, 2021, pp. 208–221.

[2] D. H. Somavarapu and D. Guzzetti, "Toward Immersive Spacecraft Trajectory Design: Mapping Arbitrary Drawings to Natural CR3BP Periodic Orbits," *AAS/AIAA Spaceflight Mechanics Conference, Austin, TX*, 2023.

[3] R. Mathur, "Visual Interactive Trajectory Design," *AAS/AIAA Astrodynamics Specialist Conference, Portland, ME*, 2019.

[4] E. B. A. Ramaswamy, "Visualizing and Designing Spacecraft Trajectories in VR," August 2023, https://www.youtube.com/watch?v=eSr3uFUjeko.

[5] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A Software Framework for Nonlinear Optimization and Optimal Control," *Mathematical Programming Computation*, Vol. 11, No. 1, 2019, pp. 1–36, 10.1007/s12532-018-0139-4.