BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

ON-ORBIT DOCKING SIMULATOR

SIMULÁTOR DOKOVÁNÍ NA OBĚŽNÉ DRÁZE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **ROSTISLAV ČERVENKA**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. PETER CHUDÝ, Ph.D., MBA**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

154974

Institut:	Department of Computer Graphics and Multimedia (DCGM)
Student:	**Červenka Rostislav**
Programme:	Information Technology
Title:	**On-orbit docking simulator**
Category:	Mobile applications
Academic year:	2023/24

Assignment:

1. Discuss the historical development of spacecraft on-orbit docking.
2. Research key features of spacecraft dynamics modeling.
3. Design and implement an on-orbit docking simulator for Android OS.
4. Perform testing and evaluation of the developed docking simulator.
5. Suggest future research directions.

Literature:

- Chaub, Hanspeter, "Analytical mechanics of space systems," 2018, 978-1-62410-521-0, 4338127229.
- Ho, Chi-Chang J. and McClamroch, N. Harris, "Automatic spacecraft docking using computer vision-based guidance and control techniques," Journal of Guidance, Control, and Dynamics, Vol.16, No. 2, pp.281-288, 1993, doi: 10.2514/3.21001.

Requirements for the semestral defence:
Tasks No. 1, 2 and partially task No. 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Chudý Peter, doc. Ing., Ph.D., MBA**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	9.5.2024
Approval date:	23.1.2024

Abstract

This thesis delves into the multifaceted domain of on-orbit docking. The goal is to design and implement an on-orbit docking simulator for Android devices. The simulator will be based on the Clohessy-Wiltshire equations, which are used to model the relative motion of two spacecraft in orbit. The user will be in control of one of them and his goal is to utilize the available touch and motion inputs, to dock into the other. The simulator encapsulates the complexities of multiple on-orbit docking scenarios, providing an immersive and educational experience for users interested in space technology.

Abstrakt

Tato práce se ponoří do rozmanité domény dokování na oběžné dráze. Cílem je navrhnout a implementovat dokovací simulátor na oběžné dráze pro Android zařízení. Simulátor bude založen na Clohessy-Wiltshireových rovnicích, které se používají k modelování relativního pohybu dvou kosmických lodí na oběžné dráze. Uživatel bude ovládat jednu z nich a jeho cílem je využít dostupné dotykové a pohybové vstupy a připojit se k druhé. Simulátor zapouzdřuje různé scénáře dokování na oběžné dráze a poskytuje pohlcující a vzdělávací zážitek pro uživatele se zájmem o vesmírné technologie.

Keywords

docking, simulator, orbit, Clohessy–Wiltshire equations, CW equations, Android, spacecraft, space, Unity

Klíčová slova

dokování, simulátor, oběžná dráha, Clovessy–Wiltshireovy rovnice, CW rovnice, Android, kosmická loď, vesmír, Unity

Reference

ČERVENKA, Rostislav. *On-orbit docking simulator*. Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Peter Chudý, Ph.D., MBA

On-orbit docking simulator

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Ing. Peter Chudý, Ph.D., MBA. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Rostislav Červenka
May 5, 2024

Acknowledgements

I would like to express my gratitude to my supervisor Mr. doc. Ing. Peter Chudý, Ph.D., MBA for his support regarding this thesis.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
AOSP	Android Open Source Project
APKs	Android Application Packages
CW	Clohessy-Wiltshire
CSA	Canadian Space Agency
ESA	European Space Agency
ISS	International Space Station
JAXA	Japan Aerospace Exploration Agency
JSC	Johnson Space Center
LM RCS	Lunar Module Reaction Control System
Li-DAR	Light Detection and Ranging
NASA	National Aeronautics and Space Administration
PCs	Personal Computers
SDK	Android software development kit
UI	User Interface
3D	Three-Dimensional

List of Symbols

Symbol	Meaning
x, y, z	Coordinates of a spacecraft in a three-dimensional space
$\dot{x}, \dot{y}, \dot{z}$	First derivatives of the spacecraft's position
$\ddot{x}, \ddot{y}, \ddot{z}$	Second derivatives of the spacecraft's position
ω	Angular velocity
ω_0	Initial angular velocity
μ	Gravitational parameter
t	Time
Δt	Change in time
$x(t), y(t), z(t)$	Position at a given time
x_0, y_0, z_0	Initial position
A, B	Matrixes
F	Force
F_x, F_y, F_z	Component of force in the given direction
m	Mass
u	Velocity
τ	Torque
r	Distance from a reference point
θ	Angle
$\Delta\theta$	Change in angle
α	Angular acceleration
I	Moment of inertia
ψ	Yaw angle
θ	Pitch angle
ϕ	Roll angle
$R_z(\psi), R_y(\theta), R_x(\phi)$	Rotation matrixes along given axes
R	Rotation matrix

Chapter 1

Introduction

Space exploration, a continuously evolving field of science, involves various processes and technologies. One such example is on-orbit docking, a process in which two spacecraft connect to each other in space. It is a critical maneuver for space missions, as it allows the construction of space stations, refueling and repairing spacecraft, and transferring crew and cargo. However, on-orbit docking is a challenging task that requires precise calculations and maneuvering for its success.

While space exploration and its accompanying technologies fascinate many, the intricate nature of on-orbit docking can present a significant challenge in comprehending the process. This is where a simulator can help.

This work aims to create a simulator of the on-orbit docking process, providing an immersive and realistic experience for enthusiasts, students, and professionals.

The simulator will be developed for the Android platform, taking advantage of its capabilities, such as the gyroscope and the touch screen, to create an engaging control system. Combining this with the immense accessibility of Android devices, this work aims to provide a broad audience with an opportunity to experience the reality of on-orbit docking.

Chapter 2 provides a historical overview of the development of on-orbit docking. It covers the world's first docking achievements, the Apollo program, which is important in this project, and some information about docking in modern times.

Chapter 3 explains the modeling of spacecraft dynamics. It covers the Clohessy-Wiltshire equations, which are used to model the relative motion of two spacecraft in orbit, attitude dynamics, which are used to model the spacecraft's rotation, and rotational matrices, which combine these two.

Chapter 4 describes the design of the project. It also covers the Android operating system, which was chosen for the project, and the Unity game engine, which was used to implement the simulator.

Chapter 5 covers the project's implementation. It explains the simulator's development process and the individual components used in it.

Chapter 6 contains a description of user testing. It covers the testing process, the form used in it, and the feedback received.

Chapter 7 talks about the future development of the project. It covers the possible improvements and features that could be added to the simulator.

Chapter 8 concludes the thesis. It summarizes the work done and the results achieved.

Chapter 2

Historical development of on-orbit docking

This chapter provides an overview of the historical development of on-orbit docking. From the first conceptualizations to the modern-day achievements, this chapter covers the most significant milestones in the field of on-orbit docking, providing us insight into the impact of these achievements on space exploration.

2.1 First docking achievements

At the beginning of space exploration, the idea of docking two spacecraft in orbit was considered a challenging task. This section will cover the first docking attempts and the lessons learned from them.

Planning

Before beginning the docking procedure, the two spacecraft must get close enough. To achieve this, they use the space rendezvous procedure, a set of orbital maneuvers in which two spacecraft arrive at the same orbit and close to each other [37]. The Gemini 6 was to be the first spaceflight to achieve a crewed rendezvous with the uncrewed Agena Target Vehicle. The mission was planned for launch on October 25, 1965, but had to be canceled when its Agena target vehicle was destroyed during its launch [45].

Instead, the National Aeronautics and Space Administration (NASA) decided to launch an alternative Gemini 6A mission, which would perform the first crewed rendezvous with the already orbiting Gemini 7 spacecraft, launched eight days prior. This first attempt was scheduled for December 12, 1965, but was aborted minutes after lift-off due to a booster malfunction. The second attempt was made on December 15, 1965, and was successful, with Gemini 6A rendezvousing with Gemini 7 in orbit. This marked the first successful rendezvous of two crewed spacecraft [46].

Successful docking

The first-ever successful docking of two spacecraft was performed during the Gemini 8 mission. The primary goals of Project Gemini included proving the techniques required for the Apollo Program to fulfill President John F. Kennedy's goal of landing a man on the Moon and returning him safely to Earth before the end of the 1960s. The as-

tronauts chosen for Gemini 8 were Neil Armstrong, a civilian test pilot with experience in the X-15 rocket plane program, and David Scott, a U.S. Air Force pilot. Neil Armstrong was chosen as the command pilot for Gemini 8, and David Scott was the pilot. The mission was planned to be three days long, with a rendezvous and docking with an Agena target vehicle on the first day, a spacewalk on the second, and a deorbit burn on the third. For the docking procedure, the astronauts practiced using a simulator at the Manned Spacecraft Center, later renamed the Lyndon B. Johnson Space Center (JSC), in honor of the late President [39].

The Agena target spacecraft was launched before Gemini 8 on March 16, 1966. Nearly two hours later, Gemini 8 was launched from the Florida Cape Kennedy Air Force Station. The launch was successful, and the spacecraft began the rendezvous procedure. Everything went according to plan, and the spacecraft successfully docked with each other at a relative speed of 30 cm/s. However, after some positioning maneuvers, the spacecraft began to roll unexpectedly. After realizing that the problem was on the side of the Gemini, Armstrong undocked. After removing the Agena, the roll rate increased to one rotation per second, and the astronauts were forced to use the re-entry control system thrusters to stop the roll. The spacecraft was stabilized, but with the re-entry system used, the astronauts were forced to abort the mission and perform an emergency re-entry to follow the mission rules. The re-entry was successful, and the astronauts landed in the Pacific Ocean instead of the planned Atlantic, where they were recovered by the backup recovery ship, the U.S.S. Leonard P. Mason. The post-mission analysis showed that the problem was caused by a stuck thruster on the Gemini, which caused the spacecraft to roll [39].

Crew transfer

The Soviet Union employed automated docking systems, unlike the United States, which used manual docking. The Soviet Soyuz 4 mission was the first operation to perform a crew transfer between two spacecraft. The primary goal was to dock with the Soyuz 5 spacecraft, transfer two crew members from it, and deliver them back to Earth. The Soyuz 4 spacecraft was launched on January 14, 1969, with the Soyuz 5 launched two days prior. The maneuver was successful, with the two spacecraft being docked for over four hours. With a connection tunnel between the spacecraft not yet developed, the astronauts had to perform a spacewalk to transfer between the spacecraft. The mission became a resounding success for the Soviet space program [50].

Docking with a space station

The world's first space station was the Soviet Salyut 1, launched into Earth's orbit on April 19, 1971. To make docking with the station easier, the Soviet Union developed an internal transfer tunnel so that astronauts wouldn't have to transfer to the space station using extravehicular activities [47]. The first mission to Salyut 1 was called Soyuz 10 and launched on April 22, 1971. The objective of docking with the space station was not successful because of the failure of the automatic control system. The mission had to be called off, and the astronauts returned to Earth without achieving their goal [49].

A second attempt to dock with the space station was made on 7 June 1971 during the Soyuz 11 mission. This docking attempt was successful as the spacecraft connected after a 3-hour and 19-minute-long maneuver. The three crew members, Georgy Dobrovolsky, Vladislav Volkov, and Viktor Patsayev, boarded the space station and stayed on board for 22 days [38].

On June 22, the crew reactivated the Soyuz 11 and started their return to Earth one hundred thirty kilometers above Earth; the Soyuz 11 was integrated into its three components using explosive bolts, with the crew staying inside the middle one. During the separation, an accident occurred as a pressure equalization valve opened. The valve was supposed to open when the spacecraft was already inside the Earth's atmosphere, but it opened way earlier because of the shock caused by the explosive bolts. The space vacuum outside the capsule caused the air to escape and the pressure to rapidly drop. The astronauts were not wearing their pressure suits at the time and died within two minutes. The ground crew discovered this only after opening the hatch of the capsule. The three crew members are even now the only humans to have ever died in space [38].

Docking with an inert space station

The Soviet space station Salyut 7 lost contact with Earth on February 17, 1985. With all its systems shut down, the space station began making unpredictable movements in the Earth's orbit. All attempts to revive the station from Earth failed, and the author of the space station even reached -20 degrees Celsius [48][3].

To repair the station and bring it back to life, the Soyuz T-13 mission was approved. The Soviet spacecraft typically utilized common docking procedures, but manual docking was required, with the Salyut 7 computers being unresponsive. The space stations' uncontrolled movements further complicated this procedure. The astronaut Vladimir Dzhanibekov was chosen to lead this mission, as he had previously successfully docked with Salyut 7. Accompanying him would be Viktor Savinykh [3].

A successful launch happened on June 6, 1985, and the spacecraft neared the space station. After matching the station's rotation, a first-ever docking with an inert space station was achieved. The cosmonauts then connected the dead station's batteries with its solar panels and then rotated the station to allow sunlight to reach them. Afterward, they began working on reviving the Salyut 7 systems. They achieved this feat in just ten days and were the first to resurrect an inert space station and bring it back to life. The astronauts then remained on the space station for some time before successfully returning to Earth [3].

2.2 Apollo program

NASA's Apollo program used the lunar orbit rendezvous strategy to land men on the moon and bring them back to Earth. This strategy required using two spacecraft: the command and lunar modules. The command module would remain in lunar orbit, while the lunar module would land on the moon and then return to the command module, where they would re-dock. Only the command module would return to Earth, and the lunar module would be discarded [40].

Docking with a lunar module

The first docking of a command module with a lunar module was performed during the Apollo 9 mission. The goals of the mission were to test the lunar module in Earth orbit, including docking and undocking procedures with the command module. The mission was to take ten days and would be launched using the Saturn V rocket, the same rocket that would be used for the Apollo 11 mission. The lunar module was named Spider, and the command module was named Gumdrop [23].

The mission was launched on March 3, 1969, with astronauts James McDivitt, David Scott, and Rusty Schweickart. The lunar and command modules were first docked when the lunar module was still attached to the Saturn V rocket. On the fifth day of the mission, the two spacecraft were undocked, and the lunar module performed a test flight, which took about 185 kilometers away from the command module. The lunar module was then re-docked with the command module, and the two spacecraft remained docked for the rest of the mission. The mission was successful, and the astronauts returned to Earth on March 13, 1969 [23].

