SatSim: A Program Demonstrating a Novel Routing Algorithm for LEO satellites

Tobias James Tomkinson, BSc
School of Computer Science
University of Auckland

A dissertation submitted for the degree of
Computer Science Honours

September, 2023

SatSim: A Program Demonstrating a Novel Routing Algorithm for LEO satellites

Tobias James Tomkinson, BSc.
School of Computer Science, Auckland University

A dissertation submitted for the degree of *Computer Science Honours*. September, 2023.

Abstract

This dissertation presents a detailed explanation on the development and implementation of a novel pathfinding algorithm for Low Earth Orbit satellite networks. The paper begins with an introduction, highlighting the problem statement and defining key terms. Background knowledge on satellites, coordinate systems, great circle navigation, and more are provided to establish a solid foundation to enable full understanding of the pathfinding algorithm, followed by a thorough literature review. The literature review includes the research aims ("what existing routing methods exist?" and "have they ever been implemented?"), selection process (including requirements and specifications), and a general collection of information related to inter-satellite communication. The algorithm itself is presented with both code and pseudocode, and is accompanied by a step-by-step explanation of its workings. The implementation section discusses data reading, conversion between coordinate systems, interface scripts, and the actual implementation of the pathfinding algorithm in regards to the simulation software. Finally, the conclusion highlights the results that have been achieved and identifies any remaining unsolved problems.

Declaration

I declare that the work presented in this dissertation is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This dissertation does not exceed the maximum permitted word length of 12,000 words including appendices and footnotes, but excluding the bibliography. A rough estimate of the word count is: **7237 words**

Tobias James Tomkinson

Contents

Chapter 1

Introduction

This dissertation demonstrates a new inter-satellite routing algorithm and demonstrates the algorithm using software that outputs the time taken for data transmission and length of the path traveled between two points while following a Great Circle Path.

1.1 Statement of the problem

Starlink, the satellite internet provider, currently only has a limited and relatively slow inter-satellite routing system in place. This means that certain locations that are remote and places with reduced Starlink coverage have limited access to the internet. However, with the implementation of an inter-satellite routing system, Starlink could potentially provide more robust coverage to a much wider customer base.

To address this issue, a novel inter-satellite routing algorithm has been developed and its effectiveness is examined in this dissertation. The algorithm takes into account the shortest path between two points on the Earth's surface and calculates the optimal routing path between Starlink satellites. By doing so, this new routing algorithm offers an effective way to expand and improve Starlink coverage, especially in areas where traditional ground-based internet infrastructure is either non-existent or limited, such as remote pacific islands or the University of Auckland.

Chapter 2

Background Knowledge

2.1 Satellites

2.1.1 Applications

Satellites play a crucial role in various applications, serving a wide range of purposes that include weather forecasting, television signals, amateur radio, internet communications, and the Global Positioning System (GPS), among others.

Weather Satellites are extensively used in weather forecasting. They provide valuable data and imagery that enable meteorologists to track weather patterns, monitor storms, and gather information about atmospheric conditions across the globe.

Television Satellite television relies on the use of geostationary satellites. These satellites are positioned in geostationary orbits, which means they remain fixed relative to a specific location on Earth's surface. This stationary positioning allows for the reliable transmission of television signals to a large geographical area.

Radio Satellite-based radio communication is another significant application. Satellites facilitate long-distance communication for amateur radio enthusiasts, broadcasting stations, and emergency services. By relaying radio signals, satellites enable the transmission of audio content across vast distances.

Internet Satellites also play a vital role in providing internet connectivity, particularly in remote and underserved areas. Satellite internet communication involves

the transmission of data signals to and from satellites in orbit, enabling users to access the internet regardless of their geographical location.

GPS The Global Positioning System (GPS) heavily relies on satellites to determine precise location information. GPS satellites, typically situated in medium Earth orbit (MEO), continuously transmit signals that GPS receivers on Earth use to triangulate their position. This technology is widely used in navigation systems, mapping applications, and various location-based services.

2.1.2 Satellite Terminology

Orbits An orbit is a fundamental concept in astronomy and astrophysics, referring to the path that an object, such as a satellite, planet, or celestial body, follows around another larger body due to the force of gravity. This concept has been widely studied and applied in various fields, including space exploration, satellite technology, and planetary motion.

In an orbit, the smaller object is continuously pulled toward the larger body by the force of gravity. However, because of its initial velocity perpendicular to the direction of gravity, the smaller object is also moving sideways. This combination of gravitational attraction and sideways motion results in a curved path around the larger body. As a result, the object never falls directly into the larger body but maintains a stable path around it.

The shape of the orbit can vary depending on factors such as the velocity of the object, the mass of the larger body, and the initial angle of launch. The most common types of orbits include circular orbits, elliptical orbits, and parabolic orbits.

Circular Circular orbits are symmetrical and maintain a fixed distance from the larger body at all times. An example of this is a geostationary satellite orbit, where a satellite remains stationary relative to a point on Earth's surface.

Elliptical Elliptical orbits are more elongated and have varying distances from the larger body. Planets in our solar system follow elliptical orbits around the Sun, and so do many artificial satellites.

Parabolic Parabolic orbits occur when an object has enough velocity to escape the gravitational pull of the larger body, resulting in a path that approaches infinity.

This often occurs during interplanetary missions, where spacecraft are launched with enough speed to break free from a planet's gravity.

Orbital Plane An orbital plane can be described as orbit's angle that crosses through the Earth's center. To determine this plane, you only need three points in space that are not in a straight line. One example is the positions of a large object's center (like a host planet) and a celestial body's center that is orbiting it, observed at different times in its orbit [1].

This orbital plane is described using two factors in relation to a reference plane: inclination and the longitude of the point where the orbit crosses the reference plane - simply put, the rotation relative to Earth.

Inclination Inclination refers to the tilt or angle of a satellite's orbit relative to the Earth's equator. It measures how much the satellite's orbital plane deviates from being directly above the equator.

The inclination of a satellite's orbit affects its coverage area and the regions it can effectively serve. Satellites with low inclinations cover a larger portion of the Earth's surface between the latitudes near the equator. As the inclination increases, the satellite's coverage area extends towards the higher latitudes, providing service to more polar regions.

The inclination of a satellite's orbit is typically measured in degrees. A satellite in a zero-degree inclination orbit, known as an equatorial orbit, travels directly above the Earth's equator. Satellites in polar orbits have inclinations of 90 degrees, which means they pass over or near the Earth's poles.