Apollo 11

Before the Apollo 11 mission started, more testing of the lunar orbit rendezvous was required. This was done during the Apollo 10 mission, launched on May 18, 1969. The mission was to test the lunar module in lunar orbit and to test the procedures for the Apollo 11 mission. The mission was successful, and the lunar module was brought to within 15 kilometers of the lunar surface [22].

The Apollo 11 mission was launched on July 16, 1969, with astronauts Neil Armstrong, Buzz Aldrin, and Michael Collins. The mission was to perform the first crewed landing on the Moon. The mission was a resounding success, and the astronauts safely returned to Earth on July 24, 1969. The spacecraft used for the mission had two main components: the command module Columbia and the lunar module Eagle [44][43].

During the Apollo 11 mission, there were two docking events. The first happened during the outbound trip to the Moon when the transposition, docking, and extraction maneuver was performed. This maneuver involved the Columbia separating from the Saturn V rocket, turning around, and docking with the Eagle. The Eagle was then extracted from the third stage of the Saturn V rocket, and the two spacecraft remained docked until they reached the moon orbit. There, they separated, and the Eagle landed on the moon, while Columbia remained in lunar orbit [44][43].

The second docking event happened after the successful lunar landing. The Eagle launched from the moon and rendezvoused with the Columbia in lunar orbit. The two spacecraft docked, and the astronauts transferred from the Eagle to the Columbia. The Eagle was discarded, and Columbia began the journey back to Earth. The Columbia splashed down in the Pacific Ocean on July 24, 1969, and the U.S.S. Hornet safely recovered the astronauts [44][43].

Apollo-Soyuz

After intense competition between the United States and the Soviet Union during the Cold War, the two countries decided to cooperate on a joint mission in space, The Apollo-Soyuz test Project. The mission took five years to realize and was the first international space mission to take place. The mission was to be a symbol of relaxation of the strained relationship between these two countries [17].

The first to launch was the Soviet Soyuz 19 on July 15, 1975, from the Baikonur Cosmodrome in Kazakhstan. On the same day, about eight hours later, Apollo launched. This Apollo mission wasn't officially numbered and counted as a separate mission after the conclusion of the Apollo program [17].

The spacecraft made orbital adjustments for two days to reach the same orbit. On July 17, the vehicles successfully docked together, and the first international space handshake took place. For this to succeed, the two nations developed a universal docking system

jointly. The two crews remained docked for 47 hours and completed numerous joined activities, both in scientific and political nature. A successful undocking followed, and both spacecraft safely returned to Earth. This marked the last collaborative docking mission in a while, as the next successful docking of a U.S. and a Russian spacecraft would be almost 20 years later [17][**?**].

2.3 Modern docking

Docking technology has advanced greatly since the first attempts and is now used more than ever. This section will cover some of the most significant impacts of modern docking.

International Space Station

The International Space Station (ISS) is a cooperative project of five space agencies and 15 countries. The agencies are NASA - United States, Roscosmos - Russia, European Space Agency (ESA) - Europe, Japan Aerospace Exploration Agency (JAXA) - Japan, and Canadian Space Agency (CSA) - Canada. It is the largest space station to have ever been constructed. It consists of numerous modules launched separately and then connected in space. Its primary purpose is scientific research and performing space environment experiments [27].

The construction of the ISS began in 1998 when the first module, Zarya, was launched. The Russian modules were almost all docked robotically. The Space Shuttle delivered the other modules and then had to be linked by astronauts during extravehicular activities with the help of the shuttle's robotic arm. The first crewed mission to dock with the space station was the Russian Soyuz TM-31 in the year 2000, and the space station has been occupied ever since [26].

To enable docking between various spacecraft from different countries, the International Docking System Standard was formulated. The standard was developed with collaboration from all the space agencies cooperating on the ISS [12].

After the Space Shuttle was discontinued in 2011, the ISS is also serviced by commercial cargo vehicles, notably SpaceX's Dragon and the Orbital Sciences Corporation's Cygnus [26].

Autonomous docking technologies

Autonomous docking refers to the automated procedure through which spacecraft autonomously approach and attach themselves to other spacecraft or space stations, eliminating the necessity for human intervention. This process relies on a fusion of sensors, algorithms, sophisticated guidance, navigation, and control systems. This allows the spacecraft to execute precise maneuvers and successfully dock with their designated targets without human involvement [36].

This type of docking requires really precise sensors to determine the relative positions and velocities of the spacecraft. This is achieved using Light Detection and Ranging (LiDAR), radar systems, and cameras [36].

Machine Learning and Artificial Intelligence algorithms are used to make decisions in real-time. These intelligent systems calculate the optimal trajectory and can quickly adapt to changing conditions [36].

These technologies have already been tested on numerous space missions. SpaceX's Crew Dragon and Cargo Dragon spacecraft have, for example, used this technology to dock with the ISS autonomously, and they are not the only ones [36].

There are also challenges, with the need for standardization and ensuring the reliability and safety of operations being the main ones. Despite them, autonomous docking has the potential to significantly reduce the complexity and cost of spacecraft servicing, which is why its future is looking bright [36].

Chapter 3

Spacecraft dynamics modeling

3.1 Clohessy–Wiltshire equations

The Clohessy-Wiltshire (CW) equations can be used to model the relative motion of two spacecraft in an orbit. These two spacecraft are called the chaser and the target. In this model, the target is in a circular orbit around a central body, as is the chaser. The chaser is assumed to be the one docking with the target whose movements are calculated by the CW equations. The target is instead assumed to be moving without change in its orbit [31] [2].

This model is a first-order approximation, which means it doesn't consider some less impactful forces affecting the system. It is assumed that the distance between the chaser and the target is small enough so that the gravitational forces generated by the central body affect each of them the same. The central body is assumed to be a point mass, and perturbations from other celestial bodies are also not considered [31] [2].

This model uses a rotating target-centered coordinate system. The origin of the coordinate system is in the center of the target spacecraft, and the coordinate axis rotates with it as it moves in its orbit [18].

There are three axes in this coordinate system. The x-axis is along the radius vector of the target spacecraft, the y-axis is along the velocity vector of the target spacecraft, and the z-axis is along the angular momentum vector of the target spacecraft [2]. You can see the diagram in Figure 3.1.

The CW equations are [2]:

$$\ddot{x} = 3\omega^2 x + 2\omega\dot{y} \tag{3.1}$$

$$\ddot{y} = -2\omega\dot{x} \tag{3.2}$$

$$\ddot{z} = -\omega^2 z \tag{3.3}$$

where $\omega = \sqrt{\mu/a^3}$ is the orbital rate of the target body. μ is the gravitational parameter of the central body, and a is the target body's circular orbit radius.

These equations have a closed-form solution, which can be used to calculate the position and velocity of the chaser spacecraft at any time, given the initial position and velocity. The solution is [2]:

$$x(t) = Ax_0 \tag{3.4}$$

where the matrix A is:

$$\mathbf{A} = \begin{bmatrix} 4-3\cos(\omega t) & 0 & 0 & \frac{1}{\omega}\sin(\omega t) & \frac{2}{\omega}(1-\cos(\omega t)) & 0 \\ 6(\sin(\omega t)-\omega t) & 1 & 0 & -\frac{2}{\omega}(1-\cos(\omega t)) & \frac{1}{\omega}(4\sin(\omega t)-3\omega t) & 0 \\ 0 & 0 & \cos(\omega t) & 0 & 0 & \frac{1}{\omega}\sin(\omega t) \\ 3\omega\sin(\omega t) & 0 & 0 & \cos(\omega t) & 2\sin(\omega t) & 0 \\ -6\omega(1-\cos(\omega t)) & 0 & 0 & -2\sin(\omega t) & 4\cos(\omega t)-3 & 0 \\ 0 & 0 & -\omega\sin(\omega t) & 0 & 0 & \cos(\omega t) \end{bmatrix}$$

where $\mathbf{x(t)} = [x(t), y(t), z(t), \dot{x}(t), \dot{y}(t), \dot{z}(t)]^T$, and $\mathbf{x_0} = [x_0, y_0, z_0, \dot{x}_0, \dot{y}_0, \dot{z}_0]^T$. x_0, y_0, z_0 are the initial coordinates, and $\dot{x}_0$, $\dot{y}_0$, $\dot{z}_0$ are the initial velocities of the chaser spacecraft. t is the time since the beginning of the simulation.

If there are thrusters that apply a force to the chaser spacecraft, then the equations change [31]:

$$\frac{F_x}{m} = \ddot{x} - 3\omega^2 x - 2\omega\dot{y} \tag{3.5}$$

$$\frac{F_y}{m} = \ddot{y} + 2\omega\dot{x} \tag{3.6}$$

$$\frac{F_z}{m} = \ddot{z} + \omega^2 z \tag{3.7}$$

The solution also changes [19]:

$$x(t) = Ax_0 + Bu \tag{3.8}$$

where the matrix B is:

$$\mathbf{B} = \begin{bmatrix} \frac{1}{\omega^2}(1-\cos(\omega t)) & \frac{2}{\omega^2}(\omega t - \sin(\omega t)) & 0 \\ -\frac{2}{\omega^2}(\omega t - \sin(\omega t)) & \frac{4}{\omega^2}(1-\cos(\omega t)) - \frac{3}{2}t^2 & 0 \\ 0 & 0 & \frac{1}{\omega^2}(1-\cos(\omega t)) \\ \frac{1}{\omega}\sin(\omega t) & \frac{2}{\omega}(1-\cos(\omega t)) & 0 \\ -\frac{2}{\omega}(1-\cos(\omega t)) & \frac{4}{\omega}\sin(\omega t) - 3t & 0 \\ 0 & 0 & \frac{1}{\omega}\sin(\omega t) \end{bmatrix}$$

where $\mathbf{u} = [F_x/m, F_y/m, F_z/m]^T$, F_x, F_y, F_z are forces applied by thrusters, and m is the mass of the chaser spacecraft.

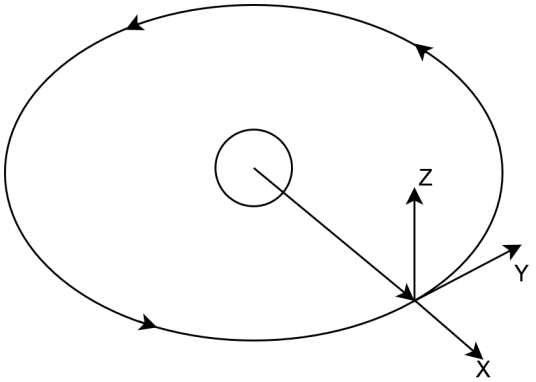

Figure 3.1: CW Coordinate system

3.2 Attitude dynamics

In this model, the spacecraft is assumed to be a rigid body with three fixed principal axes of rotation. This means the spacecraft's shape and mass distribution don't change during the simulation. Negligible forces, such as gravitational forces, are also not considered.

To determine the change in orientation in a spacecraft, torque needs to be calculated. Torque is the product of the force applied and the distance from the rotation point. The equation to calculate torque is [7]:

$$\tau = rF\sin(\theta) \tag{3.9}$$

where τ is the torque, r is the distance from the point of rotation to where the force is applied, F is the force, and θ is the angle between the force and the lever arm vectors.

Based on the torque applied, we can use Newton's second law for rotation to calculate angular acceleration. Angular acceleration is the rate of change of angular velocity. The equation to calculate it is [41]:

$$\alpha = \frac{\tau}{I} \tag{3.10}$$

where τ is the torque, I is the moment of inertia, and α is the angular acceleration.

The moment of inertia measures an object's resistance to changes in its rotation rate. It is calculated as [41]:

$$I = \sum_i m_i r_i^2 \tag{3.11}$$

where i is the number of point masses that make up the object, m_i is the mass of the i-th point mass, and r_i is the distance from the i-th mass to the axis of rotation.

If we assume constant torque during a certain time, then we can calculate the angular velocity using the following equation [41]:

$$\omega = \omega_0 + \alpha\Delta t \tag{3.12}$$

where ω is the angular velocity, ω_0 is the initial angular velocity, α is the angular acceleration, and Δt is the time period.

Finally, to calculate the new orientation of the spacecraft, we can use the following equation [6]:

$$\Delta\theta = \omega\Delta t + \frac{1}{2}\alpha\Delta t^2 \tag{3.13}$$

where $\Delta\theta$ is the change in orientation, ω is the angular velocity, α is the angular acceleration, and Δt is the time period.

The orientation of the spacecraft will be represented using Euler angles. There are three such angles, known as yaw (ψ), pitch (θ), and roll (ϕ). They will be updated based on the calculated change in orientation $\Delta\theta$.

3.3 Vector rotation

To model the motion of a spacecraft in orbit using the CW equations, we need to calculate the vector $\mathbf{u} = [F_x/m, F_y/m, F_z/m]^T$. The forces applied by the thrusters and the spacecraft's mass are known, but based on the spacecraft's orientation, the forces need to be transformed from the target-centered coordinate system into the one used by the CW equations.

To do this, we need the rotation matrix. The rotation matrix is a matrix that can be used to perform rotations in a three-dimensional Euclidian space. First, we need the yaw (ψ), pitch (θ), and roll (ϕ) matrices, and these are used to rotate around the z, y, and x axes respectively. These matrices are [42]:

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

To get the final rotation matrix, we need to multiply these three matrices together, which gives us the following:

$$\mathbf{R} = \begin{bmatrix} \cos(\psi)\cos(\theta) & \cos(\psi)\sin(\theta)\sin(\phi) - \sin(\psi)\cos(\phi) & \cos(\psi)\sin(\theta)\cos(\phi) + \sin(\psi)\sin(\phi) \\ \sin(\psi)\cos(\theta) & \sin(\psi)\sin(\theta)\sin(\phi) + \cos(\psi)\cos(\phi) & \sin(\psi)\sin(\theta)\cos(\phi) - \cos(\psi)\sin(\phi) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}$$

Finally, to get the needed vector $\mathbf{u} = [F_x/m, F_y/m, F_z/m]^T$ we need to multiply the rotation matrix by the applied forces, like this:

$$\mathbf{u} = \mathbf{R} \begin{bmatrix} F_x/m \\ F_y/m \\ F_z/m \end{bmatrix} \tag{3.14}$$

Now, the vector u is ready to be used in our CW equations.

Chapter 4

Design

4.1 Android operating system

Android is a mobile operating system that integrates hardware and software resources. It is used in various devices, such as smartphones, tablets, smartwatches, and even cars [5]. Android is by far the most popular mobile operating system, with a market share of 70.48% as of January 2024, which is leagues ahead of the second most popular operating system, iOS, with its market share of 28.8% [32].

The development of Android started back in 2003, when Andy Rubin, Rich Miner, Nick Sears, and Chris White founded Android Inc. Originally, it was supposed to improve the operating systems of digital cameras, but the company's direction quickly changed to focus on smartphones, due to the decline of the digital cameras market. In 2005, Google acquired the company, but the founding members continued to work on the operating system (OS). After Apple launched its iPhone, Google formed the Open Handset Alliance. This group of 84 technology and mobile companies work together on developing Android. The first official version, Android 1.0, launched in 2008, with a new version getting released almost every year, culminating with the newest Android 14 released in October 2023 [9].

Android is an open-source operating system, meaning anyone can use and modify it for their purposes. This is one of the main reasons for its popularity, as it allows a wide variety of devices to be created using it. It is based on Linux, another open-source operating system. Only the core of Android is open source and is part of the Android Open Source Project (AOSP) [1]. Most Android devices have Google Play Services, which bring Google-branded products to Android, such as YouTube, Gmail, and the Google Play Store. To utilize these services on your created Android device, you need a license from Google [5].

The development of applications is done using the Android software development kit (SDK) and the Kotlin, Java, and C/C++ languages. Google Play Store is the primary source of applications for Android [14]. It enables users to download, install, update, and remove over 3.5 million apps as of January 2024 [34]. The file format used is the Android Application Packages (APKs). There are also third-party application marketplaces, such as the Amazon Appstore [14].

Android was chosen as the operating system for the docking simulator because of its widespread user base, open-source nature, and the fact that it is used in a wide variety of devices.

4.2 Unity Game Engine

For implementing this project I have chosen the game engine Unity, the main reasons were its beginner friendliness and its support for Android. Unity is cross-platform. It allows development for personal computers (PCs), mobile devices, consoles, virtual reality platforms, and even web browsers. This game engine can be used to either create three-dimensional (3D) or two-dimensional experiences [35].

The Unity game engine was launched in 2005, initially for Mac OS X and later for Microsoft Windows and Web browsers. 2012, the Asset Store was introduced, allowing developers to trade with assets, scripts, and tools. Originally Unity used to use the Boo programming language for its scripts, which changed with the release of Unity 5.0 in 2015 [10].

Unity uses the C# programming language for scripting. C# is a general-purpose programming language developed by Microsoft. C# is object-oriented and statically typed, and its memory is managed automatically. Many developers widely use it, and it is the fifth most popular programming language as of January 2024 [33].

The main building block in Unity is a Game Object, which is its basic element. Everything in Unity is a Game Object, including the camera, lights, user interface (UI) elements, and, of course, all the „physical“ objects, such as cars, the ground, etc. The Game Objects are organized into a tree structure. The tree's root is the scene, the environment in which the game takes place [35].

The Game Objects can have components attached to them. These components define the behavior of the Game Object. There are many types of components, such as the Transform component, which defines the Game Object's position, rotation, and scale. Other important ones are the Rigidbody, Colliders, Renderers, and scripts, which allow changing the Game Objects' behavior by manually writing in C# [35].

The default physics engine in Unity is the PhysX engine, which Nvidia developed. It is a real-time physics engine that simulates rigid bodies, cloth, fluids, and soft bodies. This project will not use it as we will only use rigid bodies whose movements are dictated by our equations [35].

4.3 Camera

The camera is one of the most essential elements in a game. It is the player's window into the game world. During this simulation, only one camera will be used, making the user feel as if they are controlling the chaser spacecraft.

The camera used in this simulation is a first-person camera, which typically means that it is positioned at the user's eye level. That is only partially true for this simulation, and the user won't be able to see the spacecraft as the camera will be situated right at the spacecraft's tip. The camera will then move along with the spacecraft, and the user can see the docking process from the spacecraft's perspective.

4.4 User Interface

The user interface (UI) is the point where the interaction between the user and the application happens. The goal is to make it as intuitive and easy to use as possible. The UI for this application is divided into two parts. The first and the most important is the in-game UI,

which the user sees during the simulation. The second is the main menu, which is used when the user opens the application. In this application, a lot more focus will be put into the first one, and the second one will have fundamental controls, where the user can select the docking scenario and start the simulation.

The UI used during the simulation will have three main components: the functions panel, the information panel, and the control panel. The first one is the functions, which will control the simulation. There will be buttons to pause, resume, reset, and end the simulation and one that will explain the different parts of the information and control panels. The functions panel will be positioned in the top left corner of the screen.

Information panel

The second component is the information panel, which will display information about the simulation. The first item displayed is the coordinates x, y, and z of the chaser spacecraft relative to the target's docking port. The total distance to cross will be calculated and displayed based on these numbers. The second item is the spacecraft's relative rotation and angular velocity in the roll, pitch, and yawn angles. Finally, the rate at which the chaser moves towards the target will be displayed.

Translation controls

The third component is the control panel, which will control the chaser spacecraft. It is divided into two parts: the translation controls and the rotation controls. The translation controls will move the spacecraft in the x, y, and z directions.

The translation controls will be situated in the bottom right of the screen and comprise six buttons, two for each axis. Figure 4.1 shows the buttons' exact position and functions. The buttons will be slightly transparent to allow the user to see uninterrupted. After touching a button, the corresponding thrusters will be activated, and the spacecraft will begin to move in the desired direction. The thrusters are deactivated when the user stops touching the button.

The user may not always want to use the thrusters at full force, so a slider will be on the left side of the screen, allowing the user to control the force the thrusters apply.

Rotation controls

The rotation controls offer an intuitive way to manipulate the chaser spacecraft's orientation by leveraging your phone's motion sensor capabilities. Instead of traditional buttons, these controls are designed to respond to the physical rotation of your device. The rotations of the device correspond to the movement in the yaw, pitch, and roll angles, as seen in Figure 4.2.

This project will use the rotation vector sensor, which represents the device's orientation as a combination of an angle and an axis, in which the device rotates through an angle θ around an axis (x, y, or z). The rotation vector sensor is a software-based sensor that combines data from the device's gyroscope, accelerometer, and magnetometer, which are hardware-based sensors [15].

A button will be added to the control panel to allow users to change their device's rotation without affecting the spacecraft's orientation. The device's current orientation will be saved when the user presses this button. It will be situated in the bottom left corner of the screen.

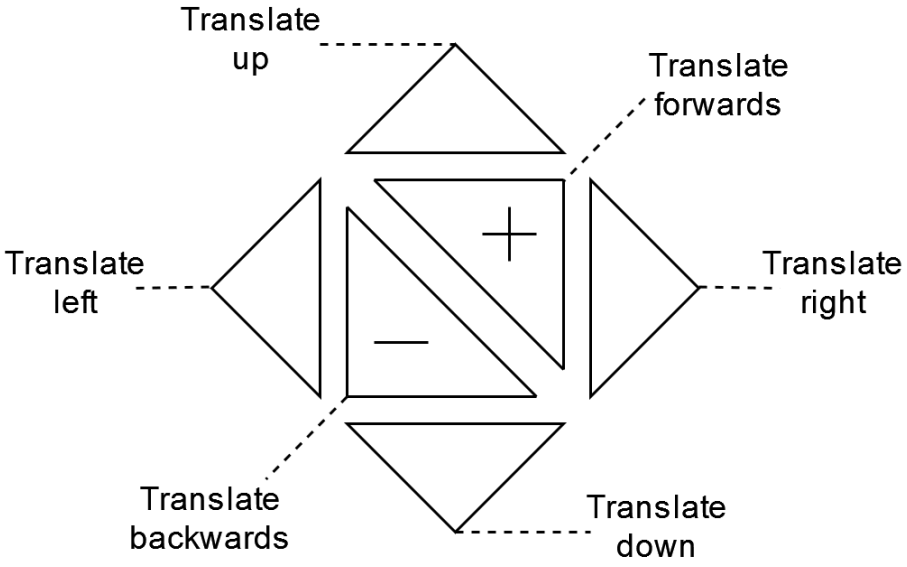

Figure 4.1: Translation controls

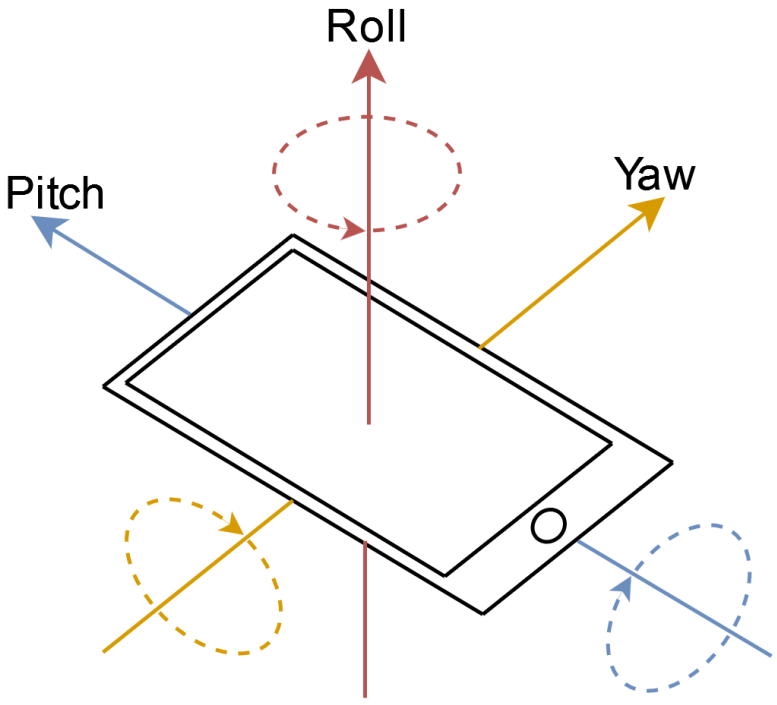

Figure 4.2: Rotation controls

4.5 Scenarios

The docking simulator will offer two distinct scenarios based on real-life events that immerse the user in the world of space exploration.

The first scenario will be based on the famous Apollo 11 mission. The user will control the iconic Eagle spacecraft, returning from its successful moon landing. The user's mission is to successfully dock with the Columbia spacecraft waiting for him in moon orbit.

The second scenario will feature an astronaut brought into orbit using the Space Shuttle. The user's goal will be to use the astronaut's Extravehicular Mobility Unit to navigate through space and successfully enter the Hubble Space Telescope.

4.6 Simulation's finish

The simulation can end in two ways: either the user successfully docks the chaser spacecraft or crashes into the target. In the first case, the user will be greeted with a success screen displaying the time it took to complete the simulation. In the second case, the user will see a failure screen showing the parameters that caused the crash.

To succeed, the user must precisely approach the target's docking port at an appropriate speed. According to the International Docking Standards, Table 4.1 shows the initial contact conditions.

Table 4.1: Initial Condition Limiting Values for Docking Simulation [29]

Initial Condition	**Limiting Value**
Closing (axial) rate	0.05 to 0.10 m/sec
Lateral (radial) rate	0.04 m/sec
Pitch/Yaw rate	0.20 deg/sec (vector sum of pitch/yaw rate)
Roll rate	0.20 deg/sec
Lateral (radial) misalignment	0.10 m
Pitch/Yaw misalignment	4.0 deg (vector sum of pitch/yaw)
Roll Misalignment	4.0 deg

Chapter 5

Implementation

This chapter will describe the implementation of the docking simulator, its development process, the challenges faced, and the solutions found.

5.1 Starting the project

The first step in creating the docking simulator was to set up a Unity project. I downloaded Unity Hub, which is a management tool for Unity projects. It allows for creating new projects, opening existing ones, and managing their Unity versions.

Setting up the project was very straightforward. I created a new project and chose a 3D mobile template, which added all the necessary packages for mobile development. I also downloaded and selected the Unity Editor version 2020.3.16f1, the latest version released then. Thought this came with some problems, as some of the tutorials available on the official Unity website were not available for this version of the Unity Editor.

As this was my first time working with Unity, I utilized one of their learning templates. These templates are pre-made projects with an already implemented game containing some exercises, which can be used to learn the basics of Unity development. I had to download an older version of the Unity Editor 2021.3.33f1 to use these learning templates, as that was the newest available version. I tried all the templates available, but I concentrated on one of them, which was a kart racing 3D game, which, while not the same as my intended project, had enough similar features to be useful for my learning process.

For creating and editing c# scripts, which are used in Unity, I used Visual Studio 2022. This is the recommended IDE for Unity development, and it is integrated into Unity, so it can be opened directly from the Unity Editor. I already had Visual Studio installed on my computer, and had experience working with it, so this was a natural choice for me.

5.2 Unity scenes

Scenes in Unity are assets that contain all the objects in a game. They create different parts of the game, such as the main menu, levels, and settings. In my project, I created four scenes: the main menu, the Apollo 11 mission, the Hubble Space Telescope mission, and the tutorial.