Orbital Control Orbital control refers to the methods and techniques employed to manage and adjust the trajectory and position of satellites in space. Precise orbital control is crucial for various reasons, including maintaining the desired orbit height, correcting orbital deviations, and ensuring proper positioning for the intended mission objectives.

Orbital control maneuvers involve altering a satellite's speed, direction, or both to achieve the desired changes in its orbit. These maneuvers can be performed using propulsion systems onboard the satellite, which can either be chemical rockets or electric propulsion systems. By carefully executing these maneuvers, satellite operators can optimize orbital parameters, counteract the effects of atmospheric drag, and counterbalance perturbations caused by gravitational interactions with other celestial bodies.

Orbital control is essential for maintaining the long-term stability and operational efficiency of satellite systems. It allows for orbit maintenance, inclination adjustments, formation flying (coordination of multiple satellites in close proximity), and orbital debris avoidance, among other activities.

Orbit Height The orbit height of a satellite refers to its distance from the Earth's surface. Different satellite applications require specific orbit heights to fulfill their intended functions effectively. Accurate control of orbit height is crucial for satellite systems to achieve their operational goals, optimize coverage, and ensure efficient use of resources.

LEO Low Earth Orbit (LEO) satellites are defined as satellites that orbit the Earth at an altitude below 2,000 kilometers above the Earth's surface [2]. These satellites are used for a variety of different purposes, such as communication, navigation, and scientific research. Unlike geostationary satellites, which orbit at a fixed position above the equator, LEO satellites travel at high speeds and can complete a full orbit of the Earth in less than 128 minutes. Due to their physical proximity to the Earth, LEO satellites have low latency and offer high-bandwidth communication services, making them ideal for satellite internet providers such as EchoStar, Gravity Internet, Starlink, and others [3].

MEO Also known as medium Earth orbit or intermediate circular orbit, MEO is a region of space roughly located between 2,000 and 36,000 kilometers from the Earth's surface. MEO is commonly used for satellite navigation systems, such as GPS. Satellites in MEO orbits provide wider coverage compared to LEO satellites, making them suitable for global positioning and timing services.

GEO Geostationary orbit is 35,786 kilometers above the Earth's surface. Satellites in GEO have a fixed position relative to a specific point on the planet's surface. Due to said fixed position, these satellites are often used for applications that require continuous coverage over a specific region, such as satellite broadcasting, telecommunication services, and weather monitoring. However, the longer signal travel distance and higher latency make GEO satellites less suitable for real-time interactive applications.

Satellite Constellations A satellite constellation is a group of artificial satellites working together as a system [4]. These constellations are usually made up of satel-

lites in medium Earth orbit (MEO) or low Earth orbit (LEO) because the coverage provided by a single satellite is limited - a constellation of satellites working together can provide consistency, which is especially useful for communication services that require continuous global coverage. In LEO, a single satellite's coverage area is limited because it moves quickly as it orbits around the Earth. As a result of this, to ensure that a significant portion of the Earth's surface is covered at all times then satellite constellations are required. This is why Starlink, a satellite internet service provided by SpaceX, requires a vast network of satellites.

Satellite constellations are used in a variety of applications; for example, the Global Positioning System (GPS) is a satellite constellation that provides highly accurate location data for navigation. Other examples include the Iridium and Globalstar satellite telephony services, as they also use satellite constellations to provide reliable and high-quality communication services to remote areas around the world.

2.1.3 Communication Terms

Laser Communication Laser communication between satellites involves the utilization of inter-satellite laser links. This method falls within the domain of free-space optical communication, and is an acronym for "Light Amplification by Stimulated Emission of Radiation" (LASER).

Laser history The longest recorded instance of laser communication in space involves transmitting over a distance of 24,000,000 kilometers to the MESSENGER spacecraft [5]. However, inter-satellite communication usually occurs over significantly shorter distances - a few hundred kilometers at most. The first instance of inter-satellite communication took place in 2001, and since then laser communication has become more commonplace and usable.

ISL Inter-Satellite Links, or ISL, are wireless communication links established between different satellites within the same satellite constellation. These links allow satellites to directly exchange data with one another without the need to relay through ground stations - as a result, ISLs are relevant in the context of this paper, and indeed satellite communications in general.

Satellites equipped with ISL technology use various communication mechanisms, including radio frequency (RF) links, optical links or laser-based communication,

depending on the specific needs of the satellite and the distance between the satellites [6]. These communication links can operate in different frequency bands, allowing for flexible and efficient data transmission. For the sake of convenience and understanding, specific frequencies (and other hardware specific details) are not covered in this paper's algorithm.

ISLs enable coordinated actions among satellites, leading to greater coverage, alternative communication paths if ground stations experience disruptions due to adverse weather (or other factors) and generally ensure continuous communications.

Intra- vs Inter-Plane Intra-plane routing involves managing the communication of satellites within a shared orbital plane. Inter-plane refers to communication between different orbital planes. Do note that any communication involved would be inter-satellite communication, not intra-satellite as there is no communication involved within a single satellite.

UDL The User Data Link, or UDL, is the communication link between the satellite and the end users/ground stations. The UDL is responsible for transmitting user-generated data collected by the satellite's payload to designated ground stations or user terminals.

As a couple of examples; remote sensing satellites primarily use specific frequencies to transmit large volumes of data collected from Earth, while satellites providing internet services would utilize higher frequency bands for higher data transfer rates [7]. A UDL also facilitates bi-directional communication, allowing ground stations to send commands and control data to the satellite such as adjusting orbits, or managing power consumption.

IOL Inter-Orbit Links, or IOL, are communication links that connect satellites operating in different orbits, allowing them to exchange data and cooperate across orbits. This communication capability is especially relevant in satellite constellations where satellites are deployed in multiple orbits, each with specific advantages for certain mission objectives.

IOL can be established using various technologies, including RF links, optical communication, or even advanced laser communication systems. The choice of technology depends on factors such as the distance between satellites, the data transfer rate required, and the availability of line-of-sight connections between satellites in different orbits [7].

The primary functions of Inter-Orbit Links include data relaying and backhaul; IOLs enable the transfer of data collected by satellites in lower orbits to satellites in higher orbits, which can then relay the information to ground stations. This backhaul capability is very useful for efficient data management and distribution across the entire satellite network. Much like the aforementioned ISLs, IOLs can also be used to redistribute data and workload to other operational satellites in the constellation in the event of a partial network failure, ensuring constant communication.