1. Main Menu - This scene is the first one the user sees when they open the application. Its main purpose is to allow the user to navigate to other scenes and to greet the user

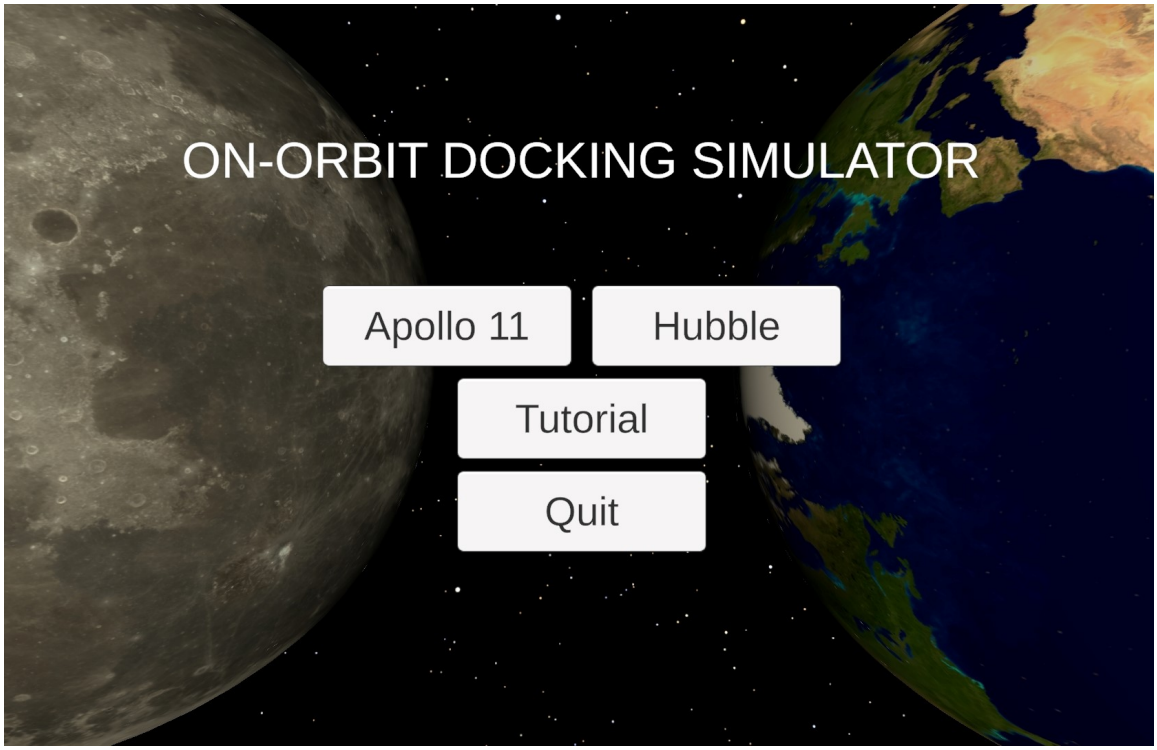

Figure 5.1: The main menu of the docking simulator

when they open the application. The scene contains four buttons: two for starting the chosen mission, one for opening the tutorial, and the last for closing the application. In the background of the scene are the models of the Earth and the Moon, which are rotating, each one on their side, closer to the mission that corresponds to them. The scene can be seen in Figure 5.1.

2. Tutorial - This scene explains the simulation's controls and UI to the user. It also tells him what constitutes a successful docking and how to achieve it. More about the tutorial can be found in the 5.10 section.

3. Apollo 11 - This scene is one of two game scenes. This one corresponds to the Apollo 11 scenario. It contains the models of the Columbia spacecraft and the Moon.

4. Hubble Space Telescope - This scene is the second game scene. This one corresponds to the Hubble Space Telescope scenario. It differs from the Apollo 11 scene in the models used (Earth and the Hubble Space Telescope), the starting position and rotation of the chaser spacecraft, and the values used to calculate the chaser's movements in space using the CW equations.

Camera

The implementation of the camera for this project is quite simple. There is only one camera in the scene, situated at the tip of the chaser spacecraft. It moves along with it to provide the user with a first-person perspective of the docking process. The only special thing about this camera I edited is its clipping planes, which are the distances from the camera

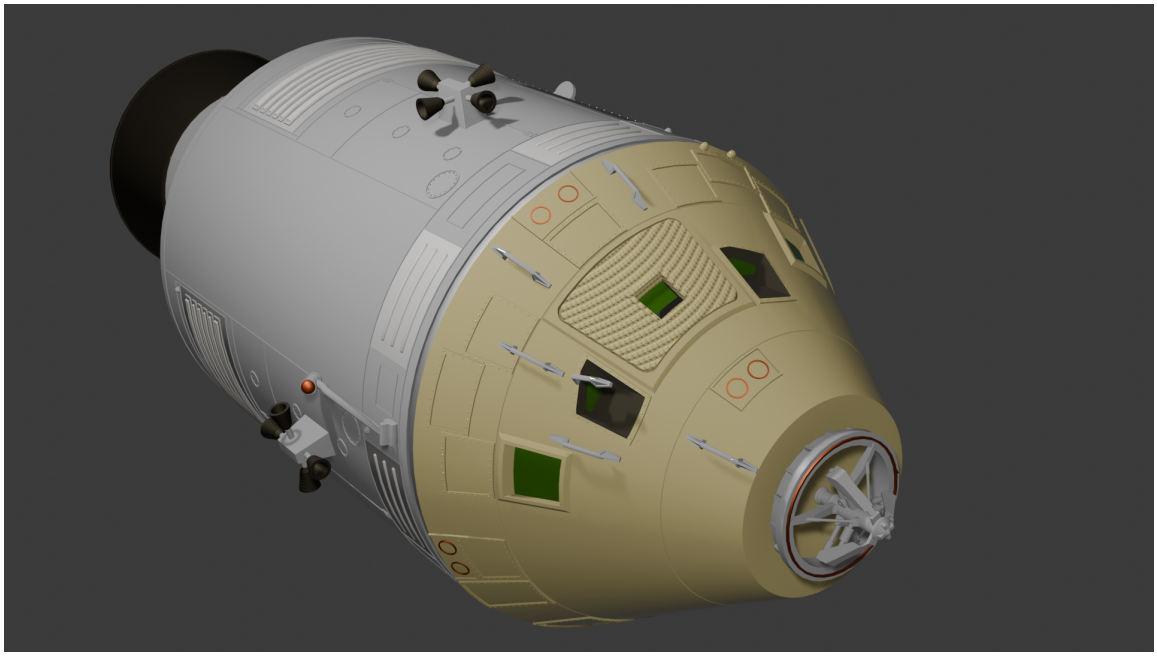

Figure 5.2: Columbia model in Blender

where objects start to render and stop rendering. I set the far-clipping plane to the highest possible value to ensure the user could see the Moon or the Earth, as they are far from the spacecraft. The near-clipping plane was set to 1.55 meters, which allows the user not to see the chaser spacecraft, as it is the distance from the camera to the spacecraft's tip.

5.3 Models and textures

In this section, I will describe the models and textures used in this project, how I obtained them, and how I implemented them into the Unity project.

Models

The first model implemented in this project was the Columbia spacecraft from the Apollo 11 mission. Initially, I had planned to obtain this model from the official NASA website, but I sadly found that model to be entirely lacking for my purposes. For this project, this model needed to have a highly detailed docking port because the user would see it close up, and that was not the case with the NASA model. After some searching, I found a model on the website Cadnav[8] with a non-commercial license, which was perfect for my purposes. The model was in the .max format, a 3D modeling file format used by Autodesk's 3ds Max software and not supported by Unity. That meant I had to convert it to a format that Unity can use, the .fbx format. To do this I used the Autodesk 3ds Max software, which I had to download and install, typically this software is not free, but Autodesk offers a 30-day free trial, which was enough for my purposes. The model was then placed into its designated Apollo 11 scene and was ready to be used. The model can be seen in Figure 5.2.

Initially, the second model I planned to implement in this project was the Eagle spacecraft from the Apollo 11 mission. However, I reconsidered this notion, as I decided on the position of the camera, which will be situated at the tip of the spacecraft, which means

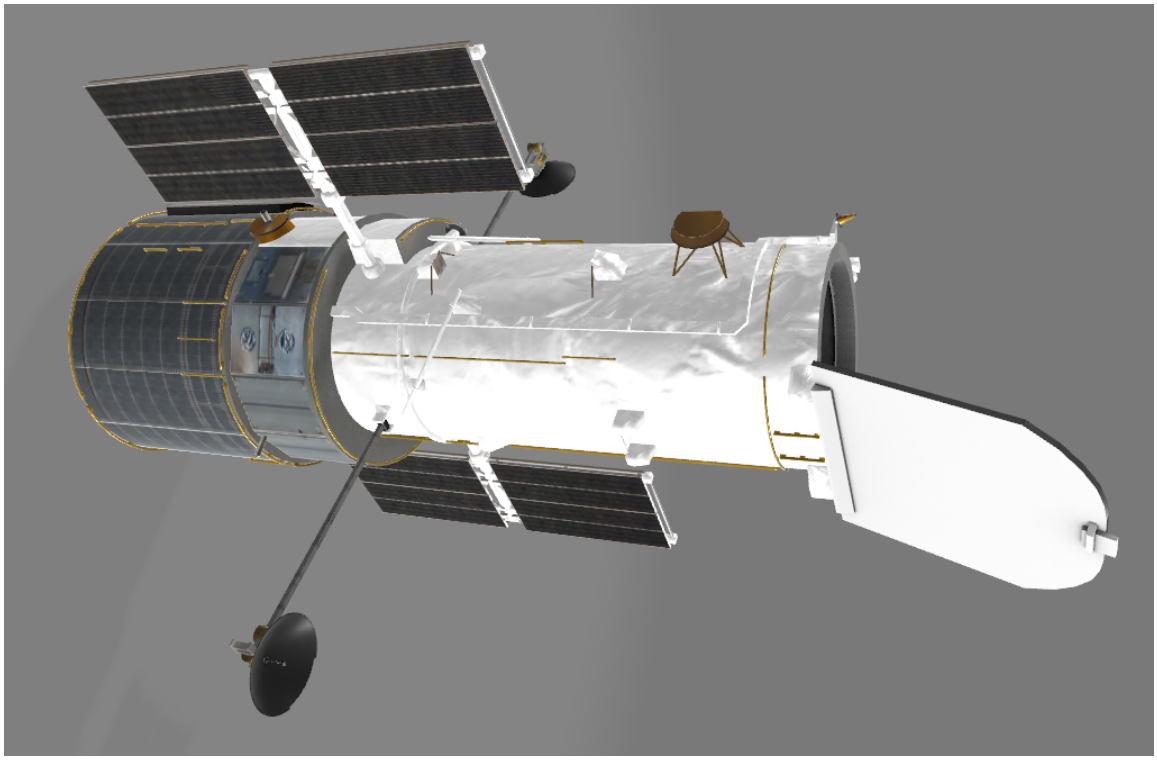

Figure 5.3: Model of the Hubble Space Telescope

the spacecraft will not be visible to the user viz. section 5.2. Instead, I used a standard Unity cube with similar dimensions to the Eagle spacecraft to represent it. This cube was used in the Apollo 11 and the Hubble Space Telescope missions.

Another model to implement was the Hubble Space Telescope. This time, I obtained a high-quality model from the official NASA website[4]. The model was in the .glb format, which is a 3D file format that is used for the efficient transmission of 3D models. This format is also not officially supported by Unity, but I found a user-created plugin, UniGLTF[28], which allowed me to import the model into Unity without any problems. The model was then placed into its designated Hubble Space Telescope scene and can be seen in Figure 5.3.

The last model I needed for my project was a sphere model. Although Unity already provides a sphere model, it was not detailed enough for my purposes. I needed a sphere to represent the Moon and the Earth for my two missions, but the problem was that both needed to be significant. When I scaled the Unity sphere that much, it stopped looking like a sphere. I found a highly detailed model of a sphere on the website Sketchfab[30], which was perfect for my purposes. The model was in the .glb format and was easily imported into Unity using the UniGLTF plugin. The model was then placed into the Apollo 11 and the Hubble Space Telescope scenes, where appropriate textures were placed.

Textures

I employed texture compression techniques to optimize performance and reduce memory usage on mobile devices. Unity provides options for compressing textures without significant loss of visual quality, ensuring efficient resource utilization during runtime.

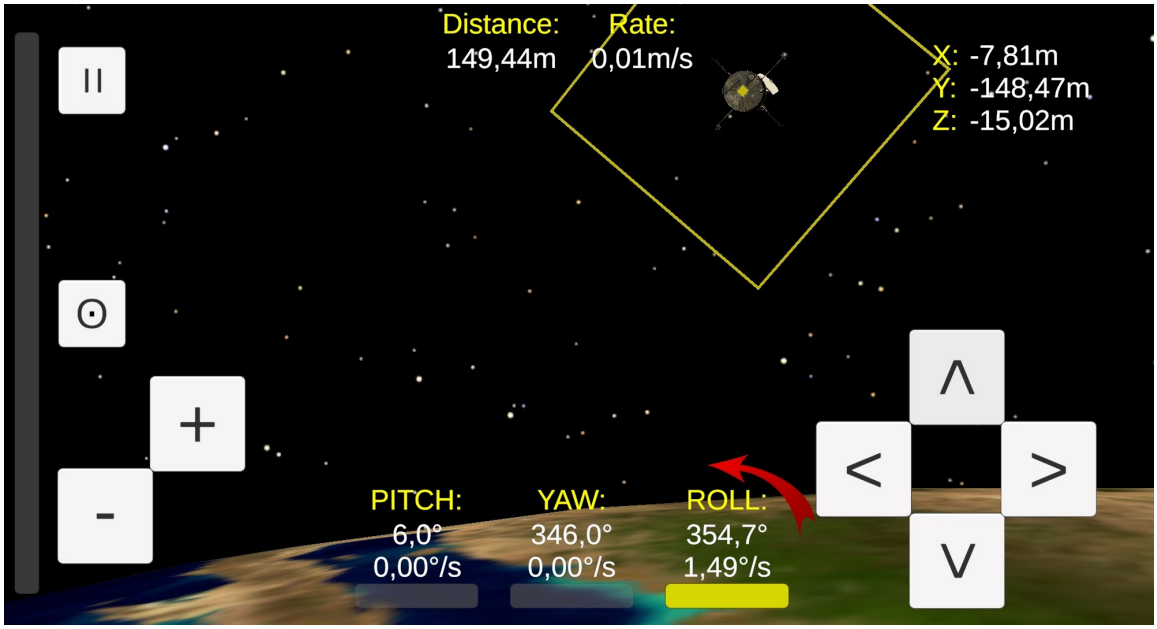

Figure 5.4: The user interface of the docking simulator

Unity automatically textured the Columbia spacecraft and the Hubble Space Telescope models using the textures provided by their creators. Thus, the Earth and the Moon were the only other objects that required textures.