Backhaul Backhaul is the process of sending data or communication signals from a remote or local network to a central network over long distances [8]. Its main purpose is to gather data traffic from various remote locations or access points and transfer it to a central hub or data center for further distribution and connection to the broader network infrastructure.

Tag switching Tag switching, also known as marker switching, is a networking method where incoming packets are assigned tags containing information about source, destination, and priority. These tags allow for more efficient packet forwarding as routers/switches/etc. are able to make routing decisions based on the tags rather than the entire packet or the overall approach. This improves network performance and optimizes resource usage.

2.1.4 Starlink

Starlink is a satellite internet constellation that is owned and operated by the space-focused company, SpaceX [9]. This network of satellites has been developed to improve the way people can access the internet, providing connectivity to various areas of the globe with a focus on traditionally inaccessible areas.

With approximately 4983 satellites in orbit [10] as of writing, Starlink is providing a substantial increase in global connectivity - access to internet is no longer limited by geographical location and most infrastructure constraints.

Starlink has been designed to function in multiple “shells”, with collections of satellites operating at different heights. Three are in varying stages of completion - the first at a height of 340 km, the second at 550 km, and the third and final at 1,150 km [9]. The FCC approved Starlink’s request, during early April of 2019, for a total of approximately 12,000 satellites to be placed in orbit; 7,500 at 340km, 1,600 at 550km and 2,800 at 1,150km [11].

2.2 Coordinate Systems

2.2.1 Geographic Coordinate System

The Geographic Coordinate System (GCS) is a coordinate system that uses two variables, latitude and longitude, to indicate a location on the earth's surface [12]. This coordinate system enables accurate location tracking and mapping of physical features on the planet and is used in various applications, such as navigation systems and geolocation services.

2.2.2 Cartesian Coordinate System

However, using GCS in Unity Engine requires modifications since Unity Engine uses the Cartesian Coordinate System, or CCS [13]. Unlike GCS, which uses angles to describe locations, CCS uses three values to represent points relative to a central vector of (0,0,0), which is called the origin.

The Simplemaps dataset [14] that is used in this dissertation uses GCS. Therefore, to use GCS data in Unity, it is necessary to convert between the two coordinate systems (see section 6.2 for implementation).

2.3 Great Circle Navigation

2.3.1 Explanation

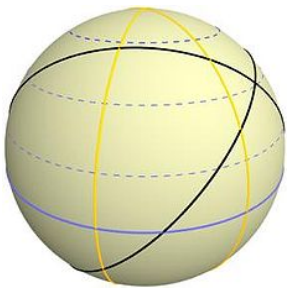

Figure 2.1: A pair of orthodromic routes on a globe

Great circle navigation, also known as orthodromic navigation, is a technique used to determine the shortest path between two points on a globe [15]. The shortest path between two points on the surface of a sphere is always along a great circle, which is defined as the intersection of the sphere and a plane passing through its center and the two points.

The mathematics involved in great circle navigation are somewhat complicated and require a more intimate understanding of spherical trigonometry than is needed for this dissertation; the concept can be simplified to "a way to minimize distance and time in navigation". By taking the shortest path

possible between two points, great circle navigation can save time and fuel, making it a valuable tool in various fields such as shipping, aviation, and now even network packets.

2.3.2 History of Great Circle Navigation

2.4 Data Structures & Algorithms

This paper utilizes various algorithms and data structures, including:

- Satellite Orbit Propagation: The software employs basic algorithms to simulate the movement and position of satellites over time. This involves calculations based on orbital elements, including Keplerian parameters, to model the satellite orbits.
- Visualization Techniques: The software incorporates visualization algorithms and techniques, such as 3D rendering and graphical representations, to provide users with a visual representation of the satellite constellations and their routing paths. Fortunately, the majority of the "heavy lifting" is done by Unity Engine.

These algorithms and data structures work together to enable the software to simulate and analyze routing effectively - see Section 4.1 for more.

2.4.1 Routing Algorithms

Bellman-Ford Bellman-Ford's shortest path algorithm is used to find the shortest paths from a single source vertex to all other vertices in a weighted graph [16]. This algorithm can handle graphs with both positive and negative edge weights but is designed to work correctly with graphs that do not contain negative cycles.

The algorithm operates based on the principle of dynamic programming and is iterative in nature. It gradually refines estimates of the shortest path distances until the optimal shortest paths are found.

Dijkstra's Algorithm Dijkstra's algorithm is a widely used algorithm in computer science and graph theory for finding the shortest path between a specified starting node and all other nodes in a weighted graph. The algorithm works based on the principle of iteratively exploring the graph from the starting node outward, gradually building the shortest path tree as it progresses.

Dijkstra vs Bellman-Ford Both algorithms aim to determine the shortest distance from a source node to any other node in the graph - However, they're different in their approaches and their computational characteristics. Dijkstra's algorithm is an algorithm that operates efficiently on graphs with non-negative edge weights, as it always selects the node with the smallest distance to "explore" next. In contrast, the Bellman-Ford algorithm is a dynamic approach that accommodates graphs with negative edge weights, but requires multiple iterations to ensure a stable route.

Novel Algorithm To determine the most efficient route for satellite communication, the software employs a shortest path algorithm using a Great Circle Path, or GCP. This is explained in greater detail later in the paper.

Now that all of the background information has been covered and context has been provided, we can move on to the literature review.

Chapter 3

Literature Review

3.1 Aims of Research & Research Questions

The main goal of this review is to conduct an investigation into the various routing methods used for low earth orbit satellites. Said research is aimed at determining the feasibility of these methods in terms of their implementation, and whether they have already been deployed. There are two key research questions that are hoped to be resolved in this review:

1. What is the practicality of the existing routing algorithms?

2. Have said routing algorithms been implemented, either virtually (in software) or physically (in a satellite constellation)?

3.2 Search & Paper Selection Process

To gather relevant information for the research, a comprehensive literature search was conducted using various academic databases and online repositories - peer-reviewed journals and conference proceedings were examined to ensure the credibility and accuracy of the sources.

As research regarding inter-satellite links originated in 1977 [17], it was assumed that avoiding filtering by year would be acceptable, as the approaches would remain relevant regardless of the publication date (at least in terms of software).

After the initial search, specific inclusion and exclusion criteria were applied to select papers for further review. The inclusion criteria focused on papers written in English that were directly relevant to satellite routing algorithms. Papers that were

not in English, were not peer-reviewed, and did not directly address the research questions mentioned earlier were excluded.