The Moon textures were obtained from the official NASA website[13]. There were many different qualities of textures available, in the end, I chose the 8k resolutions, that were then compressed in Unity to save space. The Earth textures were obtained from a model of Earth on the website Free3d[16] and then compressed. The textures were then placed on the sphere models in the Apollo 11 and the Hubble Space Telescope scenes.

5.4 User Interface

The user interface is a crucial part of the docking simulator, as it is the primary way the user interacts with the simulation. It also provides the user with important information about their controlled spacecraft, essential for a successful docking attempt. These are the two parts of UI, which will be discussed in detail in the following sections. The final UI is in Figure 5.4.

Buttons

This first section of UI consists of eight buttons: the pause button, the six translation motion buttons, two for each axis, and the reset device rotation button. The docking simulator is an Android application, meaning it can be played on relatively small screens, so the buttons are pretty large and appropriately distanced. Icons for these buttons were made using ASCII characters for simplicity and clarity.

I placed the pause button, a staple of most mobile games, on the top left side of the screen. This button pauses the game, hides the UI, and displays a menu that allows the user to re-

sume the game, restart the simulation, or return to the main menu. The icon for this button is a simple equals sign rotated 90 degrees, commonly used to represent pause.

Below the pause button, the reset device rotation button can be found. This button is essential for the user, allowing the simulation to register a new base orientation for the user's device. Based on the divergence from this orientation, the spacecraft will rotate in the yaw, pitch, and roll angles. The device's base rotation changes when the user pauses and resumes the simulation. This allows the user to move his device around without affecting the spacecraft's orientation. The icon for this button is a simple circle, an o ASCII symbol, with a dot in the middle. Even though the function of this button is not evident to the user at first glance solely based on the icon, the user is informed of its function in the tutorial.

The translation motion buttons are situated at the screen's bottom right and left corners. I originally planned for them to be in the same place, but this new layout allows the user to use both hands for simultaneous translation motion in multiple directions. These buttons are divided into two groups: the first group for the y-axis, which, from the user's perspective, moves the chaser forward and backward, and the second group for the x and z axes, for movement up, down, left, and right. The first group is situated in the bottom left corner of the screen. Their icons are the plus ASCII symbol for forward motion and the minus symbol for moving backward. The second group is situated in the bottom right corner of the screen. The icons for these buttons are the greater than and less than symbols for moving right and left and the same symbols but rotated 90 degrees for moving up and down.

Initially, in my design, I planned to have a slider on the left side of the screen for the user to control the force applied by the thrusters, but after implementing it, the UI became too cluttered, so the slider was removed. As its substitution, I modified the translation motion buttons so that the longer the user holds them, the more force is being applied by the thrusters. This allows for precise movements to finish the docking process and large movements when the user is far from the target.

Statistics

The second section of the UI is the stats section. It consists of eleven text fields that change based on the user's manipulation of the chaser spacecraft. The text in these fields is white, and each has a yellow title above or next to them that describes what the text field is showing. These text fields are divided into two groups: the first contains information about the chaser spacecraft's position, and the second contains information about the chaser spacecraft's rotation.

On the top right of the screen, the user can see the chaser spacecraft's x, y, and z coordinates relative to the target. They precisely consider the distance between the target's docking port and the middle of the chaser's front side, where the camera is situated. The x, y, and z coordinates correspond to their counterparts in the CW equations, viz Figure 3.1. These coordinates are displayed in meters with two decimal places, as the user needs such precision to finish docking successfully.

The following text fields are positioned at the screen's top center. It consists of two members; the first displays the total distance to the target, and the second displays the chaser's rate of movement compared to the target. As with the previous text fields, these two are displayed in two decimal places, with the distance in meters and the rate in meters per second, respectively. The total distance is calculated using the Pythagorean theorem, based

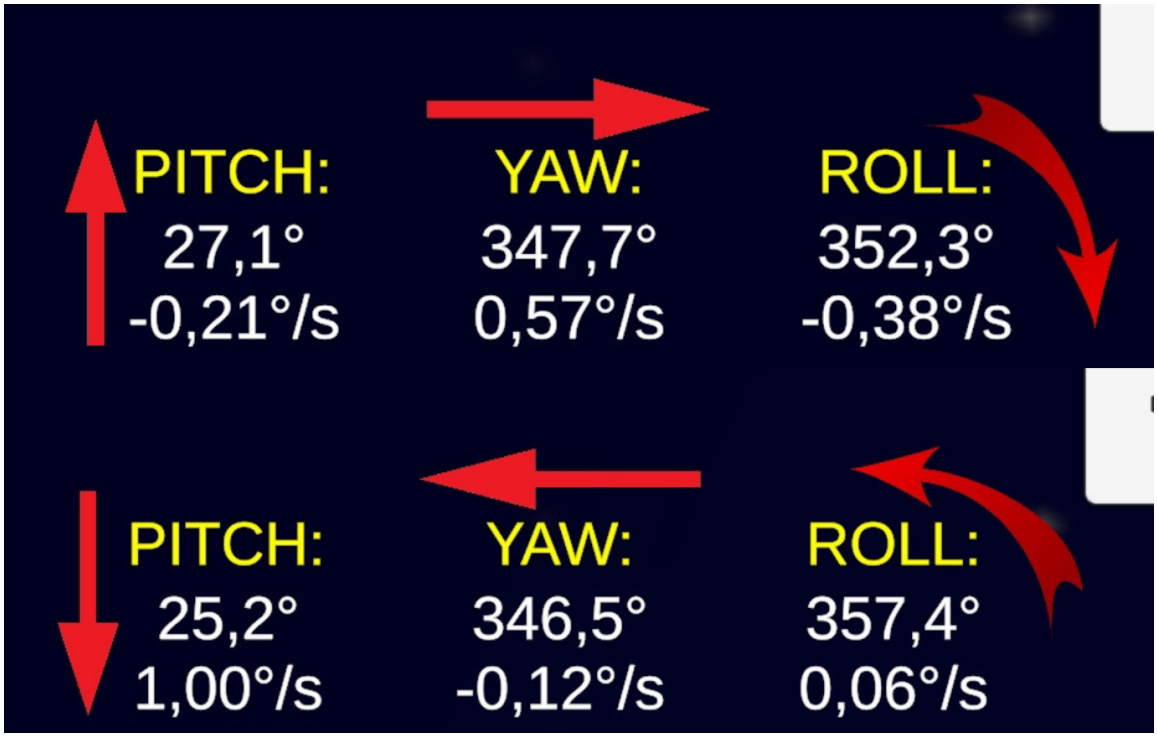

Figure 5.5: UI arrows indicating the rotation of the chaser spacecraft

on the chaser's x, y, and z coordinates. The rate is then determined by getting the target's velocity in all three axes from the CW equations and combining them.

The last set is positioned at the bottom center of the screen. It consists of six text fields, two for each roll, pitch, and yaw angle. The first displays the current rotation of the chaser spacecraft at the corresponding angle, and the second displays the rate at which the chaser is rotating at that angle. The rotation is displayed in degrees with only one decimal place and the rate in degrees per second with two decimal places, allowing the user to almost completely stop the spacecraft from rotating. The rotation rate is determined using the same method employed while calculating the rate of movement toward the target.

I added arrows around each rotation text field to make the UI more user-friendly and let the user know how the chaser spacecraft is rotating at any moment. These arrows are red-colored and are pointing in the direction of the rotation. The arrows are displayed only when the chaser spacecraft is rotating at least 0.05 degrees per second in any given direction, and they disappear when the rotation stops. These arrows can be seen in Figure 5.5.

The last part that can be considered a statistic is the progress bars. There are four of them in the UI, one for each rotation angle and one for translational movement. They illustrate the force the thrusters are currently applying in the corresponding direction. The force starts at zero and increases the longer the user holds the translation motion buttons or the longer the user rotates the device. Three horizontal progress bars are at the bottom of the screen, below the corresponding angle. The fourth is vertical and situated on the left side of the screen. The progress bars can be seen in Figure 5.4.

The text changes color to let the user know when the specific statistic is within the successful docking limits. When the value is within the limits, the text is green. When the value

is close to the limits, the text is yellow, and when the value is outside the limits, the text field and progress bar are normally white.

5.5 Canvases

A canvas in Unity is the fundamental component of the UI. It contains all other UI elements, such as buttons, text fields, and images. There are three different ways to render a canvas in Unity: Word Space, where the canvas is stationary; screen Space - Camera, where it is rendered as a flat texture before a specific camera; and the one used in this project, Screen Space - Overlay. This method renders the canvas on top of everything else in the scene, making it perfect for UI elements. Because this project targets Android devices with differing screen sizes, the canvas is set to scale with the screen size, ensuring that the UI elements are always in the same place and have the same size, regardless of the screen.

A canvas is utilized in every scene in this project, but the two mission scenes have four; the tutorial has three, while the main menu has only one. The main menu scene is the simplest, containing only one canvas with four basic buttons. The mission scenes are more complex, containing four canvases: the first and main one for UI used during the simulation, the second for the pause menu, and the third and fourth ones for the success and failure screens, respectively. The tutorial scene contains three canvases: the first for text and images that explain the simulator, the second is a modified version of the one used during the simulation, and the third for the pause menu because some people need a break even during the tutorial.

The victory canvas is simple. It congratulates the user on a successful docking and allows him to return to the main menu. The victory canvas can be seen in Figure 5.6. The failure canvas is a bit more complex; it displays the parameters that caused the crash and informs the user of the parameter's marginal limits. Here is an example of the text displayed: „Your velocity was too high! Correct under 0.1 m/s. Your velocity: x m/s“. The canvas allows users to restart the simulation or return to the main menu. The pause is simple, too. It consists of three simple buttons: resume, restart, and return to the main menu, and a text that informs the user that the simulation is paused. The pause canvas can be seen in Figure 5.8.

In scenes with multiple canvases, a script manages them after representing them as game objects. It utilizes a public variable to store the canvas currently being used and activates and deactivates it using a built-in SetActive() function based on the user's actions.

5.6 Movement

The target spacecraft doesn't move, so the only movement that needs to be implemented is the chaser spacecraft. The chaser spacecraft's movement is managed by one script, the PlayerMove.cs script. This script is attached to the chaser spacecraft Game Object and is responsible for all motion concerning it.

The script is the same for both missions, but the script then needs to know which mission is currently being played to correctly calculate the chaser spacecraft's movement. This is done by setting a public boolean variable isMoon in the Unity Editor, which is then used in the script to determine the values of the variables used in the CW equations. It also gives different starting values to the chaser spacecraft's position and rotation based on the mission being played to give the user a different experience each time.

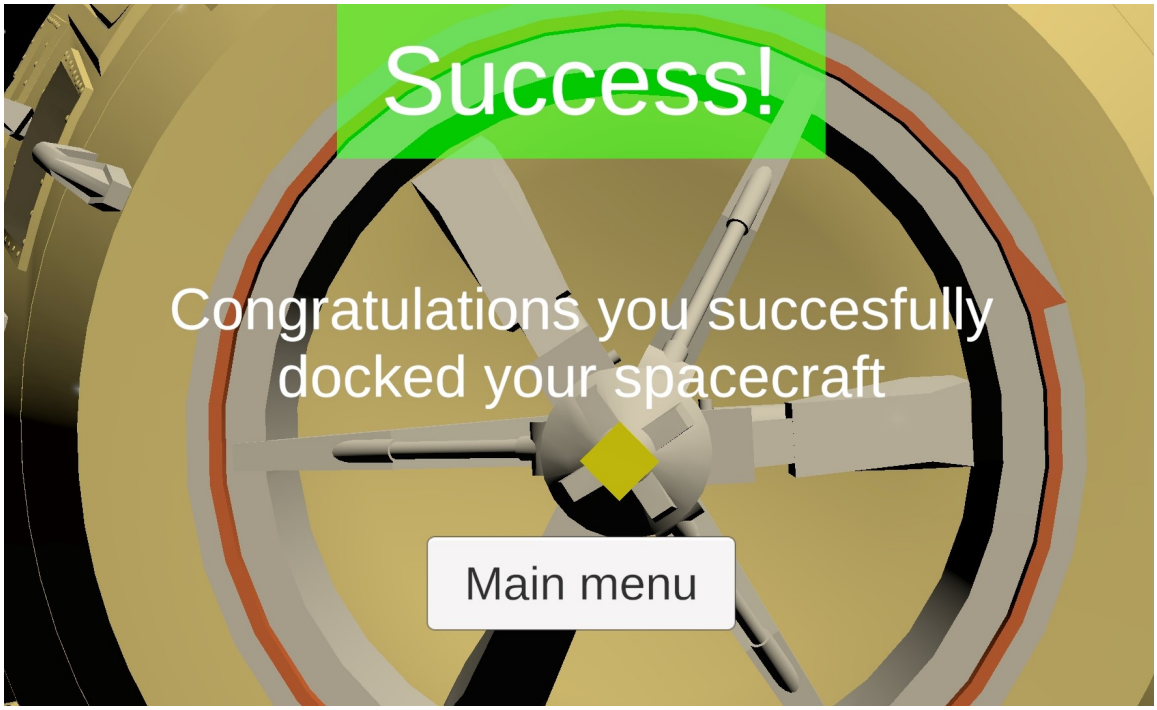

Figure 5.6: Finished docking, success screen, closeup of the Columbia spacecraft

Unity provides a built-in function, Start(), which is called when the script is initialized. In this function, the script initializes some of the variables used in the script that depend on the mission selected and those that need to be calculated: the inertia and orbital rate of the target body.

The main function in the script is another built-in FixedUpdate(). This function is called every frame and ensures that the time between frames is constant at 0.02 s. Another version of it is Update(), which is also called every frame, but the time between frames constantly changes. This behavior wasn't ideal for this project, as the chaser spacecraft's movement is based on the time between frames, so the FixedUpdate() function was used.

Translation implementation

The FixedUpdate() function first checks if any of the movement buttons are pressed, if they are the script updates a vector that represents the force applied by the thrusters. The direction of this vector is then transformed using the current rotation of the spacecraft. This vector is then, along with the current position velocity, orbital rate of the target body, and the time between frames used to calculate the new position of the chaser spacecraft using the CW equations, as described in the 3.1 section. The script then updates the chaser spacecraft's position based on the calculated values.

The standard Unity buttons are designed to detect if they have been pressed but don't have a built-in function to detect if they are being held. I wanted the users to be able to hold the buttons to move the chaser spacecraft, so I had to implement this functionality myself. To do this, I edited the standard Unity button script to call a function when the button is pressed and another when it is released. I then utilized this in the PlayerMove script to set a boolean variable to true when the button is pressed and false when it is released.

This variable is then used in the FixedUpdate() function to determine if the button is being held and how long it has been.

Rotation implementation

The rotation of the chaser spacecraft is changed based on the change in the user's device's rotation. To detect this change, the script uses the user's device's gyroscope sensor.

Before starting to work with the gyroscope in my script, I had to make sure that it was supported on the device. I checked this by using SystemInfo.supportsGyroscope property. The gyroscope could then be accessed using the Input.gyro property. The gyroscope returns a Quaternion, but this could be changed using the .eulerAngles property, which returns the yaw, pitch, and roll angles.

The gyroscope returns the device's current rotation, but the controls are designed to respond to the rotation change, not the rotation itself. To achieve this, the script saves the device's base rotation, which is saved half a second after the start of the simulation, when the user pauses and resumes, and when the reset device rotation button is pressed.

The script uses the RotationDifference () function to calculate the difference between the device's current rotation and the base rotation. This function calculates the difference between two Euler angles but ensures the difference stays between -180 and 180 degrees. This is achieved by subtracting the two angles, and if the result is greater than 180 or less than -180, subtracting or adding 360 from it, respectively. This is useful to determine the direction in which the spacecraft should rotate.

Then, in the FixedUpdate() function, we first check whether the gyroscope is supported; if it is, we get the device's current rotation and base rotation in Euler angles. Then, using the RotationDifference() function, we calculate the difference between the two. Then, we check which Euler angle has the greatest difference. This is done only to allow the user to rotate the spacecraft in one direction at a time, as being able to rotate in multiple directions at once would make this simulation too difficult, as when trying to rotate in one direction, it is really easy to rotate in another by accident.

When it is detected, we verify which direction the device is being rotated in and whether it has reached a specific angle threshold. This threshold is set to 30 degrees, allowing the user to rotate his device accidentally without affecting the spacecraft. If the threshold is reached, the script then calculates the torque the spacecraft is experiencing, which is then used to rotate the spacecraft using equations described in the 3.2 section.

5.7 Chaser parameters

For our CW equations, we need to know the mass of the chaser spacecraft and the force applied by its thrusters. The mass of the Eagle's ascent and descend stages with all fuel was 15103 kg, but for our purposes, we need the weight of only the ascend stage and some of the fuel. The dry mass of the ascend stage was 2445 kg, and it held 2376 kg of propellant, but in our simulation, some of the fuel was already used. I couldn't find a precise number, so I estimated that around half of the fuel was used, leaving us with 1183 kg of fuel, which makes the spacecraft's total mass 3628 kg [20]. During our simulation, the spacecraft's mass will be constant, as the fuel usage is not implemented.

Our simulator will simulate the usage of the Lunar Module Reaction Control System (LM RCS). This system allowed the Eagle spacecraft to perform rotational and translational maneuvers. The Eagle had sixteen thrust chamber assemblies, each providing 445 N

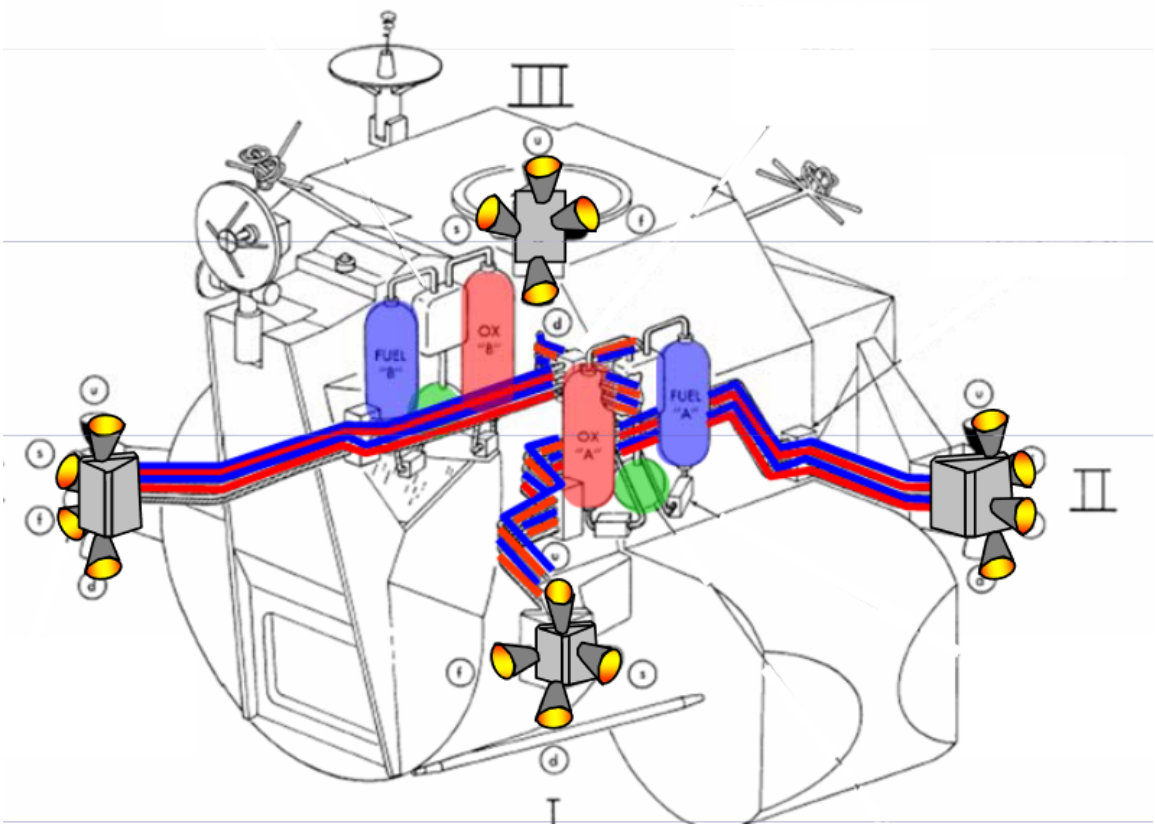

Figure 5.7: Placement of the RCS thrusters [21]

of thrust. You can see their placement in Figure 5.7. Based on their placement, we can see that four were used simultaneously for moving forward and backward, and two were used for the other two axes. The rotation also allowed for the use of four thrusters. Four thrusters have a combined force of 1780 N, and two have 890, and these are the numbers used in our CW equations [21]. This has some problems, such as thrusters not consistently producing constant power. This is simulated using the Random function with a range of 0.5-1.5 that randomizes the output of the thrusters in each frame. The thrusters in our start at no power and, linearly over the course of 2 seconds, go to their maximum. This is illustrated by progress bars on the UI.

5.8 Collision

This project uses Unity's built-in collision detection system to determine whether the chaser spacecraft has attempted to dock with the target spacecraft and whether it's successful or not. This system is based on colliders, which are components that define the shape of a Game Object for physical collisions. Both the chaser and target spacecraft have colliders attached to them; the chaser is box-shaped, and the target in both missions is a capsule. These shapes were chosen as they best represent the spacecraft's shape.

The collision detection system is implemented in the PlayerMove script. The script uses the OnCollisionEnter() function, which is called when two objects in the scene collide. As the only two objects with colliders are the target and chaser spacecraft, we know

that the collision is between them. The function then checks if the chaser spacecraft has met the criteria for a successful docking attempt and displays either the success screen or the failure screen based on the result.

The criteria for a successful docking attempt in this table 4.1 were too complicated for users to understand easily, so I simplified them, and you can see them in this table 5.1.

Table 5.1: Implemented limiting values for the On-orbit Docking Simulator

Initial Condition	**Limiting Value**
Closing (axial) rate	to 0.10 m/sec
Lateral (radial) misalignment	0.10 m
Pitch misalignment	4.0 deg
Yaw misalignment	4.0 deg
Roll Misalignment	4.0 deg

5.9 Background

In Unity, a skybox is used to simulate the background of a 3D scene. It is a cube or a sphere that surrounds the scene and displays a texture on its inside surface. A skybox typically represents the sky, distant landscape, or, like in this project, space. The skybox used in this project was obtained from the Unity Asset Store[11]. It represents a starry night sky, which is perfect for this project as it gives the user the feeling of being in space.

In addition to the skybox, the Earth and the Moon were placed in their respective scenes. To simulate the orbiting of our spacecraft around their respective celestial body, they are being rotated around their y-axis. The Earth is rotated at a rate of approximately 0.063 degrees per second, as the Hubble Space Telescope orbits the Earth every 95 minutes [25]. On the other hand, the Columbia spacecraft orbited the Moon every 2 hours [24], so the moon's rotation is a little slower, at approximately 0.05 degrees per second.

Object positioning

To simplify the calculations of the chaser spacecraft's position, the center of the target docking port was aligned with the origin of the scene, positioned at the (0, 0, 0) coordinates. The initial chaser's position differs in the two missions, but the distance to the target remains relatively the same.

The original plan was to set the Earth's and Moon's positions to reflect reality, but Unity doesn't allow objects to exist that far from their origin. To solve this problem, I divided the distance between them and the origin by 100 and then did the same with their size to hide this fact from the user.

5.10 Tutorial

The tutorial is a section of the application that explains the simulation's controls and UI to the users. It also tells them what constitutes a successful docking and how to achieve it. Originally, the tutorial in this project was supposed to be simply text and image-based, but after some testing, I found that users had a hard time understanding the spacecraft's rotation controls. To solve this, I made the tutorial more interactive.

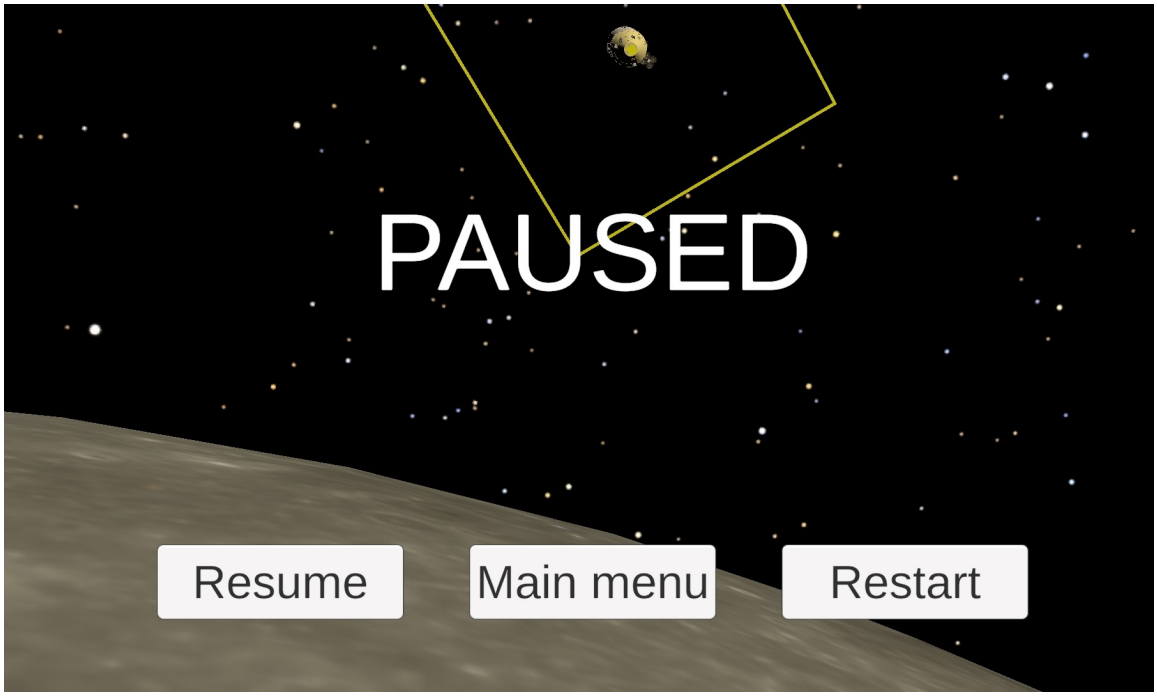

Figure 5.8: Paused canvas with the moon and the skybox in the background

When you first open the tutorial, it welcomes you, and right after that, you find yourself in the rotation controls section. After that comes the explanation of the UI and the translation controls. Finally, the explanation of what constitutes a successful docking and how to achieve it will be shown.

In the rotation section of the tutorial, your goal for this section is to try to stabilize the rotation of your spacecraft between 359° and 1° for at least five seconds. The movement script won't allow the spacecraft to move, so you can focus only on the rotation. The UI was edited and doesn't show you your positioning or rate of movement; it only shows the current rotation and the rate of rotation. In addition, it contains a timer that counts the time you have been stable and informs you of the fact you are in the tutorial. You start with the yaw angle, then the pitch angle, and finally, the roll angle. The rotation section of the tutorial can be seen in Figure 5.10.