3.3 Outcome of Search, Validity, Aggregation & Synthesis of Information

In the course of this research, several relevant papers related to satellite routing algorithms for multilayered satellite networks and LEO constellations were identified and analyzed. The papers covered various routing strategies and techniques that have implications for LEO satellites operation and the simulation software presented in this paper.

One of the papers, "MLSR: a novel routing algorithm for multilayered satellite IP networks" by Akyildiz et al. [18], introduced the Multiple Layered Satellite Routing (MLSR) algorithm, which utilizes multiple layers (LEO, MEO and GEO) for routing in a satellite network. Although Starlink currently employs only LEO satellites, the MLSR algorithm may become more relevant for potential partnerships or expansions in the future, though it is also becoming increasingly relevant as StarLink is installing multiple shells at different altitudes, though all in LEO. Akyildiz et al.'s work demonstrates comparable performance to the Bellman-Ford shortest path routing algorithm, except during a short oscillatory phase when the communication path switches to a higher satellite layer.

Similarly, Song et al.'s conference paper, "A novel unicast routing algorithm for LEO satellite networks" [19], presented a dynamic source routing algorithm (DSRA) that aims to find the shortest route using weights to determine the most optimal satellite to hop to within the same plane. In principle, Starlink could be employing a similar dynamic routing mechanism to optimize data transmission between satellites within the same orbital plane. Both DSRA and Starlink's speculated dynamic routing scheme take into consideration weighted metrics, such as signal strength, available bandwidth, and satellite health status, to identify the most efficient satellite hops for data transmission.

The paper by Roth et al., "Implementation of a geographical routing scheme for Low Earth Orbiting satellite constellations using intersatellite links" [20], proposed a routing approach based on geographical address identifiers, derived from MAC addresses and Dijkstra's algorithm. This approach allows for dynamic ground stations, which is similar to what can be observed of Starlink's geographical routing and MAC address utilization to facilitate communication between satellites and

ground stations. Although specific details of Starlink's implementation are not publicly available, the similarities in the routing approach suggest relevance to Starlink's operation.

The literature review conducted by Li et al. provided valuable insights into current algorithms, risk management methods, and applications of approaches towards LEO routing [21]. It highlights the significance of differentiated service routing mechanisms and the general reduction of end-to-end delay in LEO-based satellite networks. Notably, the review referenced Gounder et al.'s snapcut-based satellite network routing algorithm [22], which explores a virtual connection-oriented approach similar to tag switching (see 2.1.3), considering memory constraints of the satellites. This aligns with the novel algorithm demonstrated in this dissertation, albeit operating at a different communication layer.

Additionally, Li et al. cites Zhu et al.'s work [23], which proposes a software-defined routing algorithm that leverages real-time congestion information to determine the most optimal route within a satellite network. The concept of real-time adaptation to network conditions is particularly relevant, as it could enhance the algorithm developed in this dissertation and potentially lead to improved performance in LEO satellite networks.

3.4 Results & Limitations

The studies' main limitations include reliance on software simulations, neglecting some factors like power consumption and network resilience, and limited exploration beyond LEO satellite constellations [18]–[21]. Updated research and hardware-based experiments are needed to address these concerns and ensure the algorithms practicality and adaptability, though for the sake of this dissertation, this will not be explored further as it would be quite impractical given the scope of such testing.

Another limitation observed in some studies, such as the work of Gounder et al. [22], is the consideration of specific communication layers for routing decisions. While the use of tag switching or similar virtual connection-oriented approaches may optimize memory usage on satellites, it could also introduce additional complexity in network management and potentially limit scalability.

Currently, most of the algorithms that were surveyed compare themselves to either Dijkstra's algorithm or the Bellman-Ford shortest path routing algorithm, which have their own set of limitations. In three-dimensional space, there can be issues such as satellite orientation, as even with a rotating head it can only point

in one direction at a time. These two algorithms do not naturally handle such constraints and would require additional modifications or pre-processing steps to handle these obstacles effectively. This is avoided by assuming omni-directional communication in the potential hardware alongside calculating the route satellite by satellite, as opposed to limiting the path to using inter- and intra-plane communications.

Furthermore, the number of potential paths between two points in three-dimensional space (given the fact that Starlink has multiple layers of LEO satellites in its constellation) can increase significantly compared to a two-dimensional plane. As a result, the complexity of the aforementioned algorithms can increase, impacting their performance and making them less efficient for larger graphs/satellite constellations.

Furthermore, another issue rises when the sun is located behind satellites. Due to the sun's brightness, which is much greater than a given satellite's optical lasers, and powerful broadband radio frequency sources, the resulting interference with communication systems leads to a significant rise in the noise floor. Again, this is an issue that in principle needs to be compensated for. However, it is important to note that the complex modeling that would be required for this concern is too complicated to be included in this dissertation.

With the literature review concluded, we can move on to the algorithm itself.

Chapter 4

Pathfinding Algorithm

The algorithm detailed in this chapter outlines a method of finding the shortest routes between two points within the coverage area of a satellite constellation. This algorithm is a key component of the simulation software. It ensures that data is relayed between satellites efficiently to establish a clear path of communication.

4.1 Explanation

The algorithm iterates through different satellites in the constellation and runs the same commands on each satellite until it reaches a suitable candidate for downlinking to the destination ground position, also known as the end point. It first gathers the N closest satellites (that can be set by the user) to the current satellite as an array of objects which is referred to as *distanceSorted*. It then compares the direction between the current satellite and the great circle path to the end location, followed by the current satellite and *distanceSorted*. If the angle between the current satellite and the direction to the ground station is less than 40 degrees from the ground station, then the next point chosen is the ground station and the algorithm terminates. If the angle between the ground station and satellite is greater than 40 degrees - if the ground station cannot "see" the satellite - then the satellite simply chooses the nearest option from *distanceSorted* as the current satellite and repeats execution of the algorithm from there.

In regards to actual hardware-based implementation, each satellite in a given constellation must be equipped with Inter-Satellite Links (ISLs), which would enable the transfer of data that is required during the iteration process. For the distance calculation and sorting, onboard sensors such as GPS receivers and inertial measurement units would provide accurate position and orientation data. In

addition to this, the laser communication technology mentioned earlier would be the primary method of transmitting commands and data between satellites, and the User Data Link (UDL) comes into play for interactions with ground stations. The coordination between satellites, guided by ISLs and laser communication, ensures the propagation of commands and data across the constellation and the successful execution of the routing algorithm.

4.1.1 Clarification

The algorithm itself is executed first from the starting ground station, then each satellite. The great circle path is calculated as well as a direction to the intended destination which is computes on the basis of the destinations IPv6 address which contains an identifier for the ground station location. The ground station picks a satellite to up-link and from there each satellite tries to find a suitable satellite along the great circle path until it finds a satellite that is suitably close and visible to the end location ground station - it will down-link from there.

The algorithm proceeds as follows:

Pseudocode

1: **function** FINDROUTE(*startPoint*, *endPoint*)
2: *satellitePoints* ← empty list of *SatelliteScript* objects
3: *totalSatellites* ← all *SatelliteScript* objects
4: *currentSatellite* ← closest satellite to *startPoint*
5: remove *currentSatellite* from *totalSatellites*
6: *distanceSorted* ← *n* closest satellites sorted by distance from *currentSatellite*
7: *nextSatellite* ← satellite with minimum angle between *currentSatellite*, *nextSatellite*, and *endPoint*
8: *distanceLimitMax* ← maximum distance limit from *currentSatellite* to *endPoint*
9: *distanceToGround* ← distance from *currentSatellite* to *endPoint*
10: **while** *distanceToGround* > *distanceLimitMax* **do**
11: add *currentSatellite* to *satellitePoints*
12: remove *currentSatellite* from *totalSatellites*
13: *currentSatellite* ← *nextSatellite*
14: *distanceSorted* ← *n* closest satellites sorted by distance from *currentSatellite*

15: *nextSatellite* ← satellite with minimum angle between *currentSatellite*, *nextSatellite*, and *endPoint*
16: *distanceLimitMax* ← maximum distance limit from *currentSatellite* to *endPoint*
17: *distanceToGround* ← distance from *currentSatellite* to *endPoint*
18: *yield*
19: **end while**
20: **if** *satellitePoints* has more than one item **then**
21: *renderSatellites* ← *satellitePoints*
22: **end if**
23: *yield*
24: **end function**

While the functional implementation can be found in Section 6.3, the next section we will cover is the requirements to run the visualisation software.

Chapter 5

Methods Used

5.1 Requirements and Specifications

This section explores the requirements and specifications that underpin the functionality of the simulation software discussed in this dissertation. The requirements section encompasses the technology and tools needed for building the simulation, and outlines the hardware and software prerequisites for end users. The specifications section outlines what the software should actually do.

5.1.1 Requirements

The development of an interactive graphical simulation to visualise the algorithm requires the use of specific tools and frameworks. This subsection focuses on the primary tool, Unity Engine, chosen to implement the simulation and explains the rationale behind its selection.

5.1.1.1 Unity Engine

Unity is a cross-platform software framework that enables the development of both three-dimensional and two-dimensional applications [24]. Although Unity is primarily used as a game engine, it is also used for other interactive experiences, such as films, architecture, and engineering projects. In this dissertation, Unity Engine will be used to create an interactive graphical simulation of the algorithm proposed in this paper.

Unity was initially released in 2005 for Mac's OSX operating system, but it quickly expanded its reach to include support for development on Microsoft Windows. The engine has a user-friendly interface and provides a primary scripting API

in C# using Mono [24]. This allows developers to build both the Unity editor in the form of plugins and applications themselves through use of a single language.

Unity can be used for projects of all sizes and complexities - from a global phenomenon like Pokemon Go [25] to the indie drawing tool Pixel Studio [26].

The decision to use Unity as the platform for the project was made after careful consideration of various factors that align with the project's requirements and objectives. Unity emerged as the most suitable choice due to its versatility, real-time simulation capabilities, user-friendly interface, and the ability to create an interactive and usable environment for the intended users. This section discusses the reasons behind selecting Unity and elaborates on its key features and benefits for the simulator software:

Versatility and 3D Workspace Unity is known for its versatility, especially when it comes to creating 3D applications and simulations. As the project involves modeling and simulating a satellite network in three-dimensional space, Unity's native support for 3D graphics and physics made it a suitable choice. The platform provides a suitable environment to design and manipulate 3D objects, making it well-suited for accurately representing satellite positions, orbits, and interactions in a realistic manner. It can handle rendering a large number of satellites simultaneously, and this capability ensures that the software can accurately portray the dynamics of a satellite network and facilitate in-depth analysis.

Real-Time Simulation Real-time simulation is a critical requirement for this project, as it allows users to observe and analyze the behavior of the satellite network as it unfolds. Unity's real-time rendering engine ensures that the simulation runs smoothly, providing users with instant feedback on any changes and allowing them to make adjustments as needed. This capability is particularly important when exploring the routing algorithm as it enables users to witness the immediate impact of their decisions in a dynamic environment.

User-Friendly Interface The ease of use and accessibility of the platform were essential considerations for this project. Unity offers an intuitive and developer-friendly interface that simplifies the process of creating interactive simulations.

Creation of Usable Interface for Users Aside from its developer-friendly environment, Unity also does well in enabling the creation of a usable interface for end users as well. The simulation software incorporates interactive elements such

as menus that allow users to interact with the satellite network simulation effectively. Users can easily access and manipulate parameters and observe simulation results using the virtual environment. Additionally, Unity's compatibility with multiple platforms ensures that the software can be deployed on various devices, further enhancing accessibility for users across different systems, though currently the simulation software is only built for Windows.

What to Expect from the Software By leveraging the capabilities of Unity enables of a final software product that offers an intuitive and immersive simulation for researchers. They are able to use a three-dimensional workspace that accurately represents satellite positions and orbits in real time. The interactive interface will ultimately allow users to explore different network configurations, and traffic scenarios while observing the immediate impact of their choices. Furthermore, the software provides data output to aid users in understanding the performance and efficiency of the satellite network under various conditions.

5.1.2 Other requirements

End user platforms must meet the following requirements:

- Access to a formatted dataset from simplemaps, which provides world locations. This is included with the software build in the StreamingAssets folder.
- Operating System: Windows 7, 8, 10 or 11 (64-bit versions only).
- CPU: X64 architecture with SSE2 instruction set support.
- Graphics Card: DirectX 10 (shader model 4.0) capabilities.
- Hardware: The system should have hardware vendor officially supported drivers installed.

5.1.3 Specifications

The program is designed to visualise the orbit of LEO satellites, and has a number of features that enable ease of use and generally a more optimised experience. By default, it uses a StarLink satellite constellation - however, this can be changed by modifying the "starposA" file in the StreamingAssets folder of the software.

The software is able to find paths between any two points of the constellations coverage area, and should not result in routing loops. From a functional point of

view, the graphical rendering of the algorithm can be rendered in both real time and accelerated time, on top of rendering smoothly, and the Earth and satellites rotate and orbit appropriately and in a semi-realistic manner.