5.11 Application build

Building the application for Android devices was simple. Unity provides a comprehensive tutorial on how to do this; I just had to follow it and make some decisions based on the nature of my application. The app was built for Android devices with a version of Android 5.1 Lollipop (API level 22) or higher.

The final size of the application was just under 59 MB, which is a bit for a simple simulator, but most of the size comes from the Earth and Moon textures. I already reduced their quality, and reducing it any more would make them look almost unbearable. As for the icon of the application, I chose a close-up of the Columbia spacecraft docking port. The icon can be seen in Figure 5.10.

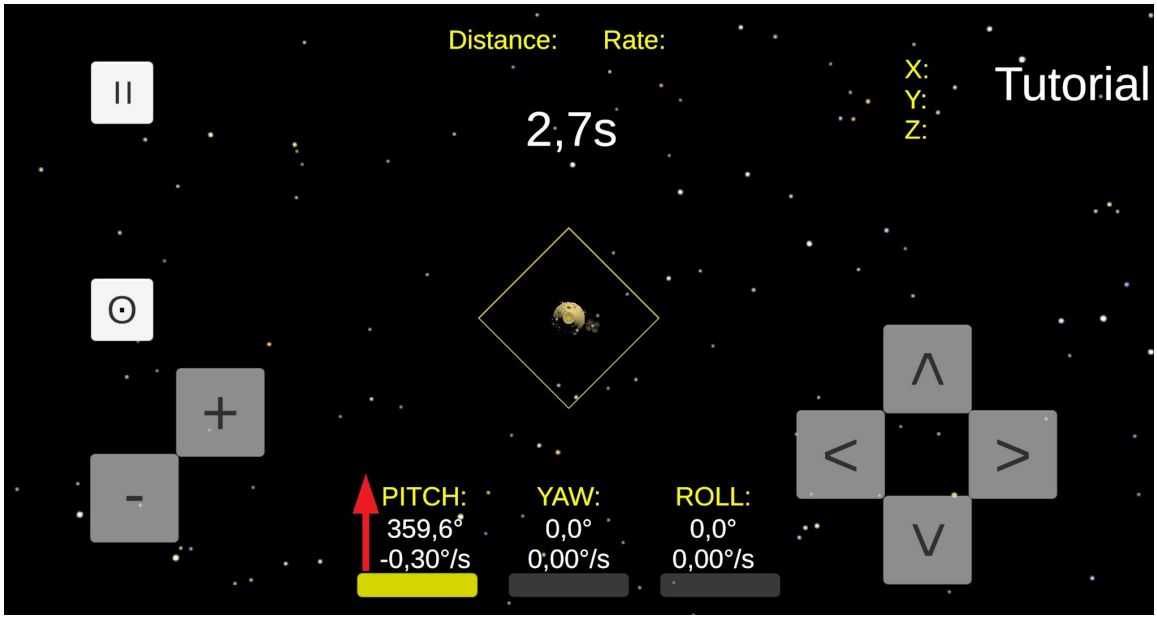

Figure 5.9: The rotation section of the tutorial

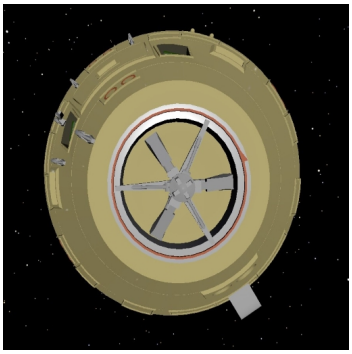

Figure 5.10: The icon of the application

Google Play

The plan was to upload the application to the Google Play Store, but I had to give up this plan. The reason is that the Google Play Store requires that at least twenty people test the application for fourteen days before it can be published. I still uploaded the application to the Google Play Store, but it is not available to the public, only to the people I invited and who have a link to it.

Chapter 6

Testing and evaluation

This chapter will describe the testing process of the simulator. Its main focus will be on user testing.

6.1 Unity Remote

During development, some of the testing was done using Unity Remote. Unity Remote is an application that allows you to test your Unity project on an Android device using a USB connection. It is a great tool for testing the application on a real device without building it every time you make a change. The current version I use is Unity Remote 5.

It is not the same as having the app built and then downloaded to your device. It just streams the view from the Unity Editor to the device and sends live inputs back. This meant that the app was not as responsive as it would be if it was built and downloaded. It also looked a little different than the real app would, and some changes had to be made after the app was officially built.

This application enabled the testing of the gyroscope, a capability not supported within the Unity editor. It was used throughout the development of the rotation controls.

6.2 User testing

User testing is an important aspect of any development process. It allows the developer to see how the users interact with the application and what problems they encounter. This project is no exception, and the users revealed problems with the simulator that were not apparent to me.

The main part of this section concentrates on the answers received from a form sent to the users after they had finished testing the simulator. The form had four open-ended questions: Did you have any difficulties using the simulator? Did you encounter any errors? What changes would you like to see? Do you have any other comments? These questions won't be discussed here but in the next chapter 7. This section will discuss the other ones.

Eleven people filled out the form in total. Their ages ranged from 20 to 24, and both men and women were represented. The testers were all students, and most had some experience with video games and simulators, but none knew much about the docking procedure.

The first question asked the users if they managed to finish the docking process successfully. The results can be seen in Figure 6.1. Most of the users, nine out of eleven,

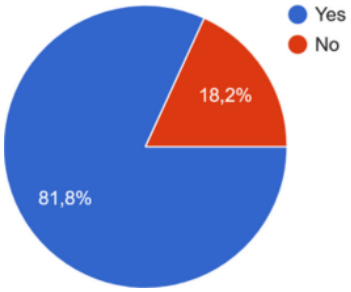

Figure 6.1: Did you successfully finish the docking process?

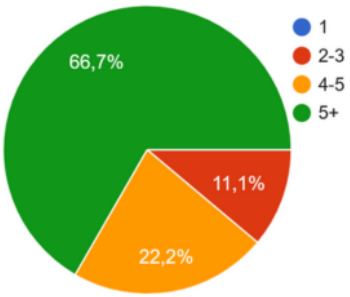

Figure 6.2: If yes, how many tries did it take?

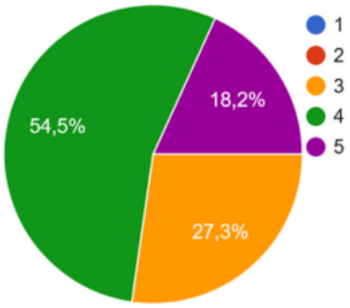

Figure 6.3: Rate the difficulty of the mission, 1 being the easiest

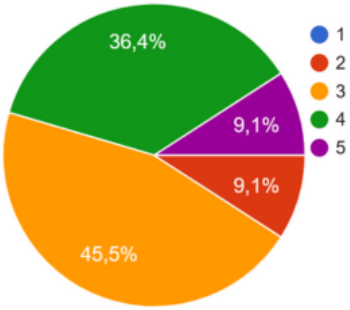

Figure 6.4: Rate the rotation controls, 1 being the best

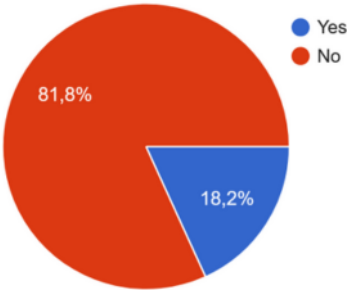

Figure 6.5: Would you understand the controls without the tutorial?

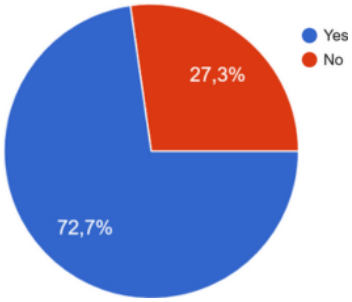

Figure 6.6: Did you find the simulator realistic?

successfully finished the docking process. The two who didn't finish were the ones who had the most problems with the rotation controls.

The next question concerned only the users who managed to finish and asked them how many tries it took them. The results can be seen in Figure 6.2. For the majority of the users, it took quite a lot of effort, with six of them being able to finish after only five or more tries. Nobody succeded on their first try, with only one being able to finish on their second or third try.

The results from previous questions hint that the mission was quite difficult, and the results agree. The users were asked to rate the difficulty on a scale from 1 to 5, with 1 being the easiest. As can be seen in Figure 6.3, the average rating was 3.9. This is a bit higher than I expected and shows that the simulator presented considerable challenges.

The rotation controls were next, rated on a scale from 1 to 5, with 1 being the best. The results can be seen in Figure 6.4. The average rating was 3.4, which shows that while the controls work, they are not ideal, and future work should concentrate on improving them.

The users were then asked if they would understand the controls without the tutorial. The results in Figure 6.5 show us that the tutorial is essential, and I am glad I decided to implement it.

An important part of the simulator is its realism. The users were asked if they found the simulator realistic. The results can be seen in Figure 6.6. Most of the users, eight

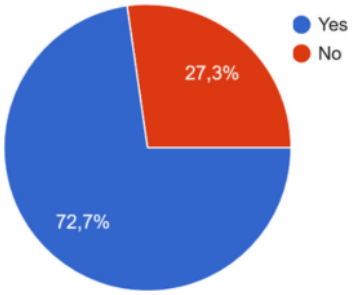

Figure 6.7: Was the UI intuitive?

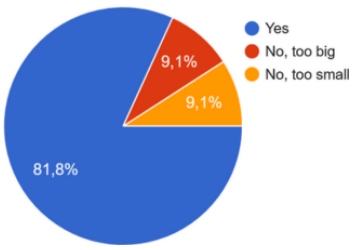

Figure 6.8: Were the buttons the correct size?

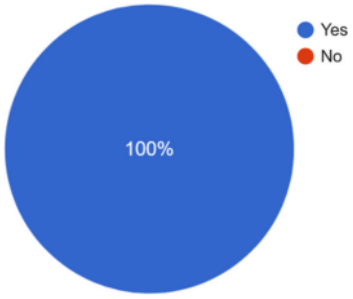

Figure 6.9: Were the statistics eligible?

out of eleven, found the simulator realistic, which is a good result that shows us that the implementation of our equations was not done in vain.

The last three questions were centered around the UI. They can be seen in Figures 6.8, 6.9, and 6.7. The results show that the users were satisfied with the UI and that no big changes had to be implemented.

6.3 Evaluation

Overall, the simulator demonstrated success in its main goal, which was to simulate the docking process. The users were able to finish the docking process, and most of them found the simulator realistic. The users were also satisfied with the UI, and no big changes had to be implemented. However, there were some problems as well. Some users had a hard time understanding and using the rotation controls. The mission's difficulty was rated high and proved frustrating for some.

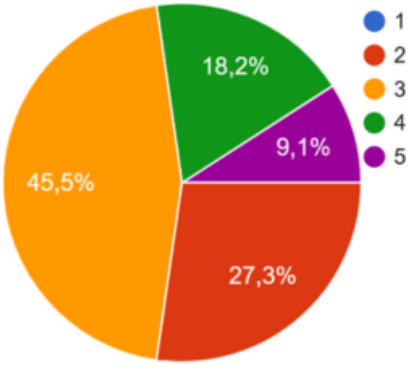

Figure 6.10: Rate your whole experience, 1 being the best

As we can see in Figure 6.10, most users were generally satisfied with their experience. I have some ideas on how to make improvements, and some were also suggested by the testers. These will be described in chapter 7.

Chapter 7

Future reaserch

This chapter will describe the docking simulator's possible future research and development. Although the users were overall satisfied, they provided some suggestions for improvement. This chapter will mainly focus on the answers to my form received during the user testing. We will concentrate on these four questions: Did you have any difficulties using the simulator? Did you encounter any errors? What changes would you like to see? Do you have any other comments?

7.1 UI

Although the users agreed that there was nothing wrong with my UI, they believed it could be improved. As this was my first time making a UI in Unity, I took a really simplistic approach, which was not ideal for a space-themed application. The UI can be seen in figure 5.4.

The users suggested that the UI should reflect a space-themed aesthetic, as the simulator is set in space. This improvement was mainly suggested for the buttons, as the ones used in my work are indistinguishable from others used in generic applications and do not evoke any associations with space travel.

The screen's borders could also be changed to reflect the space theme better. It would be nice if the user felt they were controlling and sitting inside a spacecraft. This could be achieved by adding some frame around the screen, which would be a part of the spacecraft's interior and invoke this feeling.

7.2 Movement

Rotation

The rotation controls were the most problematic part of the simulator. The users had a hard time understanding and using them, and most of the failures to dock were caused by them. The users complained that there is a very narrow window for some angles when the application detects rotation. This was especially true for the yaw angle.

I have two ideas that aim to improve this problem if not entirely resolve it. The first is to customize the threshold of detection for each angle individually. This would allow the user to rotate the device more freely in some directions and less in others and not do it the same way for every angle.

The second idea is to use the accelerometer. Currently, only the gyroscope is used to detect the device's rotation, but as the results show, this is not ideal. An accelerometer measures the acceleration of the device along its three axes, which can be used to detect changes in the device's rotation. This would allow for more precise measurements of the device's rotation and would make the rotation controls easier to use.

Thrusters

The thrusters in the simulator don't act like real thrusters, and their effects are simplified. Real thrusters have many nuances that have not been implemented in this project.

One of these is that thrusters output a minimum force, which makes precise maneuvering in real life harder than in this simulator. Thrusters also have a delay between the input and the output, and based on the propulsion system, this can be quite significant. The thrusters in this simulator output force instantly, which is not realistic. The last important detail is that thrusters consume fuel, which has also not been implemented.

All of these could be implemented in the future to make the process of on-orbit docking more realistic.

7.3 Difficulty

Some users found the missions available in the simulator too difficult to complete. Although this project aims to make the simulator realistic, this problem could still be tackled by creating a difficulty setting. There would be two difficulties: easy and realistic. The easy mode would then have laxer requirements for completing successful docking, which would hopefully mean the mission could be completed by anyone. If that didn't help, the controls could also be adjusted to produce lower force, making the process longer but more convenient.