Ground station locations are rendered from a file providing city lists, and is elaborated on more in the next chapter.

5.1.3.1 Development Device

As a result of the relatively underpowered machine used for the development of the simulation software, the program itself is quite optimised in terms of requirements. Visualisation has been optimised through removal of shadows, texture filtering and use of Unity Engine's native particle system to render cities and satellites in large numbers without impacting the speed of the simulation. Further decisions that were made to improve performance can be found in Section 6.3.1. In terms of hardware, matching or exceeding the specifications of the development device should allow for a smooth experience.

The following details are of the machine used for development:

```
------------------
System Information
------------------
             Machine name: DESKTOP-Q9LVBDF
               Machine Id:
                  {4843BDB9-2DCF-4572-B743-34AC803326B2}
         Operating System: Windows 10 Pro 64-bit (10.0,
            Build 19044) (19041.vb_release.191206-1406)
                 Language: English (Regional Setting:
                    English)
      System Manufacturer: Acer
             System Model: TravelMate P259-G2-M
                     BIOS: V1.32 (type: UEFI)
                Processor: Intel(R) Core(TM) i3-7130U CPU @
                   2.70GHz (4 CPUs), ~2.7GHz
                   Memory: 12288MB RAM
      Available OS Memory: 12156MB RAM
                Page File: 7485MB used, 6526MB available
              Windows Dir: C:\Windows
          DirectX Version: DirectX 12
      DX Setup Parameters: Not found
         User DPI Setting: 96 DPI (100 percent)
       System DPI Setting: 96 DPI (100 percent)
          DWM DPI Scaling: Disabled
                 Miracast: Available, with HDCP
Microsoft Graphics Hybrid: Not Supported
 DirectX Database Version: 1.0.8
           DxDiag Version: 10.00.19041.0928 64bit Unicode
```

Now that we've covered the algorithm and requirements, the next section explores how it is functionally implemented.

Chapter 6

Implementation

6.1 Reading Data for Cities and Satellites

6.1.1 City Structure

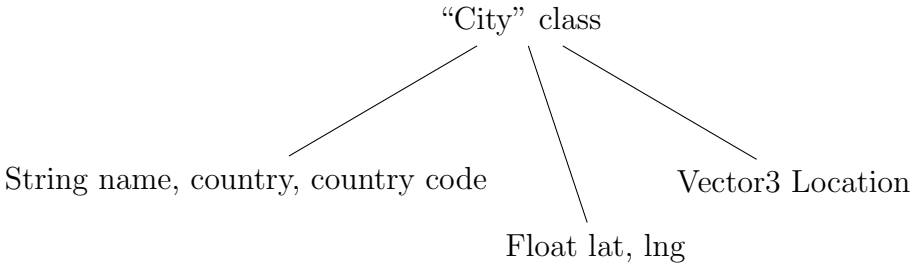

Figure 6.1: The structure of the city object

The structure of the city objects that cover the globe is defined in figure 6.1.1.

```
public struct city
{
    public string name, country, code;
    public float lat, lng;
    public Vector3 location;
}
```

The list of cities is a .csv file that has been modified with a PHP script to strip it of cities that are too close to each other - this allows for an even spread of cities around the globe regardless of size. An example of the file formatting is as follows, using the first entry of the list:

Literal appearance:
“Farah”, “32.3436”, “62.1194”, “AF”, “Afghanistan”
Explanatory appearance:

City Name	Latitude	Longitude	Country Code	Country Name
“Farah”	“32.3436”	“62.1194”	“AF”	“Afghanistan”

6.2 Conversion Between Coordinate Systems

To be able to use the available dataset, conversion was required; the datasets for cities and the dataset for satellites used the Geographic coordinate system, while Unity uses the Cartesian coordinate system. This section refers to the background knowledge of Section 2.2.

```
location = Quaternion.AngleAxis(longitude, -Vector3.up) *
   Quaternion.AngleAxis(latitude, -Vector3.right) * new
   Vector3(0, 0, (1274.2f / 2) + 2);
```

The function Quaternion.AngleAxis generates a rotation that revolves the first variable by x degrees around the second axis. The Vector3.up class serves as an abbreviation for Vector3(0, 1, 0), and Vector3.right is a shorthand notation for Vector3(1, 0, 0). Unity Engine provides this function and two functions by default to enhance code readability.

Because the program is structured to conform to a 1:10000 proportion of Unity's inherent unit (one meter), the computation of Earth's diameter results in 1274.2 meters, as opposed to an approximate 12742 kilometers [27]. An extra offset of two units is included to reposition the point that symbolizes the city (seen at the end of the calculation), ensuring better visibility atop the three-dimensional Earth model.

The outcome is that the location of each city is translated into a point in three-dimensional space, rather than being positioned relative to the globe.

6.2.1 Satellite Structure

Much like the cities, satellite data is read from a file included in the program. Said file, “starposA.json”, is from `https://satellitemap.space/` and features a variety of data on the satellites in the StarLink constellation. The main part is the orbital parameter; it contains the current location of the satellite and a future position of the satellite (lat and lat2, etc) and from this we can calculate the orbital plane of the satellite using the following code, demonstrated in the “SatelliteScript.cs” file.

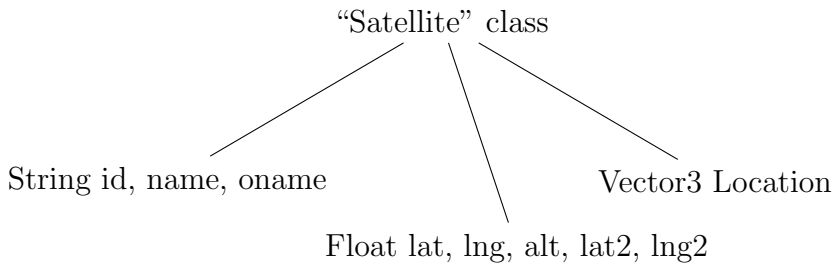

Figure 6.2: The structure of the satellite object

```
        satData = satDataIn;
        Vector3 futurePosition =
           Quaternion.AngleAxis(satData.lng2, -Vector3.up) *
           Quaternion.AngleAxis(satData.lat2,
           -Vector3.right) * new Vector3(0, 0, (satData.alt
           / 10) + (1280 / 2));
        Vector3 A = Vector3.zero;
        Vector3 B = transform.position;
        Vector3 C = futurePosition;
        Vector3 direction = Vector3.Cross(B - A, C - A);
        upVectorOfOrbit = direction / direction.magnitude;

        float revPerDay = 15;
        float revPerSecond = revPerDay / 240;
        speed = revPerSecond;
```

These variables are then applied to a satellite for every frame of the "Update" function to rotate it around the Earth at a speed of fifteen full orbital rotations each day.

```
    private void Update()
    {
        transform.RotateAround(Vector3.zero,
           upVectorOfOrbit, speed * speedMultiplier *
           Time.deltaTime);
    }
```

6.3 Pathfinding Implementation

The implementation is done through an iEnumerator function in Unity Engine, which enables it to run asynchronously from the rendering aspect of the program. This is important as the simulation software renders satellites and routes in real time, and waiting on calculating the route each frame is computationally expensive. As it stands, the device that was used for development (see 5.1.3.1) has assisted in improving and optimising the algorithm in question, though optimisations can only go so far.

The actual implementation below is the functional component of the pseudocode shown in Section 4.1. In terms of functionality, variables and object names have been chosen for maximum readability.

```
IEnumerator findRoute()
{
    print("calculating␣route");
    city startPoint = startCities[startDropdown.value];
    city endPoint = endCities[endDropdown.value];
    List<SatelliteScript> satellitePoints = new
       List<SatelliteScript>();
    totalSatellites =
       FindObjectsOfType<SatelliteScript>().ToList();
    SatelliteScript currentSatellite =
       totalSatellites.OrderBy(t =>
       Vector3.Distance(t.transform.position,
       startPoint.location)).ToArray()[0];

    totalSatellites.Remove(currentSatellite);

    SatelliteScript[] distanceSorted =
       totalSatellites.OrderBy(t =>
       Vector3.Distance(t.transform.position,
       currentSatellite.transform.position)).Take(nClosest)
    .ToArray();
    SatelliteScript nextSatellite = distanceSorted.OrderBy(t
       => calcAngle(currentSatellite.transform.position,
       t.transform.position,
       endPoint.location)).ToArray()[0];
```

```
    float distanceLimitMax =
       Mathf.Abs((currentSatellite.satData.alt / 10) /
       Mathf.Sin(40 * Mathf.PI / 180));
    float distanceToGround =
       Vector3.Distance(currentSatellite.transform.position,
       endPoint.location);

    while (distanceToGround > distanceLimitMax){
        yield return null;
        satellitePoints.Add(currentSatellite);
        totalSatellites.Remove(currentSatellite);
        currentSatellite = nextSatellite;
        distanceSorted = totalSatellites.OrderBy(t =>
           Vector3.Distance(t.transform.position,
           currentSatellite.transform.position)).Take(nClosest)
        .ToArray();
        nextSatellite = distanceSorted.OrderBy(t =>
           calcAngle(currentSatellite.transform.position,
           t.transform.position,
           endPoint.location)).ToArray()[0];

        distanceLimitMax =
           Mathf.Abs((currentSatellite.satData.alt / 10) /
           Mathf.Sin(40 * Mathf.PI / 180));
        distanceToGround =
           Vector3.Distance(currentSatellite.transform.position,
           endPoint.location);
    }
    if (satellitePoints.Count > 1){
        ///renderPath(startPoint, endPoint, satellitePoints);
        render = satellitePoints;
    }
    yield return null;
}
```

6.3.1 Performance Considerations

While the simulator effectively visualizes pathfinding outcomes, it's worth mentioning that its performance is influenced by factors such as network size and available computational resources. Larger-scale networks may demand optimisation efforts to ensure real-time responsiveness during simulation.

Because of the lack of said computational resources during development, a number of decisions were made to reduce demand - particularly in regards to graphical requirements. This included the implementation of a city selection menu on starting up the program, as the sheer number of cities, rendered using Unity Engine, caused significant slowdowns when rendered all at once.

Finally, we move onto the conclusion.

Chapter 7

Conclusions

7.1 Results

In this section, we provide an overview of the simulator's current functionality and its output. The simulator was developed to visualize the pathfinding algorithm's behavior within a Low Earth Orbit satellite network.

7.1.1 Simulator Functionality

The simulator offers the following core functionalities:

- Network Visualization: The simulator renders a graphical representation of the satellite network, displaying satellites as nodes and potential communication links as edges.
- Pathfinding Algorithm Integration: The implemented pathfinding algorithm is demonstrated in the simulator. Users can run pathfinding simulations and observe the computed paths and output.
- Configurable Parameters: Certain parameters, such as simulation speed and the n closest satellite positions, can be adjusted through the interface. Countries can be configured through the configuration file.

7.1.2 Output and Visualization

To illustrate the simulator's output, we present two visualizations showcasing the algorithm's performance under distinct scenarios.

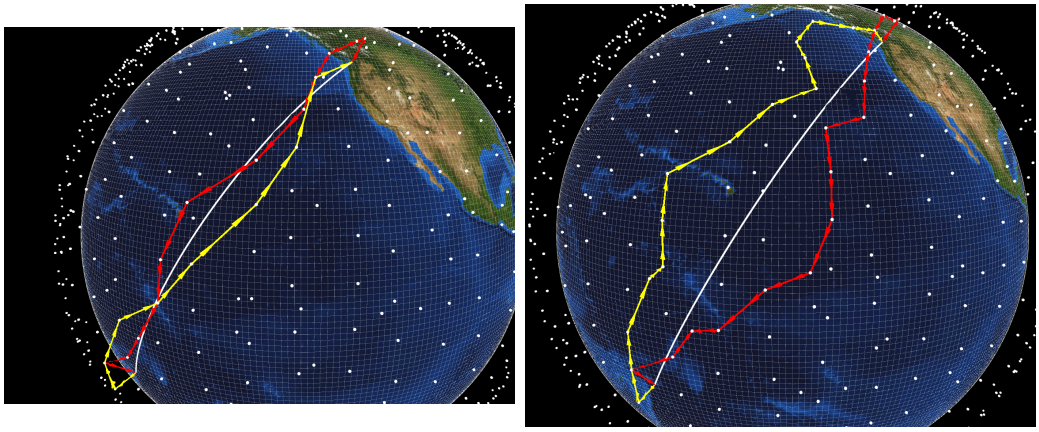

Figure 7.1: Scenario 1: Pathfinding, n=10 (left) vs. n=5 (right). It can be observed that with a higher n value, there are much fewer hops and an overall smoother path.