7.4 Realisticalitty

While the simulator aims to be realistic and form the form, we can see that the users thought it was 6.6, and there are always ways to make it even more so. Some have been described before, like the better implementation of thrusters and some changes in the UI, but some were still missed.

The user might notice the absence of other space objects, primarily the Sun. Instead of using the light from the Sun, the simulator uses a simple light source that doesn't move. The Sun could be implemented to move around the scene, and its light could illuminate it. This would make the simulator more realistic and make the user feel like they are in space.

This simulator utilizes the CW equations to calculate the chaser spacecraft's movement, but that has some flaws. The CW equations system assumes the movement around a circular orbit. While the Hubble Space Telescope and the Columbia spacecraft were almost in a circular orbit, it was still a bit elliptical [25][24]. The CW equations also don't work for larger distances and velocities, but that is not a problem in this simulator if the user doesn't decide to take a trip to Mars [31].

The CW system is a first-order approximation, meaning it doesn't consider some of the forces affecting it. These include gravitational forces from other celestial bodies, atmospheric drag, solar radiant pressure, and others [31]. These forces could be implemented in the simulator to make it more realistic.

Chapter 8

Conclusion

The goal of this thesis was to examine the world of space exploration and translate knowledge gained into designing, implementing, and testing a simulator of the on-orbit docking process. The goal was successfully completed.

To accomplish our goal, we had to look back in time to the first docking procedures and follow the advancements in the field to the sophisticated automated systems of today. This journey is described in chapter 2. Then, I started the research into spacecraft dynamics modeling. Through a comprehensive review of literature and research, the key features of spacecraft dynamics modeling have been examined, revealing the physics and mathematics supporting successful docking maneuvers and how they can be implemented in our simulator to provide users with an immersive and realistic experience. These key features are described in chapter 3. After learning the necessary theory, the next step was to design the simulator. This included the game engine selection, with Unity being the chosen platform. The design process is described in chapter 4.

The culmination of this research and design is the implementation of an on-orbit docking simulator for the Android platform, described in chapter 5. This simulator serves as a practical tool for training and education, allowing users to experience the challenges and intricacies of docking in space firsthand. It takes advantage of Android's capabilities to provide an unusual and engaging control system centered around gyroscope usage. The simulator features two missions, each with unique environments and starting positions, a tutorial to help users understand the controls and the requirements for a successful docking process, and a comprehensive UI that provides the user with all the necessary information.

The testing and evaluation process confirmed its effectiveness in simulating docking procedures, giving the users an intuitive understanding of the challenges involved. The details are described in chapter 6.

Looking ahead, several promising avenues for future research have been identified either by the testers or by me. These include improvements to the UI, the addition of a difficulty setting, and the inclusion of more realistic features to the simulator. These are described in chapter 7.

Bibliography

[1] *Open Handset Alliance* [online]. [cit. 2024-01-24]. Available at: https://www.openhandsetalliance.com/.

[2] *Clohessy-Wiltshire Equations* [PDF]. 2013 [cit. 2024-01-24]. University of Texas at Austin. Available at: http://www.ae.utexas.edu/courses/ase366k/cw_equations.pdf.

[3] Adler, D. *The forgotten rescue of the Salyut 7 space station* [online]. 2020 [cit. 2024-01-24]. Available at: https://www.astronomy.com/space-exploration/the-forgotten-rescue-of-the-salyut-7-space-station/.

[4] Applications, N. V. T. and (VTAD), D. *Hubble Space Telescope 3D Model* [online]. 2019 [cit. 2024-04-09]. Available at: https://science.nasa.gov/resource/hubble-space-telescope-3d-model/.

[5] Brown, C. S. *What is Android? Here's everything you need to know* [online]. 2022. Available at: https://www.androidauthority.com/what-is-android-328076/.

[6] Byju's. *Angular Displacement Formula* [online]. [cit. 2024-01-24]. Available at: https://byjus.com/angular-displacement-formula/.

[7] Byju's. *Introduction to Torque and Its Applications* [online]. [cit. 2024-01-24]. Available at: https://byjus.com/physics/torque/.

[8] Cadnav. *Apollo Command and Service Module 3D Model* [online]. 2020 [cit. 2024-04-09]. Available at: https://www.cadnav.com/3d-models/model-49915.html.

[9] Callaham, J. *The history of Android: The evolution of the biggest mobile OS in the world* [online]. 2023. Available at: https://www.androidauthority.com/history-android-os-name-789433/.

[10] Cohen Peckam, E. *How Unity built the world's most popular game engine* [online]. 2019 [cit. 2024-01-24]. Available at: https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/?guccounter=1.

[11] Dallimore, G. *Real Stars Skybox Lite* [online]. 2019 [cit. 2024-04-12]. Available at: https://assetstore.unity.com/packages/3d/environments/sci-fi/real-stars-skybox-lite-116333.

[12] Donahoe, S. and Lewis, J. *International Docking System Standard* [online]. 2022 [cit. 2024-01-24]. Available at: https://www.internationaldockingstandard.com/.

[13] Ernie Wright (USRA), N. P. N. *CGI Moon Kit* [online]. 2019 [cit. 2024-04-09]. Available at: https://svs.gsfc.nasa.gov/4720.

[14] Google. *Android for Developers* [online]. [cit. 2024-01-24]. Available at: https://developer.android.com/.

[15] Google. *Motion sensors | Sensors and location* [online]. 2024 [cit. 2024-01-24]. Available at: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion.

[16] hanimomer. *Earth 3D Model* [online]. 2020 [cit. 2024-04-09]. Available at: https://free3d.com/3d-model/earth-94721.html.

[17] Herridge, L. *The Apollo-Soyuz Test Project: Success Achieved for First Rendezvous and Docking of Two Nation's Spacecraft in Space* [online]. 2020 [cit. 2024-01-24]. Available at: https://www.nasa.gov/history/the-apollo-soyuz-test-project-success-achieved-for-first-rendezvous-and-docking-of-two-nations-spacecraft-in-space/.

[18] Koblick, D. *Vectorized Clohessy-Wiltshire Hill Linear Propagation* [online]. 2012. MATLAB Central File Exchange. Available at: https://www.mathworks.com/matlabcentral/fileexchange/39340-vectorized-clohessy-wiltshire-hill-linear-propagation.

[19] Lovell, T. A. and Spencer, D. A. Relative Orbital Elements Formulation Based upon the Clohessy-Wiltshire Equations. *The Journal of the Astronautical Sciences*. December 2014, vol. 61, no. 4, p. 341–366. DOI: 10.1007/s40295-014-0029-6. ISSN 2195-0571.

[20] Mr. Floyd I. Roberson, M. W. F. E. *Apollo 11 Lunar Module / EASEP* [online]. 2022 [cit. 2024-04-21]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1969-059C.

[21] NASA. *Apollo Lunar Module Propulsion Systems Overview* [online]. [cit. 2024-04-21]. Available at: https://ntrs.nasa.gov/api/citations/20090016298/downloads/20090016298.pdf.

[22] NASA. *Apollo 10: Mission Details* [online]. 2009 [cit. 2024-01-24]. Available at: https://www.nasa.gov/missions/apollo/apollo-10-mission-details/.

[23] NASA. *Apollo 9: Mission Details* [online]. 2009 [cit. 2024-01-24]. Available at: https://www.nasa.gov/missions/apollo/apollo-9-mission-details/.

[24] NASA. *Apollo 11 Mission Overview* [online]. 2015 [cit. 2024-04-12]. Available at: https://www.nasa.gov/history/apollo-11-mission-overview/.

[25] NASA. *Hubble FAQs* [online]. 2023 [cit. 2024-04-12]. Available at: https://science.nasa.gov/mission/hubble/overview/faqs/.

[26] NASA. *International Space Station* [online]. 2023 [cit. 2024-01-24]. Available at: https://www.nasa.gov/reference/international-space-station/.

[27] NASA. *Station Facts* [online]. 2023 [cit. 2024-01-24]. Available at: https://www.nasa.gov/international-space-station/space-station-facts-and-figures/.

[28] OUSTTRUE. *UniGLTF* [GitHub repository]. 2018 [cit. 2024-04-09]. Available at: https://github.com/ousttrue/UniGLTF?tab=readme-ov-file.

[29] PARTNERS, I. P. *International Docking System Standard (IDSS) Interface Definition Document (IDD)*. NASA, July 2022. Available at: https://www.internationaldockingstandard.com/download/IDSS_IDD_Revision_F.pdf.

[30] SKETCHFAB. *Sphere* [online]. 2019 [cit. 2024-04-09]. Available at: https://sketchfab.com/3d-models/sphere-b31b12ffa93a40f48c9d991b6f168f4d.

[31] STAREK, J. A., SCHMERLING, E., MAHER, G. D., BARBEE, B. W. and PAVONE, M. Fast, Safe, and Propellant-Efficient Spacecraft Planning under Clohessy-Wiltshire-Hill Dynamics. *Journal of Guidance, Control, and Dynamics*. American Institute of Aeronautics and Astronautics. February 2017, vol. 40, no. 2, p. 418–438. DOI: 10.2514/1.G001913. ISSN 0731-5090.

[32] STATCOUNTER. *Mobile Operating System Market Share Worldwide* [online]. 2024 [cit. 2024-01-24]. Available at: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[33] TIOBE. *TIOBE Index for January 2024* [online]. 2024 [cit. 2024-01-24]. Available at: https://www.tiobe.com/tiobe-index/.

[34] TURNER, A. *How Many Apps In Google Play Store? (Jan 2024)* [online]. 2024 [cit. 2024-01-24]. Available at: https://www.bankmycell.com/blog/number-of-google-play-store-apps/.

[35] UNITY. *Unity User Manual 2022.3 (LTS)* [online]. [cit. 2024-01-24]. Available at: https://docs.unity3d.com/Manual/index.html.

[36] UPPAL, R. *The Future of Spacecraft Servicing: How Autonomous Docking is Revolutionizing Space Operations* [online]. 2023 [cit. 2024-01-24]. Available at: https://idstch.com/space/the-future-of-spacecraft-servicing-how-autonomous-docking-is-revolutionizing-space-operations/.

[37] URI, J. *55 Years Ago: The Spirit of 76 - The First Rendezvous in Space - NASA* [online]. 2020. Available at: https://www.nasa.gov/history/55-years-ago-the-spirit-of-76-the-first-rendezvous-in-space/.

[38] URI, J. *50 Years Ago: Remembering the Crew of Soyuz 11* [online]. 2021 [cit. 2024-01-24]. Available at: https://www.nasa.gov/history/50-years-ago-remembering-the-crew-of-soyuz-11/.

[39] URI, J. *55 Years Ago: Gemini VIII, the First Docking in Space* [online]. 2021 [cit. 2024-01-24]. Available at: https://www.nasa.gov/history/55-years-ago-gemini-viii-the-first-docking-in-space/.

[40] URI, J. *60 years ago: NASA Decides on Lunar Orbit Rendezvous for Moon Landing* [online]. 2022 [cit. 2024-01-24]. Available at: https://www.nasa.gov/history/60-years-ago-nasa-decides-on-lunar-orbit-rendezvous-for-moon-landing/.

[41] URONE, P. P., HINRICHS, R., GOZUACIK, F., PATTISON, D. and TABOR, C. *Physics for High School*. XanEdu publishing, 2020. ISBN 978-1-975076-51-1.

[42] Weisstein, E. W. *Rotation matrix* [online]. [cit. 2024-01-24]. From MathWorld–A Wolfram Web Resource. Available at: https://mathworld.wolfram.com/RotationMatrix.html.

[43] Williams, D. R. *NASA - NSSDCA - Spacecraft - Details - Apollo 11 Command and Service Module (CSM)* [online]. 2022 [cit. 2024-01-24]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1969-059A.

[44] Williams, D. R. *NASA - NSSDCA - Spacecraft - Details - Apollo 11 Lunar Module / EASEP* [online]. 2022 [cit. 2024-01-24]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1969-059C.

[45] Williams, D. R. *NASA - NSSDCA - Spacecraft - Details - Gemini 6* [online]. 2022 [cit. 2024-01-24]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=GEM6.

[46] Williams, D. R. *NASA - NSSDCA - Spacecraft - Details - Gemini 6A* [online]. 2022 [cit. 2024-01-24]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1965-104A.

[47] Williams, D. R. *NASA - NSSDCA - Spacecraft - Details - Salyut 1* [online]. 2022 [cit. 2024-01-24]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1971-032A.

[48] Williams, D. R. *NASA - NSSDCA - Spacecraft - Details - Salyut 7* [online]. 2022 [cit. 2024-01-24]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1982-033A.

[49] Williams, D. R. *NASA - NSSDCA - Spacecraft - Details - Soyuz 10* [online]. 2022 [cit. 2024-01-24]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1971-034A.

[50] Williams, D. R. *NASA - NSSDCA - Spacecraft - Details - Soyuz 4* [online]. 2022 [cit. 2024-01-24]. Available at: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1969-004A.

Appendix A

Questionaire

1. Did you successfully finish the docking process?
    - Yes
    - No
2. If yes, how many tries did it take?
    - 1
    - 2-3
    - 4-5
    - 5+
3. Rate the difficulty of the mission, 1 being the easiest
    - Numbers 1-5
4. Rate the rotation controls, 1 being the easiest
    - Numbers 1-5
5. Would you understand the controls without the tutorial?
    - Yes
    - No
6. Did you find the simulator realistic?
    - Yes
    - No
7. Was the UI intuitive?
    - Yes
    - No
8. Were the buttons the correct size?
    - Yes

- No, too big
- No, too small

9. Were the statistics eligible?
    - Yes
    - No
10. Rate your whole experience, 1 being the best
    - Numbers 1-5
11. Did you have any difficulties using the simulator? If yes, describe them.
    - No
    - Other
12. Did you encounter any errors? If yes, describe them.
    - No
    - Other
13. What changes would you like to see?
    - Open-ended question
14. Do you have any other comments?
    - Open-ended question