In Figure 7.1, the simulator displays a scenario where the network is devoid of obstacles or inaccessible satellites. The algorithm calculates efficient paths between satellites, prioritizing direct communication links where possible. In this example, the start and end locations are Seattle and Pago Pago, from the United States of America and American Samoa respectively.

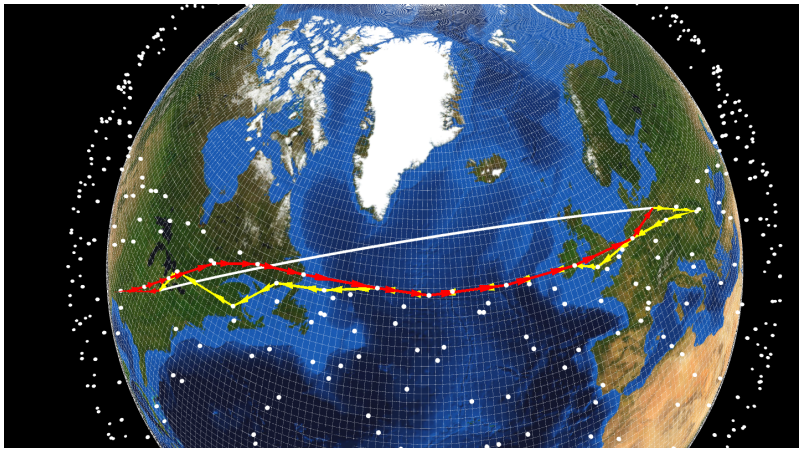

Figure 7.2: Scenario 2: Pathfinding around the poles, n=10. The path tags along the maximum latitude of satellites, found through the inclination.

Figure 7.2 depicts a more complex scenario where certain areas are somewhat inaccessible due to the inclination of satellites avoiding the poles. This route goes between Akron, United States to Berlin, Germany. The algorithm adapts to this constraint where the great circle path goes over the polar region where there are few satellites.

These visualizations highlight the simulator software's ability to offer insights into the algorithm's behavior across diverse satellite conditions. However, it's important to note that these results represent only a small amount of the potential scenarios that can be simulated.

7.2 Problems Left Unsolved

7.2.1 Simulator Features

A number of planned features would be useful should they be implemented into the simulation software. These unimplemented features included:

- The ability to dynamically choose different pathfinding algorithms.
- Enabling/disabling satellites - toggling within the simulator to simulate scenarios where specific satellites are temporarily inaccessible.
- Integrating weighted routing preferences directly into the algorithm, such as transmission caps or congestion levels.
- An expanded configuration file with options for variables such as line rendering width and other visual parameters.
- A broader optimization effort to enhance the program's performance and scalability.

7.2.2 Future Direction

The aforementioned features provide a pathway for future developments of this project. Addressing these gaps holds the potential to unlock further insights in the field of LEO satellite routing.

Bibliography

[1] *Orbital plane*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Orbital_plane.

[2] *Low Earth orbit*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Low_Earth_orbit.

[3] *Satellite Internet access*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Satellite_Internet_access.

[4] *Satellite constellation*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Satellite_constellation.

[5] *Laser communication in space*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Laser_communication_in_space.

[6] *Optical communication*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Optical_communication.

[7] *Satellite communication*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Communications_satellite.

[8] *Backhaul*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Backhaul_(telecommunications).

[9] *Starlink*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Starlink.

[10] *Jonathan's Space Pages*, 2023. [Online]. Available: https://planet4589.org/space/con/star/stats.html.

[11] *FCC OKs lower orbit for some Starlink satellites*, 2019. [Online]. Available: https://spacenews.com/fcc-oks-lower-orbit-for-some-starlink-satellites/.

[12] *Geographic Coordinate System*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Geographic_coordinate_system.

[13] *Cartesian Coordinate System*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Cartesian_coordinate_system.

[14] *Simplemaps World Cities*, 2022. [Online]. Available: https://simplemaps.com/data/world-cities.

[15] *Great-circle navigation*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Great-circle_navigation.

[16] *Bellman-Ford algorithm*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm.

[17] B. Furch, Z. Sodnik, and H. Lutz, *Optical Communications in Space – a Challenge for Europe*, en, Jan. 2002. DOI: 10.1078/1434-8411-54100102. [Online]. Available: http://dx.doi.org/10.1078/1434-8411-54100102.

[18] I. F. Akyildiz, E. Ekici, and M. D. Bender, "MLSR: A novel routing algorithm for multilayered satellite IP networks," *IEEE/ACM Transactions on networking*, vol. 10, no. 3, pp. 411–424, 2002.

[19] X. Song, K. Liu, J. Zhang, and L. Cheng, "A novel unicast routing algorithm for LEO satellite networks," eng, in *Proc. SPIE*, vol. 5985, Bellingham (Washington): SPIE, 2005, 59851P-59851P–5, ISBN: 0819460079.

[20] M. Roth, H. Brandt, and H. Bischl, *Implementation of a geographical routing scheme for low Earth orbiting satellite constellations using intersatellite links*, en, Jun. 2020. DOI: 10.1002/sat.1361. [Online]. Available: http://dx.doi.org/10.1002/sat.1361.

[21] C. Li, Y. Zhang, Z. Cui, *et al.*, "An Overview Of Low Earth Orbit Satellite Routing Algorithms," in *2023 International Wireless Communications and Mobile Computing (IWCMC)*, 2023, pp. 866–870. DOI: 10.1109/IWCMC58020.2023.10182766.

[22] V. Gounder, R. Prakash, and H. Abu-Amara, "Routing in LEO-based satellite networks," in *1999 IEEE Emerging Technologies Symposium. Wireless Communications and Systems (IEEE Cat. No.99EX297)*, 1999, pp. 22.1–22.6. DOI: 10.1109/ETWCS.1999.897340.

[23] Y. Zhu, L. Qian, L. Ding, F. Yang, C. Zhi, and T. Song, "Software defined routing algorithm in LEO satellite networks," in *2017 International Conference on Electrical Engineering and Informatics (ICELTICs)*, 2017, pp. 257–262. DOI: 10.1109/ICELTICS.2017.8253282.

[24] *Unity Engine*, 2023. [Online]. Available: https://unity.com/.

[25] *Pokemon GO*, 2016. [Online]. Available: `https://www.pokemon.com/us/app/pokemon-go/`.

[26] *Pixel Studio*, 2023. [Online]. Available: `https://play.google.com/store/apps/details?id=com.PixelStudio`.

[27] *Earth*, 2023. [Online]. Available: `https://en.wikipedia.org/wiki/Earth`.